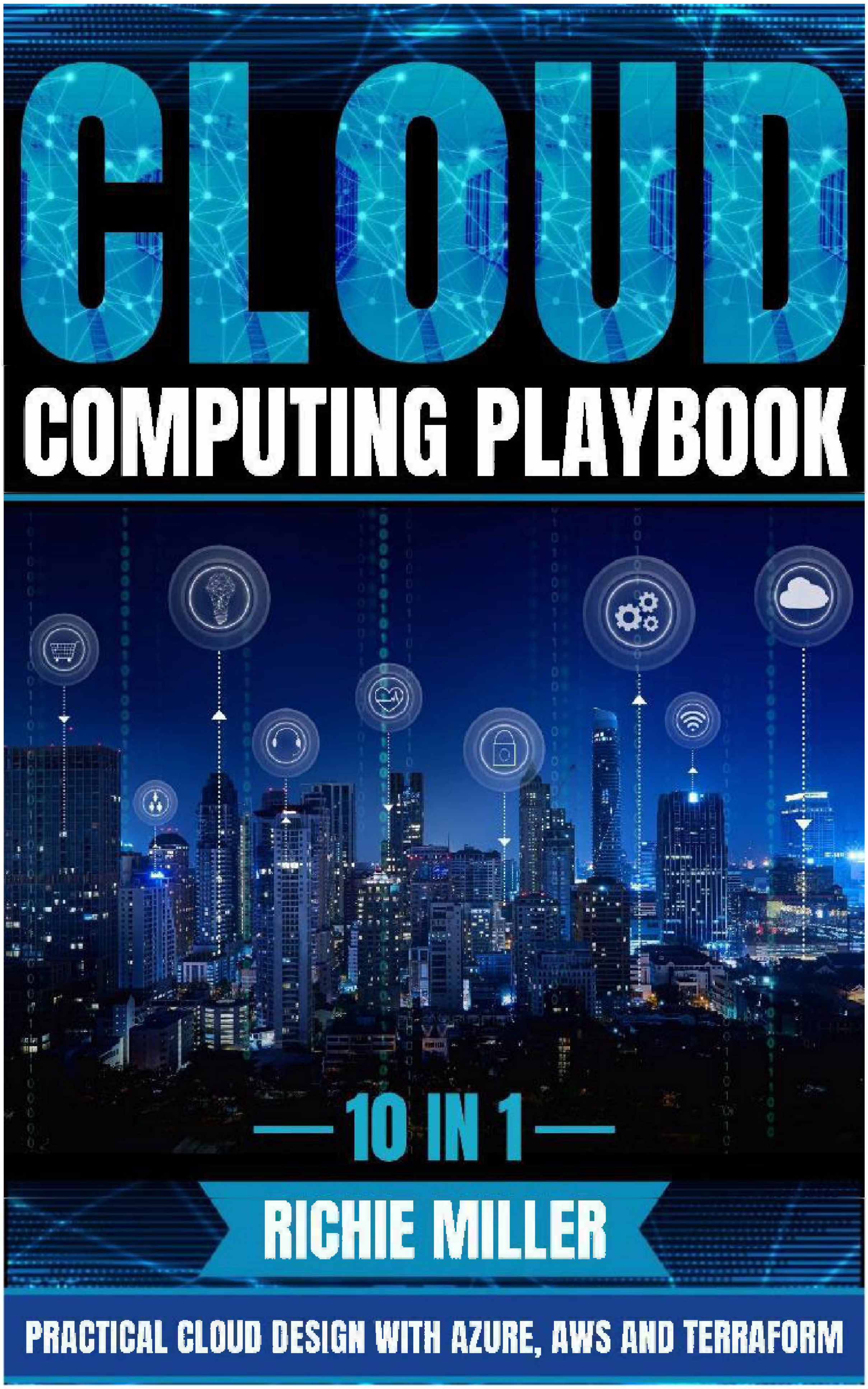
CLOUD COMPUTING PLAYBOOK



10 IN 1

RICHIE MILLER

PRACTICAL CLOUD DESIGN WITH AZURE, AWS AND TERRAFORM

CLOUD COMPUTING PLAYBOOK

10 IN 1

PRACTICAL CLOUD DESIGN WITH AZURE

AWS AND TERRAFORM

BOOK 1

CLOUD COMPUTING FUNDAMENTALS

INTRODUCTION TO MICROSOFT AZURE AZ-900 EXAM

BOOK 2

MICROSOFT AZURE SECURITY AND PRIVACY CONCEPTS

CLOUD DEPLOYMENT TOOLS AND TECHNIQUES, SECURITY & COMPLIANCE

BOOK 3

MICROSOFT AZURE PRICING & SUPPORT OPTIONS

AZURE SUBSCRIPTIONS, MANAGEMENT GROUPS & COST MANAGEMENT

BOOK 4

MICROSOFT AZURE AZ-900 EXAM PREPARATION GUIDE

HOW TO PREPARE, REGISTER AND PASS YOUR EXAM

BOOK 5

AWS CLOUD PRACTITIONER: CLOUD COMPUTING ESSENTIALS

BOOK 6

AWS CLOUD COMPUTING: INTRODUCTION TO CORE SERVICES

BOOK 7

AWS CLOUD SECURITY: BEST PRACTICES FOR SMALL AND MEDIUM BUSINESSES

BOOK 8

TERRAFORM FUNDAMENTALS: INFRASTRUCTURE DEPLOYMENT ACROSS MULTIPLE SERVICES

BOOK 9

AUTOMATION WITH TERRAFORM: ADVANCED CONCEPTS AND FUNCTIONALITY

BOOK 10

TERRAFORM CLOUD DEPLOYMENT: AUTOMATION, ORCHESTRATION, AND COLLABORATION

RICHIE MILLER

Table of Contents – Book 1

Table of Contents – Book 2

Table of Contents – Book 3

Table of Contents – Book 4

Table of Contents – Book 5

Table of Contents – Book 6

Table of Contents – Book 7

Table of Contents – Book 8

Table of Contents – Book 9

Table of Contents – Book 10

BOOK 1

CLOUD COMPUTING

FUNDAMENTALS

INTRODUCTION TO

MICROSOFT AZURE AZ-900 EXAM

RICHIE MILLER

Introduction

In the following chapters, we're going to talk about the exam and how to prepare for it. We will first cover the benefits of getting Azure certified and why you should consider getting the certification. We will then do an overview of the certification exam, what you will be evaluated on, as well as learn about the skills measured document. Finally, we will do an overview of our learning materials for this exam. By the end of this book, you will know what's needed to start studying for the Microsoft Certification exam. Let's start by learning what are the benefits of getting Azure certified and, more particular, why the exam. Let's start by asking, why do we even want to get a Microsoft certification? First of all, Microsoft certifications can really help give you a professional advantage by providing globally recognized and evidence of mastering skills in a digital and cloud business. If we look at the numbers, according to multiple studies, 91% of certified IT professionals say that certification gives them more professional credibility, 93% of decision makers agree that certified employees provide more added value, and 52% of certified IT professionals say that their expertise is more sought after within the organization after getting certified. Honestly, certifications can also help you in the financial aspect of your career. 35% of technical professionals say that getting certified led to salary or wage increases, while 26% of technical professionals reported job promotions after getting certified. Now let's get a bit more and talk why we want to get Azure certified. Microsoft Azure is one of the top cloud providers in the world for Infrastructure and Platform as Service workloads. 63% of enterprises in the world are currently running apps on Microsoft Azure, second only to AWS. However, 19% of enterprises expect to invest significantly more on Azure in 2022, and this is leading all of the other cloud vendors this year, so Azure is still growing at an astonishing rate. Finally, 44.5% of enterprises say that Microsoft Azure is their preferred provider for cloud business intelligence. If we get more specific into our Azure certification, the Azure certification portfolio is actually the biggest certification portfolio at Microsoft, and it includes 3 certifications, 10 certifications, 2 certifications, and 3 certifications. But what makes the unique? The Azure Fundamental Certification is an optional, but very highly recommended prerequisite for all of the other Azure certifications. It's the place you should start whether you have done a Microsoft certification before and now you want to specialize in Azure, or if this is your first ever Microsoft certification, the is where you should begin your Azure certification journey.

Chapter 1 AZ-900 Exam Summary

Now that we know why the is a very important and valuable exam, let's do an overview of the exam. The Azure Fundamental Certification is an opportunity to prove knowledge of cloud concepts, Azure services, Azure workloads, security and privacy in Azure, as well as Azure pricing and support. From an audience point of view, the is intended for candidates who are just beginning to work with solutions and services or are new to Azure. Also, as this is a fundamentals exam, before starting to study, candidates should be familiar with general technology concepts, but really no other requirements as you will learn the fundamentals in the study material for this exam. If we take a look at the basics, the exam costs 99 USD; however, the price might vary depending on your region. Furthermore, I highly encourage you to check with your manager or HR person as organizations will often reimburse the cost for learning and certifications. The worst that they can say is no, so it's always worth to ask. Also, something that is really nice is that fundamental certifications do not expire. For example, associate and Microsoft certifications expire after one year, but because this is a exam, it doesn't expire, so that is nice. Lastly, if you're a student, you can actually get college credit for passing Microsoft exams and earning Microsoft certifications. This works mostly in the United States, not internationally. If we take a look at the skills measured, it's split up into six categories, the first one being describe cloud concepts, which is 20 to 25% of the exam. We then have describe core Azure services, which is 15 to 20% of the exam. Third, we have to describe core solutions and management tools on Azure, which is 10 to 15% of the exam. Fourth, we have to describe general security and network security features, which is between 10 and 15% of the exam. Our fifth category is describing identity, governance, privacy, and compliance features, which is 20 to 25% of the exam. Finally, describe Azure cost management and agreements, which is between 10 to 15% of the exam. There is one keyword that is repeated throughout each objective, and that is the verb describe. Verbs are very important in Microsoft certifications. So, what does the word describe mean? As we're talking about the exam, the verb describe tells us that you do not need to know how to configure, manage or implement features. What you really need to know is what features are available and what business problems they solve. The goal of this exam is for you to be able to know what cloud computing challenges can be solved by what Azure solution. If you talk with someone and they say we have this business need for a workload, you need to be able to know, this Azure solution can help you with that. This is why the is an amazing exam for anyone working in the Microsoft ecosystem. Whether you're an IT pro, dev, project manager or business stakeholder, knowing what solutions Microsoft offers can really allow you to better understand the projects you're working on and to propose the right solution at the right time. Now that we have talked about the objectives, Microsoft also provides a document called the skills outline, or the skills measured document, and it's important to review it before and after studying for this exam. The skills outlined are the full detailed list of everything that you need to know for the exam. We will review it later, but really this should be our checklist of things to study for the exams.

Chapter 2 Skills Measured Document

If you open up your browser and type this page will be one of the first choices.

This is the Microsoft Learning page for the Microsoft Azure Fundamentals exam. On the exam page at the top, you will see the description and audience for the exam, you will be able to schedule it, but what we want to talk about is the skills measured.

You will see on the exam page, you only have the skills; however, it's important that you click this link, Download exam skills outline. If you click on it, it will open up a PDF, either it will download it or open up directly in the browser depending on your settings.

Something that is really important, because as the cloud always changes, so do Microsoft certification exams, so you might see at the top a warning like this one. This exam was updated on November 9, 2020, and if you go to the bottom, at the bottom you will have kind of a document with tracked changes on, so you can see what were the changes that were done on the date that it was changed. And Microsoft generally also announces at least one or two months in advance if a change will happen, and it will be shown the exact same way, simply the date will be on the future so you can see if Microsoft will change the objectives.

If you go to the top, you have the Audience Profile again, same thing, but what gets interesting is that for each exam objective, so let's say Describe Cloud Concepts, it's broken down into and details. Under Describe Cloud Concepts, we have Identify the benefits and considerations of using cloud services, identify the benefits of cloud computing such as high availability, scalability, elasticity, agility, and disaster recovery. Then we have to describe the differences between the categories of cloud services. So you need to know what's the difference between Infrastructure as a Service and Platform as a Service and Software as a Service and then identify a service type based on a use case, so, you need to be able to know what workloads should go where. Then for each objective really, you have all of the details on what services do you need to know, what are the different things you should be able to describe. You can either save it locally or you can even print it and then use a highlighter, once you feel confident you learned something, highlight it, and this should be the checklist for your exam. You need to be able to go in the details in the skills measured document and then be able to say that all of the different tools, services, and concepts in here, I'm able to describe. It’s an important tool for your study to pass the exam.

Chapter 3 Why Use Microsoft Azure

We're going to look at a lot of individual services within Azure throughout the book, but in this chapter, I want to give you a broad overview of what Azure can do and how it's structured. I'm going to demystify Azure for you and give you the bigger picture of the environment that all the individual services operate in before we go into many of those services later on. But first, I want to talk about the Azure Fundamentals certification. If you're studying for the exam, this book will definitely help you do that, but this book doesn't encompass all of the exam objectives. If you're studying for the exam, I encourage you to read the most study guide provided by Microsoft because it does change from time to time and then map those objectives to the topics covered in this book. That way, you'll be able to see what else you need to learn outside of this book in order to pass the exam. That said, this is a book for people new to Azure, so we're going to start from the ground up by talking about why you would want to use Azure in the first place. Azure is a cloud platform with more than 200 products and services that help you create applications and solutions. The cloud platform part just means that Microsoft abstracts away all the underlying hosting infrastructure so you can rent basic things like web hosting, computing power, databases, and storage, as well as some really solutions, like business analytics tools, artificial intelligence services, and portals for managing devices for the Internet of Things. You might never use some of those advanced tools, but they give you options you probably didn't have at least not without installing a bunch of software and services on your own servers to do those things. But even if you just host websites and file shares, traditional things that every organization does, why would you want to use Azure? Well, there's a lot that goes into managing your own servers and datacenter. There's buying the physical hardware, storing those servers in a secure place where nobody can tamper with them, there's cooling needed because servers generate a lot of heat, and there's electricity and of course backup electricity unless you don't mind your applications being unavailable during a power outage, plus all the networking components and monitoring for health, as well as to make sure that no one hacks your network and computers. But there's also less obvious things, like you might have a need for a lot of computing power at certain times of the day, week or year, so you need the servers to be able to handle that load. Let's say you're hosting an ecommerce application to sell your company's products and there's way more traffic around Christmas than during the rest of the year. You need your servers to be able to handle the load, but they sit underutilized the rest of the year. That's a waste of money and hard drives can fail or you might need to keep increasing your storage because the business groups keep generating more files. They tend to do that. Then they're storing backups of files and databases. What about disaster recovery? If there's a major outage at your datacenter, are you okay with the apps not being available or do you want to maintain another datacenter in another location that can take over that traffic? There's also the ongoing maintenance of the operating systems on those servers. They need to be patched and monitored for threats.

Then every five years or so, you need to replace all that hardware, not to mention the networking components like routers, switches, and firewalls. Microsoft, Azure and cloud computing in general was created to address many of these issues. For the rest of this book, we'll look at services in Azure for hosting applications and data and all the virtual infrastructure that allows you to do that. You'll see how easy it is to create and configure that infrastructure in Azure without having to manage any physical hardware like you do We'll start with signing up for a subscription in Azure. That'll give you a way to follow along and create your own Azure services. Then you'll start to learn how Azure is implemented using regions, how those regions are connected, and all about Azure datacenters in the regions. You'll learn about resource groups, which are the containers for holding multiple resources that make up a logical grouping, like for an application. Then we'll explore the Azure portal, which is the main way you'll be interacting with Azure and managing instances of services that you create. We'll create some resources in the portal, and I'll show you Azure Active Directory, which is the identity service in Azure for managing user accounts, and it provides the foundation for access control for managing Azure, as well as access control for the people using the applications that you deploy to Azure. Finally, we'll tie a lot of the Azure concepts together by discussing how Azure Active Directory and subscriptions are related. So let's get started by signing up for an Azure subscription next.

Chapter 4 How to Create an Azure Subscription

Let's create an Azure subscription we can use to explore the Azure portal. We're going to create a free trial account at azure.microsoft.com/free. For verification, we'll need three things, a Microsoft account, and I'll explain that more shortly, you'll need a phone number where a verification code can be sent, and you'll need a credit card. The card won't get charged, not unless you manually upgrade the account to a account. If you already have a account, you can just use that instead, but you will get charged for everything you create. Let's go to azure.microsoft.com/free.

If you've never signed up for a free account, you can do that and get 200 USD credit for 30 days to use for creating resources in Azure. In addition to that, there are certain services that are free for 12 months and other services that are always free, but they're pretty limited in functionality. If we scroll down a bit, this page describes some of the things we can do in Azure, like hosting web applications using Azure App Services, using Azure Machine Learning, creating Azure Virtual Machines or containers, and serverless options like Azure Functions and Azure Logic Apps, which let you build workflows with tons of connectors to services inside and outside of Azure.

We'll talk about all these services later on. Further down, it says that if we upgrade this free trial to a account where our credit card can get billed for usage, we'll be entitled to some free services, like 750 hours of running a Linux or a Windows virtual machine, a Azure SQL Database, and 5 GB of blob storage.

We'll talk about storage later on too. But now, let's scroll down and click Start free. The first thing we need is either a Microsoft email address or a GitHub account.

If you're not aware, Microsoft acquired GitHub in 2018, which is why you can use your GitHub identity to create an Azure account. If you're using an email address, it needs to be a Microsoft one, so usually an Outlook, Hotmail or a Live account, but you can even use a phone number now to create a Microsoft account. The point is that you need that Microsoft account already in order to create an Azure account, but you could create the account from this link. I already have a Microsoft account, so I'll enter that email address here and I'll enter my password. I don't have multifactor authentication set up for this Microsoft account, but you could do that in which case you'd need to provide another factor of authentication when logging in, like a code that's generated in the Microsoft Authenticator app on your phone or a temporary code that's sent to you by text or phone call. Now I'm brought to the screen where I can fill in my information. Since I'm logged in, it picks up my name and I need to enter a phone number. This can be a cellphone or a home phone because you can choose to have the code sent to you by text or through an automated voice call. I'll choose Text me and I get a text on my cell phone with a code. So I'll enter that code here and click Verify code. Now I need to enter my address.

Now we need to agree to the customer agreement and privacy agreement. There are links here, of course, so feel free to read through those if you'd like before agreeing. I'll click Next and here's where you need to enter the credit card.

It says you won't be charged unless you move to pricing. You should use a credit card here that's never been used for a free trial account, otherwise you may get denied, which makes sense because you shouldn't be able to keep creating free trials in order to get credits to use Azure for free. So I'll enter my credit card info, and just click Sign up. Microsoft will verify your credit card, and once that's done you'll come to a screen that says you're approved and there's a link that will bring you to the Azure portal. So let's go to portal.azure.com, which is the administrative portal for Azure and the browser brings me into the portal.

Next, we'll explore the Azure portal a bit and see how to create a resource in Azure.

Chapter 5 How to Create Resources in Azure

I'm logged into the Azure portal at portal.azure.com. An Azure Active Directory instance gets created with this Azure trial, which means we can create individual users and assign them permissions, so you don't have to keep using this administrative account to log in and really you probably shouldn't because it has superuser privileges. It's a good idea to create an administrative account in Azure AD and use that instead. On this home page, you can access some Azure services, and there's a menu on the left that has shortcuts to some default services, like any Azure virtual machines you've created, SQL databases, and the Azure Active Directory associated with this subscription.

You can access the list of all services from this menu item. You'll see a tour of the Azure portal later, but let's just look at creating a resource. I'll choose the Virtual machines link here.

That brings us to a page where all the virtual machines that were created in this subscription are listed, and of course there aren't any yet.

Let's select the list to create one, and at this point we haven't selected whether this will be a Linux or a Windows VM. That's fine. What I really want to show you here are some of the mandatory inputs.

The first one is the subscription, and that's filled out automatically. And then there's the resource group that this virtual machine will belong to. Resource groups are basically a container that holds resources, and everything in Azure, including a virtual machine, is considered a resource. We'll talk about how resource groups are used for security and deployment purposes later on. The other thing is the region. You need to select the region that this virtual machine will be created in, which basically means the datacenter where it will exist. The list is pretty small, but that's because we're using a free trial account. Microsoft limits the datacenters available, so regions in high demand don't use too much capacity on free trials. But you could contact Microsoft support if you really want to create this VM in a region that's not listed here. I won't go through anything else on this page. We'll look at creating virtual machines in the next chapter. So, now you know that you'll need to choose a region when creating a resource. Let's talk about regions in Azure in more detail, next.

Chapter 6 What are Azure Regions & Availability Zones

Now let's talk about how Azure is physically implemented. You create services in Azure, like an Azure App Service for hosting a web app or a storage account for storing files. You can then deploy your applications and files to those services. That all gets hosted on Virtual Machines in Azure. Depending on the service you choose, you may have more or less access to those virtual servers for configuration. If you create a virtual machine, for example, you have full control. If you create an app service, you don't have direct access to the virtual machine. But the virtual servers in Azure are hosted on physical servers somewhere. That somewhere is an Azure datacenter. Azure datacenters are physical buildings located all around the world. At the time, there are over 200 Microsoft Azure datacenters worldwide. Each datacenter houses thousands of servers. There are about 4 million physical servers throughout the world. We're going to talk more about how datacenters are implemented later on too. Datacenters are located in regions. A region is a geographic location, often consisting of multiple datacenters. A region is what you choose when you create a resource. You decide which region you want your service created in. We'll talk about considerations in choosing a region in just a little bit. There are often multiple datacenters within a region, which helps in case a single datacenter becomes unavailable. But within certain regions, there's what's called availability zones. Availability zones are unique physical locations within a single region. There's a minimum of three separate availability zones in the region, and each availability zone is made up of one or more datacenters equipped with independent power, cooling, and networking. Some services like storage in Azure storage accounts will replicate your data automatically across all the zones in the region. Every region is located within a geography, which in Azure is a group of regions that define a boundary for data residency and disaster recovery. A geography is generally a single country, but it can be made up of multiple countries. Within a geography, there are region pairs available. Region pairs are datacenters that are generally at least 300 miles apart to reduce the impact on availability caused by a natural disaster or a major power outage. They're connected through a dedicated regional network. Regional pairs allow you to configure automatic replication and failover for certain Azure services, like when you choose georedundant storage for your Azure storage account. Azure automatically makes copies of your data across the regions in the region pair. For services that don't have options for failover like that, you can design your own strategy for failing over to another region if your primary region isn't available. Virtual machines are an example of this. You have to deploy duplicate virtual machines in another region yourself if you want to have them available for failover. This is called the shared responsibility model in Azure. The services are there for you, you just have to design the solution to take advantage of them. This page in the Azure documentation shows you the regions that are available in the different Azure geographies. You can see some geographies have more regions than others. And on this page in the docs, you can see the region pairs that are available. They're generally located in the same country, but not always. For example, at the bottom, it says that the Brazil South region is paired with the South Central US region. Next, let's talk about the factors that go into choosing an Azure region to deploy your resources to.

Now, let's talk about the factors that go into choosing an Azure region when you're creating resources in Azure. The first is proximity to users, and this has to do with performance. There are physical limitations to how fast data can travel around the world. If most of your users are located in Australia, for example, it doesn't make sense to host your website and database in a datacenter in the United States and have every request and response travel around the world, unless of course there are other reasons to choose that datacenter. One such consideration is that not all Azure services are available in all regions, especially when they're first released. You can go to this page in the Azure docs to see what services are available in which regions.

You can choose the regions and the service you're interested in or remove the filter and scroll through all the services to see what's available. Notice how there are services that are nonregional.

These are ones that don't require you to choose a region when you create them, like the Azure Bot Services. It's also possible that within a specific service, some features might not be available in the region closest to you. A great example of this are different sizes for virtual machines. On the virtual machine pricing page, if I scroll down, the region is selected as East US.

All the classes of VMs are shown below. The amount of cores, RAM, and temporary storage is shown for each.

So, there are basic VMs and there are specialized ones for things like compute tasks. I'll just search for a certain class of VMs. And they're shown here for East US, but if I change the region to South Central US, pricing is not available for this region because you can't create this class of VM there.

So that's a consideration when choosing a region if you have specific needs for certain services. Another reason you might choose one region over another is for regulatory or compliance reasons with regards to data residency. If you work in an industry that's highly regulated or your company has policies around where the data must reside, then you might need to choose your region based on that criteria. Now let's talk about how datacenters are connected together, and for this I want to show you this awesome interactive page on microsoft.com. This lets you choose a geography which tells you how many regions are available, and then you can choose the regions and see details like where it's located, the year it opened, how many availability zones the region has, some of the products available, information on disaster recovery options like region pairing, and standards that the datacenter complies with.

On this map, you can also see how the Microsoft global network is connected. There's over 165,000 miles of fiber optic and undersea cable systems that connect Azure datacenters around the world. When you access a resource in Azure, the traffic goes from your computer through your internet service provider to a point of presence, or PoP, that's managed by Microsoft where it enters into the Microsoft global network. Microsoft has over 185 of these PoPs around the world, so you get routed to the one closest to your location. You can click on these PoPs on the interactive map. PoPs are often placed within milliseconds of global population centers. Then, IP traffic stays on the Microsoft global network to access resources in Azure where it stays encrypted, flowing across the fiber and undersea cables to datacenter regions. You choose which regions to create your Azure resources in, so you can place your applications and data closest to where your users are, which helps keep the response time quick. Next, let's talk about Azure datacenters in more detail.

Chapter 8 Azure Data Centre Fundamentals

A Microsoft datacenter is a physical location that often looks like a bunch of warehouses. You can actually see a tour of a datacenter on microsoft.com. There's a virtual tour version as well that leads you through the areas of a datacenter. Each warehouse is big enough to store a commercial aircraft just to give you a sense of the size. Inside those warehouses are thousands of physical computers that host the virtual servers that you use when you create resources in Azure. The datacenter is built to withstand failures of individual components, so it has redundant networking, electricity, and cooling systems, as well as backup power sources. Because Microsoft hosts so many customers in their datacenters, you get the economy of scale of sharing those resources. Of course, your data is all separate from other customers and encrypted, and Azure datacenters undergo security reviews and have many industry certifications to ensure that your data is protected. Multilayered security is used to protect physical datacenters, infrastructure, and operations, and Microsoft has over 3500 cybersecurity experts monitoring activities in order to protect your business assets and data. So Microsoft is investing a lot more effort in security than any single customer could with their own datacenters. Microsoft used to be pretty secretive about their datacenters, but now they're more open about how their datacenters are structured and are working to standardize server and datacenter design through the Open Compute Project. Design specifications for server racks and server blades are being shared with the open source community through Project Olympus, similar to how software is made open source. Now let's talk about the energy needed to power a datacenter because this is a major consideration not only affecting cost, but affecting the environment, and that may be important to you when considering using Azure. Microsoft often chooses datacenter locations based on proximity to renewable energy sources. Microsoft enters into agreements with power companies to build wind and solar farms across thousands of acres of land. They build datacenters near hydroelectric dams and choose temperate locations so datacenters can be cooled by the outside air. There's even a project to convert waste heat from new datacenters in Finland into heating for cities. Heat is going to be transferred to customers through a system of insulated pipes for residential and commercial heating requirements. When it comes to backup power systems at datacenters, they are often powered by diesel fuel, but Microsoft is researching alternatives like synthetic fuels and hydrogen fuel cells. They plan to eliminate dependency on diesel by 2030. Let's look at the global infrastructure map again.

The yellow icons represent renewable energy projects that Microsoft is involved in for wind and solar. These are purchase agreements with third parties. There's a lot of research and innovation going on into datacenter design, and especially cooling. One interesting development is Project Natick, which is an underwater datacenter that was operated for five years off the coast of Scotland where servers were housed in a sealed container at the bottom of the ocean floor. That allowed servers to be cooled using the temperature of the ocean. Another interesting development is the Azure Modular Datacenter. This is a shipping container that allows for setting up an Azure datacenter in a remote area where cloud computing wouldn't have been possible. They use Azure Stack to create a private cloud. These modular datacenters can be used as a mobile command center for humanitarian assistance, for military missions, and to set up wherever high performance computing is needed, and they can run connected to the internet or disconnected. So there's lots of interesting things happening with regards to Microsoft Azure datacenters, and you can read about it for hours online. But the important thing is that Microsoft spends a lot of effort on optimizing the design of their datacenters. Next, let's talk about the resources you create in Azure and how they're logically organized using resources groups

Chapter 9 Resources and Resource Group Basics

Now let's talk about resources and resource groups in Azure. A resource is just a manageable item in Azure. Let's take a look at the Azure portal. The All resources menu item shows all the resources that have been created in this subscription.

This includes things like App Services for web apps, Storage accounts, there's a Log Analytics workspace where logs are stored from the various services, a Key vault where encryption keys and certificates are securely stored, and there's a Virtual machine here. When you create a virtual machine, other resources are created too, like a public IP address for the VM so it can be reached over the internet, a disk to hold the operating system, a virtual network that the VM is connected to, and a network security group that's used to secure the network. We'll look at all these elements later on, but the point is pretty much anything that can be configured in Azure is considered a resource, even when you think it might just be part of another resource. Each of the resources in this list were created in a location, which is an Azure region, and each resources part of a subscription. A resource group is a container that holds a set of resources that share the same lifecycle. In other words, you deploy, update, and delete them together. You can add and remove individual resources to and from a resource group as your solution evolves, but the general guidance is that if a resource needs to exist on a different deployment cycle, then it should be in another resource group.

Each resource you provision can only exist in one resource group. You can move a resource to another resource group if you need to, but it won't exist in both resource groups. Resources in different groups can communicate with each other. For example, you might have three different web applications being maintained by three different teams and they all exist in their own individual resource groups, but they all share a common database. That database can be in a completely different resource group and those web apps will still be able to use it. One of the main features of a resource group is that you can apply security controls to it for administrative actions, so you can assign reader roles to developers to be able to see what resources are in the resource group, but only administrators can make changes to the resource group. Resource groups allow you to leverage Resource Manager templates so you can deploy a set of resources using a JSON template and you can export a template from an existing resource group in order to deploy those resources in a repeatable way. This is great for moving a solution from a dev environment into a production environment, for example. When you create a resource group, you specify a region that it gets created in, but a resource group is just a container with metadata about the resources it contains, so the resource group can be created in a different region than the resources in the group. You can create a resource group during the creation of most resources, like when you're in the process of creating a new virtual machine. In that case, the resource group will get created in the same region that you specify for the virtual machine. You can also create the resource group by itself and then select it as the resource group to use when creating other resources. Next, let's explore the Azure portal, and in the process we'll create a resource group.

Chapter 10 How to Explore Azure Portal

Once you have an Azure account, you have access to the subscriptions associated with that account by going to portal.azure.com. I'll select the Microsoft account I used to create this Azure account. And I supplied my password earlier, so I'm already logged into my Microsoft account in this browser. That brings us into the Azure portal, and by default we're brought to the home page, which has some shortcuts, including shortcuts to create resources in Azure. Let's look at the menu across the top. At the top right is information about your account. There's a link to sign out, a link to go to the details of this Microsoft account, and the ability to switch directories. We'll talk about Azure Active Directory tenants later on, which is what this refers to. You can also access a link here to be taken to view your Azure bill if you have one.

Next across the top is the ability to send feedback to Microsoft. Then there's a link for Support + troubleshooting. Azure provides unlimited free support for subscription management, and for technical questions there are several support plans available which do have costs involved, but during a trial, you get the Developer support plan for free, which is normally a paid plan. Next is the Portal settings.

The first tab here has to do with directory management, which again involves Azure Active Directory, so I won't talk about that just yet. But this is the one I want to show you, Appearance + startup views.

I personally like to see the left menu permanently docked and also prefer the first screen I see to be the dashboard. Let's apply these changes. Now that menu appears on the left.

We'll talk about this in a second, but first let's finish off with the menu at the top. The Notifications link shows anything that's in progress or has recently happened, like when you create a resource in Azure it will show here that it's in process and when it's finished being created. The next link is to change directories. So, this is just a shortcut to the first tab on the Portal settings. This farthest link on the left is for the Azure Cloud Shell.

This is basically a command line interface right within the Azure portal that lets you run PowerShell commands and commands for the Azure CLI, or command line interface. Those are powerful ways to manage Azure, and we'll take a look at them later on too, as well as using this Azure Cloud Shell. Now let's look at the menu on the left.

The Home link shows that screen we saw when we first logged into Azure, so that was the home screen. The dashboard is the screen that will get shown from now on when I log in because of the portal settings I changed. The dashboard is a focused view of your resources in Azure. You can visualize data from multiple resources here and pin charts and views to get a complete picture of the health and performance of applications you create in Azure. That'll make more sense as we go along. You can add and manage tiles, and they can be configured individually or you can modify the whole page by adding and removing tiles. There's some suggested ones here, but you can pin pretty much anything in Azure to your dashboard. You can create multiple dashboards, so you can have one for viewing the state of certain applications or one for viewing the state of all the virtual machines, basically whatever you want to see at a glance. Next, let's create a resource group in the Azure portal.

Chapter 11 How to Create Resource Groups in Azure

Since we talked about resource groups earlier, let's look at how to create one in the portal. You can do that from the existing list of resource groups. Along the left here are shortcuts to the various resource types in Azure.

You can modify this to list the types of resources you normally manage, but this is the default. So I don't have any resource groups yet, of course, but from the Create button we can create one. But before we do that, let's pretend we don't have a shortcut on the left menu for this type of resource. In that case, we can go to All services on the menu. From there, we could browse the categories or search for the name of the resource type we want to create or manage. If I click on the resource type, we get brought to the same screen as before. And this little pin beside the name of the resource type, this is how you would pin this view or list to the dashboard. The third way to create a resource is from the link, Create a resource.

That opens the Azure Marketplace, which Microsoft calls its online store of IT software applications and services built by companies. It looks similar to the All services screen you saw with categories of different types of services along the left, and you can use this to create basic Azure services, like a resource group. But the Azure Marketplace also contains you can install and sometimes purchase from other companies, like SendGrid is listed here, which is an online email service you can set up and use in your solutions. That's not a Microsoft service, so it's available here in the Marketplace, but not on the All services menu. Let's look at some of the categories here. Compute is where Azure services like virtual machines and function apps can be created, but also where preconfigured VMs can be chosen, like certain Linux distributions or a Windows Server VM with Visual Studio already installed. Storage has Azure storage accounts, but also offerings from other vendors that work in Azure on the right side. Same with Web. You can install plain Azure services or install a VM with WordPress already configured and ready to be used. But let's just search for resource group from here, and it shows up in the results, so let's click this. Now, instead of being brought to the list of resource groups in our subscription, we're brought to the Marketplace view, which provides an overview of the service, information about plans that are available, and so on.

Of course, there's not much here because this is just a basic Azure service, so let's create this resource. We're brought to the create screen and our subscription is selected by default. I'm only logged into one subscription.

We need to choose a name for this resource group, and this only needs to be unique within our subscription, not across all of Azure. You can name it whatever you want, it's just a string, but often companies come up with a naming convention for sorting and for searching. I like to end my resource names with an abbreviation of the resource type, so in this case I'll add an underscore and rg for resource group. Now you have to choose the region. A resource group is just metadata about all of the resources in the group, so this isn't where those resources will get created. You'll choose that separately for each individual resource that you create. This is just where the resource group or metadata itself will get created. I'll select the Azure region closest to me. Creating a resource in Azure is kind of like a wizard. You move through the tabs using the buttons at the bottom. The next page is Tags. Tags are key value pairs of metadata that you can add to resources, so they can be searched and grouped together.

You might use this to mark that a resource belongs to a certain application or a business group or an environment. Tags are helpful when it comes to billing too. There's a service in Azure for cost management, and you can filter resources by tags to see how much all the resources with a particular tag are costing you. You can also download a detailed spreadsheet of costs, and tags are included there for filtering purposes too. I've just added an environment tag. Let's move to the next screen. This is just a summary, so I'll click Create. We can see the new resource group by going to the Resource groups shortcut in the menu. Now we have one item listed.

I'll click the name, and that opens up the details for this resource group. The menu items are organized into groups, and this first group is common to all resources in Azure.

The Overview tab is where we can see all the resources included in this resource group. Of course there aren't any yet. We can also delete the resource group from the top menu. The Activity log is where any activity shows. I've already created and deleted a resource group with the exact same name, so it seems to be showing those activities on this resource group. The Access control tab is where you specify who can access the resource and what permissions they have. Of course there's only my account, and I'm the owner so I have full permissions, but you might want to grant someone access so they can't modify resources here. Tags is where we can manage the tags for the resource. The next group of menu items are different depending on the type of resource you create, but you'll often see entries for monitoring metrics and logs for the particular resource. Now let's close out of these and go to All resources. We actually don't have anything here.

Resource groups don't show up in all resources; just resources contained in resource groups are listed here. that's a tour of the Azure portal and resource groups. Next, let's talk about Azure Active Directory.

Chapter 12 Azure Active Directory Basics

You probably don't plan on being the only person managing or using the resources in Azure. So now, let's talk about the directory in your subscription that stores user identities. That's called Azure Active Directory.

I've got a shortcut on the left menu, but let's go to All services, and Azure AD is right here at the top. This is called an Azure Active Directory tenant. When a subscription gets created, it gets its own tenant, so it's a place where the users for your organization only are managed. The Azure AD tenant is the container for users and groups that you want to give access to resources in Azure. That could be to administer things like deploying web apps or creating containers in Azure Blob storage, or it could be user identities for end users for accessing a web application or uploading data to that file storage. On the Users blade, and by the way, when you click a menu option, the panel that opens with the details is called a blade, so this is where all the users in this directory are listed. You can manually add users here, sync enabled. This means that you can actually synchronize your Active Directory with this Azure Active Directory tenant and then assign those users permissions in Azure. You do that by downloading and installing a tool called Azure AD sync. this identity is a Microsoft account. It has a really long identifier because it's an external entity. I'll show you shortly how you can use your own custom domain name for your company here, but let's create a new user first.

When you create a user, the first option you have is whether you want to create the user identity in this Azure AD tenant or you want to invite an external user. External users are part of something called Azure B2B collaboration, or business to business. That's for users outside your organization who aren't part of an Azure Active Directory tenant, and it lets you give them access to applications and services. They become an object in your Azure Active Directory that you can assign permissions to, but their process is handled by their own external identity provider. Let's create a generic account called administrator. I'll just give it a name, and we can create the password or let Azure do it. Just make sure you copy this somewhere. You won't see it again, but you can reset it. It says in the Notifications area that the user was created successfully. Let's drill into the details for this user. From here, we can add this user to groups. Groups let you create, well, groups of users, so you can then assign roles to either individual users or to groups.

Those roles are used to provide access to resources in the subscription. You don't have to use groups, but it makes management a lot easier. There are lots of roles in Azure, and they can get pretty granular in terms of what they allow. Let's just give this user the Global administrator role. That will allow them to perform pretty much any action in Azure. So I'll add this role and then refresh the view. It might take a few seconds for the portal to pick up the change. This user has the Global admin role now. Farther down the menu, you can assign licenses.

By default, your Azure AD tenant has the Azure Active Directory Free license, but there are other licenses that you can assign to individual users, like Azure AD Premium P1 or P2 licenses. These give users additional features, like being eligible for Conditional Access policies. I won't go into too much depth on Conditional Access policies, but basically they allow Azure to make authorization decisions based on things like the Azure AD groups that the user belongs to or the location the user is coming from and characteristics about the device they're using. Conditional Access policies can also work with Azure AD Identity Protection, which uses machine learning to identify risky behavior. If the user is approved to sign in after the Conditional Access policies are evaluated, you can also choose to enforce authentication. That's where a user needs to provide a second factor of authentication after their username and password, and that's enabled by a service called Azure Authentication, which can text a passcode to the user's mobile device that they then enter into the browser to complete their login to applications in Azure. There's also an app the user can install on their device called the Microsoft Authenticator app. Azure can send push notifications to the app for the user to approve a sign in that was initiated in the browser. The app can also generate a rolling token code every 30 seconds that can be used when logging into Azure. That's just a quick introduction to Azure MFA. There's more to it than that, like being able to use the Microsoft Authenticator app for passwordless authentication. Let's go back to the Azure Active Directory tenant that we were looking at. You can use Azure MFA with Conditional Access policies if your users have at least an Azure AD Premium P1 license.

You can also enable it for individual users so they always have to use authentication, and you do that from this screen. Let's go back up the hierarchy here to the root of this Azure Active Directory tenant.

Another thing you can create in Azure AD is app registrations, which represent an application, like a mobile app or a web app or a web API. It creates a trust relationship between your app and Azure AD, then those applications can use Azure AD to log in their users. Remember I mentioned that you can sync your Active Directory with this Azure AD tenant.

That's done using Azure AD Connect, and you can download the tool to install it from this blade. Then you have options on how to set up that connection, either directly or using federation with a tool like AD FS. I also mentioned earlier that you aren't restricted to the domain name that Azure creates for you.

You can add your company's own domain name to this Azure AD tenant. You need to own that domain name, though, just like you would have to own it when using it for a website. You can purchase a domain name from a domain name registrar. But let's create a new user and let's look at the domain list in the tenant. It shows both domain names, the default one that Azure created and the one I added, so I can create a user principal name. That's all for Azure Active Directory for now. You might have some questions still about subscriptions and directories and how they relate, so let's talk about that relationship in a little more detail next.

Let's tie together some of the concepts you've learned about so far. Earlier when we created an Azure free trial, we used a Microsoft email account, and that created an Azure account. An Azure account is referred to as a billing account in the documentation. It's an entity that can have subscriptions. In other words, I can create multiple subscriptions and access all of them when I log in with my account. The documentation says there are several types of billing accounts. The Microsoft Online Services Program contains the Azure free account that we created. It also contains the accounts that charge our credit card for the resources consumed and the Visual Studio subscriber accounts that are basically accounts with some free credits. There are also enterprise agreements for organizations, Microsoft Customer Agreements also for organizations, and Microsoft Partner Agreements for cloud solution providers. Let's look at how these relate to subscriptions. For the type of account we created, we can have multiple subscriptions, so I could continue with my free trial subscription and also create a subscription, or I could upgrade the free trial subscription to a pay as you go.

Each of those subscriptions gets an invoice showing the resources that were consumed each month, and each subscription has a payment method, a credit card. Within a subscription, resources are created, and they're actually created within resource groups. We won't look at the other billing account types. They just add some additional layers for management and accounting. When we signed up for the Azure free trial, a subscription was created, and there was an Azure Active directory tenant created too, so you might assume that this is a relationship, but it doesn't have to be. Multiple subscriptions can have a trust relationship with the same Azure Active directory tenant, but each subscription can only be linked to one Azure Active Directory tenant, and you can change the tenant that the subscription trusts. To recap, an account can have multiple subscriptions. Subscriptions contain multiple resource groups, and resources groups contain resources. A resource can only belong to a single resource group, and a subscription has a trust relationship with an Azure Active Directory tenant. The subscription trusts Azure Active Directory to authenticate users, services, and devices. You can have multiple subscriptions trust the same Azure AD tenant, but each subscription can only trust a single directory, and all of the users have a single home directory for authentication, but each user can also be a guest in other directories. Okay, so you might be wondering why would you want to have multiple subscriptions? You might want individual subscriptions for different environments like dev and production and be able to apply separate access to manage each subscription using access control, or you might want to keep resources separate in different subscriptions to make billing easier because each subscription can provide a bill for the resources that it uses, so maybe different business groups in your organization want that. Having multiple subscriptions might seem like it could be a management nightmare, but there's something called management groups that make this easier. Management groups can contain multiple subscriptions. They can also contain other management groups so you can create a hierarchy. Maybe you create management groups for different departments so they each have their own subscriptions or for different however your organization is structured. Then you can manage security for your subscriptions at the management group level. Permissions given at the management group level will get inherited by the management groups and subscriptions underneath. All of the subscriptions under the management group need to trust the same Azure Active Directory tenant, though. There's also something in Azure called Policies, and these allow you to set rules, like virtual machines can only get created in the East US region. Then you apply that policy to a subscription or at the management group level and it gets enforced and reported on. We won't get into Azure Policies in this book, but just know that you can use ones, like requiring all resources to use certain tags or more complex policies like ones that enforce compliance to FedRAMP and HIPAA standards. You can browse all the policies available in Azure on docs.microsoft.com, and you can also create your own custom policies and apply them to your management groups and subscriptions. In summary, so far you have learned how instances of services that you create are organized into subscriptions and resource groups, and you learned how the underlying physical Azure platform is implemented through regions and datacenters. You also saw how to create an Azure subscription, navigate the portal, and create resources, and you learned about how Azure Active Directory is used to manage identity and access for managing Azure, as well as being used as the foundation for access control by users of your applications. Next, we're going to look at some of the services in Azure used for compute, like virtual machines and app services.

Let's talk about cloud computing service models. This might be a review for you, but it helps to put Azure compute options into perspective. These are categories that describe how much a resource is managed for you in the cloud. Another way to look at it is how much responsibility do you have with regards to managing the resource. Let's look at each of these. The model isn't a cloud model at all, it's just here to explain all the things that you're responsible for when you host your own infrastructure This is what we talked about earlier, but besides all the physical infrastructure, if you're hosting your applications on virtual servers, then there's virtualization software that needs to be configured and managed. Then you're responsible to configure the virtual machines, which includes OS licensing and patching. You're responsible for installing the middleware on the servers and any runtimes, like the .NET Framework or Java, and then managing the data and applications that you host on those virtual servers. When you move to the cloud with the Infrastructure as a Service cloud service model, you can choose to provision resources like virtual servers. In Azure, you can create Windows or Linux VMs and everything below the OS layer of the VM is managed for you, so you don't have to worry about hardware refreshes or disks failing, you just choose the type of virtual machine you want and you pay for it. The cost includes the operating system license, but you can also leverage existing Windows Server licenses you might already own You can install whatever you want on the VMs that you provision as Azure Virtual Machines. If you want to install a SQL Server database, for example, you can do that yourself. We'll look at some other options for SQL Server and Azure later on too. The next service model is Platform as a Service, or PaaS. Platform as a Service is a complete development and deployment environment in the cloud. Azure App Service is a compute service in Azure that allows you to host web applications and APIs in a preconfigured environment where all the runtimes are installed, like .NET Core, Java or PHP. And those frameworks get updated along with the underlying virtual machines that they run on. You only need to manage the applications and services that you develop. Microsoft manages everything else. The third cloud service model is Software as a Service. This is actually the most popular use of cloud computing in terms of the number of users. These are fully functional apps that users can connect to over the internet. Office 365 is the Microsoft solution for email, calendars, and office tools. Instead of buying software to install on the desktop and managing your own email servers, you can purchase Office 365 on a basis. You can still download and install Office tools like Word and Outlook, but they're also available in the browser. The central hosting of the Exchange Server is handled in the cloud and hosted by Microsoft. SharePoint is another Microsoft offering that falls under Software as a Service. As you can see with the Software as a Service model, you just use the software and you get a fully managed service that's available across devices and platforms. The services you'll be learning about throughout this book fall into the Infrastructure as a Service and Platform as a Service models. Now that you understand a bit about responsibilities with these service models, let's talk more about compute options in Azure next. Then we'll look at the major compute models individually, starting with virtual machines. Next, we'll talk about containers, which allow you to package applications and dependencies for deployment. After that, we'll look at Azure App Service. And finally, explore Azure Functions, which allow you to run small pieces of code without requiring a full-blown application.

Chapter 15 Azure Compute Options

Azure compute is really an overarching category for a bunch of services in Azure that provide computing power for running applications. The main services in Azure compute are virtual machines, container instances, and there are a number of ways to host containers in Azure, Azure App Service and Azure Functions. Let's discuss each of these at a high level and then we'll get into more detail. Virtual machines are software emulations of physical computers. They run on physical computers in Azure, but multiple virtual machines, or VMs, can run on the same physical host and use the resources of that host. You connect to a virtual machine in Azure using a remote desktop client, and you can manage all aspects of the operating system, including installing whatever software on the VM that you want. It gives you the most control, but also requires the most management because you're responsible for all the configuration and security patches and updates required by the operating system on that virtual machine. But you'll see that Azure offers some services to make that easier. Let's talk about some of the benefits of choosing virtual machines in Azure. Virtual machines are probably the most familiar option for most IT pros because they're just like virtual servers that you would maintain If you're planning on migrating to Azure from virtual machines can provide a approach by creating VMs in Azure similar to the physical or virtual servers you have You might have applications that require operating system resources like registry access or that use authentication mechanisms like Windows Integrated Authentication and you don't want to rewrite those apps to run on cloud services like Azure App Service, or you might have older applications or custom software that needs to be installed, VMs are the way to go. Besides having the ability to install whatever software you want on the virtual machine, you can deploy your own applications and you can host multiple applications on the VM. So, virtual machines can have some cost savings versus deploying those apps to single instances of other services, but the applications share the resources on the VM, so you need to be aware of one application using too much CPU or memory and affecting the others. Or if multiple applications use a shared library, for example, updating that library to benefit one app could cause another app to break. That's an issue that containers are meant to solve. Containers are used to wrap up an application into its own isolated package. It's for applications and services, so web apps are a typical example. When an app is deployed using a container, everything the application needs to run successfully is included in the container, like runtimes and library dependencies. This makes it easier to move the container around. Containers reduce problems with deploying applications. Containers are kind of like virtual machines, but they run on top of virtual or physical servers using a container runtime layer, similar to how virtual machines run on a virtualization layer. You can host multiple containers on a single virtual machine if you install a container runtime, and Docker is an example of one of those. In terms of the service models we discussed earlier, there are services in Azure that host containers on Platform as a Service offerings, like Azure Container Instances and Container

Apps. The next Azure compute services fall under the Platform as a Service model. Azure App Service lets you quickly build and deploy web apps, mobile apps, and API apps that can be leveraged by other applications or accessed by client apps over HTTP using REST. They also allow you to run apps and scripts, similar to how you would install a Windows service on your web servers to perform some task on a timer. You can choose your application runtime, like .NET, Node.js, and several others, and you can choose whether you want the underlying VMs to be Linux or Azure App Services takes care of managing those underlying VMs for you, which is the most obvious benefit, but you also get extra features like integration with authentication providers like Azure Active Directory to handle authenticating users to your applications. You get something called deployment slots, so you can have multiple versions of your app for development and production and quickly swap those deployment slots to promote the apps. App Services also has features to scale out the underlying VMs. You can add and remove virtual machines manually or Azure can autoscale the VMs based on metrics that you configure, like the amount of CPU being used. Azure App Services are great for running web apps and APIs that are used by mobile apps and client apps and other services, but sometimes you just need a piece of code to run in order to do some task like process a file or update a database table or send a message to another service, and that's what Azure Functions are for. It's a service to host small pieces of code, but you can chain functions together or use them as part of other solutions. Functions can run on a timer or in response to events like an HTTP call, and there are triggers that you can use, like if a file changes in Azure Storage that can trigger an Azure Function to run. You only pay for the compute power that you use. For small tasks, Azure Functions are a great way to save effort and money. Azure Functions are often called serverless computing, although that's kind of a loose term. There are always servers involved, it's just how much you need to interact with them. There's another service in Azure that's often categorized as serverless compute also. It's called Azure Logic Apps. These allow you to configure workflows right in the browser and connect to various services inside and outside of Azure using connectors. Logic Apps sometimes get discussed in the context of Azure compute, so we'll look at them later when we discuss Azure Functions. But next, let's look more closely at virtual machines in Azure.

Using Azure Virtual Machines, you can set up servers in the cloud. You can basically recreate your environment in Azure if you choose. You could have an Active Directory server storing user accounts, a DNS server, web servers, file servers, and database servers. Using a virtual network in Azure, these VMs can all communicate and security can be enabled to restrict ports, all the same things as except they're in the cloud, so you can also enable access to the internet if you choose. There are additional features of Azure networking like load balancers and firewalls that allow you to secure your VM network. You can also extend your environment into the cloud by connecting your network to a VNet in Azure, then the VMs in Azure can essentially become part of your network. So if you're running out of capacity and don't want to buy new hardware, this is an option. There are several ways to deploy VMs to Azure. You can upload your own VM images into a storage account in Azure and use them as image templates to create instances of virtual machines. There are some steps involved to prepare your disk files, but there's a much easier way to create VMs in Azure by choosing from preconfigured VM images from the Azure Marketplace. Many of them are provided by Microsoft, but also by other third parties. When you create a VM from the Marketplace, the licensing costs for the operating system are included in the price. When you create a virtual machine in Azure, there are a lot of configuration options, but the three big decisions are the image you want to use, the size of the VM, and the availability options. You start by choosing a VM image.

This is the configuration that decides what operating system is installed, and there are several available, like different versions of Windows Server and Windows desktop operating systems or different distributions of Linux. You can also choose images that are preconfigured with software. So you can create a VM image that has WordPress Server already installed or an SMTP server or development tools like Visual Studio. When you create a VM, you also choose the VM size.

Azure has predefined configurations that decide how many virtual CPUs are included and the amount of RAM. Different VM sizes are suitable for different workloads. By browsing the VM pricing page, you can see a description of each size of VM, and they're grouped into categories, series, and then the actual VM instance code that you choose when you create a VM. Under General purpose VMs are the VMs.

It says here these are suitable for development workloads and low traffic web applications and small databases. Farther down, the VMs are for most production workloads. Within each series, you can choose an instance size with a specific configuration that you need.

Each size has a price and the price is listed per month, but you're actually charged at an hourly rate and you can deallocate a VM when it's not in use to save on compute charges. There are also options with some VM classes to reserve the VM for one or three years. If you expect to be using it for that long, you can get some significant cost savings with that commitment. There's also Spot VMs that allow you to use VMs that come from unused capacity in Azure. You can get them significantly cheaper, but you have to be able to tolerate Azure evicting the VM with only 30 seconds notice so Azure can recover the capacity. If you're just running a DevTest environment or a batch processing job, that might be fine. As you go farther down, the VM series gets more and more specialized.

There are compute optimized servers, which have a high ratio.

series VMs have higher core ratio, so they're better for hosting database servers. You'll notice that some of the series aren't available in the region that's selected above. VMs get more specialized as we go farther down.

It says these HBv3 VMs are optimized for computing, like financial calculations and weather simulation. That's pretty specific and pretty expensive. There's a new tool in Azure that can make finding the right VM size much easier than reading through all these descriptions.

The virtual machine selector lets you find VM sizes by workload type, OS and software or by deployment region. I won't go through all the screens, but you can select things like the type of operating system and the minimum and maximum number of CPUs and RAM, and it'll produce a list for you. You select a VM size when you create the VM, but it is possible to resize the VM later. That's called scaling up if you're choosing a VM size with more CPU or RAM, or scaling down if you change to a smaller VM size. That's part of the elasticity in the cloud that we talked about previously. You don't have to create a new VM if you need extra processing power, and you can release those resources when you're done with them. Let's talk about the related resources that a VM needs. A virtual machine needs a disk to store the operating system. That's created when you create the VM, and it gets managed by Azure in Azure Storage. It's basically your copy of the VM image. You can also add data disks if you need to store a lot of data as part of your VM. Maybe you need database storage or some other file storage attached to your VM. A VM also needs to exist on a virtual network in Azure, generally referred to as a VNet. That's how it can communicate with other VMs and out to the internet. So even if you only have a single VM, it needs an Azure Virtual Network. You can either create one while creating the VM or you can attach a VM to an existing virtual network. The VM needs a network interface in order to communicate on the network, and you can have a public IP address for the VM so it can be remotely accessed. That could allow you to use the VM as a web server. You can also set up security rules to filter network traffic between resources on the virtual network using network security groups. We'll talk more about networking in the next module. So each of these is considered a resource in Azure with their own configuration screens, and when you add managed data disks, those have associated costs. Accessing those managed disks also have costs as storage transactions, and any data that comes out of Azure is also charged. That's actually true of Azure in general. It's free to put data into Azure, but there are egress charges when data comes out, like if you're using your VM as a web server, the data in the responses incurs charges. But these are very, very small charges, just something to be aware of when you're pricing out a solution that involves virtual machines. You can estimate your costs in Azure using the Azure pricing calculator. When you add a VM, it will add the related resources that can incur charges. Some related resources, like virtual networks, are actually free. Next, let's talk about the availability options with virtual machines in Azure.

Chapter 17 Azure VM Scale & Availability Sets

By now, you understand that Azure Virtual Machines are hosted on physical machines in an Azure datacenter. Sometimes those physical machines need maintenance or something fails or they need to be restarted. That's just reality. So if you design your solution with a single VM, you're introducing a single point of failure into your application. For that reason, a datacenter is organized into update domains and fault domains, and you can take advantage of these to create a highly available solution using multiple virtual machines. Update domains are groups of virtual machines and the underlying physical hardware that can be rebooted at the same time. Fault domains define a group of virtual machines that share a common power source in the datacenter and a common network switch. When you're creating a virtual machine in Azure, you can choose to create it in an availability set with other VMs. When you do that, Azure places your VMs in separate update domains and fault domains. So you're essentially telling Azure that these VMs are part of an application so Azure can help with the resiliency and availability. This doesn't protect you from things like operating system or failures, but it does limit the impact of potential hardware failures, network outages, and power interruptions. And you actually need to create at least two VMs within an availability set if you want the 99.95% uptime guarantee in the Azure agreement. To use these VMs for redundancy in a solution, you'd need to put them behind a load balancer. Users access a web server from a single

IP address and URL, but the load balancer routes the traffic to one of the VMs in the solution based on availability and load. To make that easier, though, Azure offers something called virtual machine scale sets. These let you create and manage a group of identical virtual machines, and Azure will put them behind a load balancer for you. You can configure virtual machine scale sets to scale with demand so Azure can add and remove VMs from the scale set as needed, and of course you configure the parameters around that. Those VMs are spread across fault domains, so you have that protection as well. Virtual machine scale sets let you maintain a consistent configuration across your VMs. You get resiliency if one of the VMs has a problem, and the autoscaling feature helps with application performance. If you plan to set up a solution that requires a lot of VMs working together, up to 1000 VMs are supported in a virtual machine scale set.

We could start from the shortcut on the left menu, but let's go to All services and search for virtual machines. I'll click that in the search results, and let's create a VM from the Create button at the top. There's a few options here.

Azure virtual machine with preset configuration just narrows down the VM sizes based on whether you intend this VM for development or production. Azure Arc lets you manage VMs in environments outside Azure, including your environment. And Azure VMware Solution virtual machine lets you move workloads from your datacenter to Azure. Let's just create a VM in Azure. That brings us to the create screen with all the tabs across the top. We'll go in order here.

I won't go through every option on these screens, just some key ones. We need to put this VM in a resource group, and all the related resources will get created there too. We could use the existing resource group, but let's just create a new one. That'll make it easier to delete these resources later. I'll give this new resource group a name. Now I have to give the VM a name. This can be pretty much anything, but you'll probably develop a naming convention for your organization. Next we choose a region. There are only a few regions available because this is a free Azure trial subscription. In a subscription, the list is much bigger. Next we choose the availability option. We can choose to use availability zones. This relates back to the discussion on Azure regions.

Most regions have availability zones where there are separate datacenters in the region. If you're planning on creating multiple VMs for your solution, you can choose which zone to put the VM in. There are also virtual machine scale sets available and availability sets. Let's choose Availability set. And since there isn't an existing one to add this VM to, I'll create a new one. We can configure the number of fault domains and update domains. Let's just leave the defaults and click OK. Next we choose the image we want to use to create this VM. We can choose from various flavors of Linux or versions of Windows.

I'll just choose a Windows Server image. Next we choose the size. There are a few popular sizes listed here, and we can browse all the sizes and their specs if we want to.

But let's choose the smallest VM listed here. Next we need an administrator name. This will be a local account on this VM. We can add other accounts for access later. After that, we can open some inbound ports. You'll want at least RDP enabled on a Windows VM so we can remote into it. So I'll leave port 3389, the default RDP port.

Next let's go to Disks.

We can choose the type of disk to use for the operating system, and we can add data disks here, but we can also do that after the VM is created. So let's move on to Networking.

This is where we can add this VM to an existing virtual network or create a new one. Since we don't have an existing VNet, we'll create one. And you can break that VNet into subnets, and then you can place VMs in different subnets and control communication between them. Let's leave the defaults. And the public IP address will get created too. The security on this VNet will be configured to allow access from port 3389 the internet so we can remote in.

We could configure load balancing here if we intend to have multiple VMs as part of this solution.

Let's go to Management. Here we can configure options like boot diagnostics, which lets you debug problems booting up your virtual machine.

You can also create a identity, which is an Azure service account for this VM. So you can do things like grant this VM's service account access to a database or to a storage account in Azure. You can enable auto shutdown. Shutting down a VM in Azure will save you money on compute charges, but you still pay for the storage of the underlying VM. So if this is a VM for development, let's say, you might only need it running during business hours. You can also shut down a VM yourself anytime you want. Let's go to Advanced, and this is where we can configure applications to install after deployment and run custom scripts while the VM is being provisioned.

And we talked about tags earlier, which help with managing large numbers of VMs. Let's review this configuration and create this VM.

Next, we'll explore the management options once the VM is created.

It took about 2 minutes for everything to get created.

We could go to the VM configuration screen from here, but let's go to All resource groups and let's see the resource group that was created.

You can see all the related resources that were created with the VM itself, like the virtual network, the operating system disk, the network interface, public IP address, and even the availability set is created as a separate resource with its own configuration. Let's drill into the VM. On the Overview tab, you can see the public IP address for this VM. So this is how you would access it from the internet. There's also the private IP address, which is the IP address of the VM on the VNet that was created. And on the Overview tab, you can also stop the VM, and this is how you can save on compute charges when not using the virtual machine.

This releases the server resources associated with the VM, like the underlying CPU and RAM. The Networking tab allows us to manage the ports that are open. We could open up port 80 for HTTP traffic here, for example.

Disks are where you can attach data disks if you need more storage later.

On the Size tab, you can resize the VM. So we could scale it up if we found that the workloads that we're putting on here are more than the VM can handle.

You just choose the new size and click Resize. If the VM is running, this will cause it to restart. On the Configuration tab, you can manage settings like the licensing for the Windows operating system. You might have existing Windows Server licenses you'd like to use in order to save on Azure costs.

Identity is where you enable the managed identity we discussed during the VM creation screens, and that allows you to provide access to other Azure resources using the Azure Active Directory identity of this VM. Backup is where you can configure options for backing up this VM. Azure has a service called Azure Backup where you can store backups of VM disks, file shares, and blobs in Azure storage. And even databases running on Azure VMs can be backed up. Of course, there are additional costs associated with storing backups. Azure Site Recovery provides disaster recovery by replicating a VM to a different Azure region.

You can use this service to fail over your Azure VMs to another region and also to replicate VMs from other environments for failover to Azure, like VMs and even Windows virtual machines in Amazon Web Services. The Updates tab lets you use other services in Azure to help you provide updates to this VM.

With Infrastructure as a Service VMs, you're responsible for updating them. Azure does help with this, though, by allowing you to leverage a service called Azure Automation to push out updates to VMs that are enrolled with the service. You need to configure that and schedule the updates, though. This works for Windows and Linux VMs. Now let's see how to connect to this VM remotely.

On the connect page, you can connect using Remote Desktop Protocol, SSH or using a service called Azure Bastion, which lets you connect using your browser and the Azure portal. RDP is the traditional way to log into a Windows VM, so let's choose this. Azure is going to create a file for us to download. I'll go to my Downloads folder and let's edit this file. If you've used Remote Desktop Connection before, you see this is just a standard file with the IP address of the VM and the RDP port 3389 already configured. The only account that has access to this VM right now is the local administrator account we created on the VM, so I'll use a different account, and the name is the VM name, backslash, the account name, and the password I used during the VM creation. Once it connects, we're brought into the remote session. Server Manager opens on the VM, and here we can manage the VM and even add it to a domain.

I've used Azure VMs many times to create test environments in the cloud by creating a VM and installing Active Directory and then joining VMs to the network, just like you would That's a standalone environment. If you want to give Azure AD users access to this VM and the applications on it, there's a way to do that using Azure AD Domain Services.

First of all, Azure Active Directory is not the same thing as Active Directory They both store user identities and allow you to create groups of users for security purposes, but Azure AD was built for the cloud and it uses web authentication protocols for authenticating users like OAuth 2.0 and OpenID Connect. Cloud native services like Azure App Service use these protocols by default. The authentication technologies used most often in Active Directory is Windows Integrated Authentication, which uses Kerberos and NTLM protocols. With Windows Integrated Authentication, a user logs into the local network and a security token gets passed around so they can get authenticated by any computer that's joined to the network, including web servers. That functionality is provided by domain services, which is just a part of Active Directory. Many legacy applications use Windows Integrated Authentication, so if you're moving them to VMs in Azure, that can be a showstopper. You may not want to modify those applications to use another authentication method, or in the case of commercial software, you probably can't change the authentication methods that they use. One possible solution if you have an Active Directory infrastructure is to extend your network to the cloud and use your Active Directory Domain Services, at least for users in your Active Directory. You would join the VMs in Azure to the same network as so really there's no connection to Azure AD for those servers. For users accessing applications on the VMs, though, they could still authenticate with Windows

Integrated Authentication. But user identities stored only in the cloud have no connection, and you may be looking to minimize your dependency to AD or you don't plan on having a hybrid cloud at all. So there is a solution in Azure AD, and it's called Azure Active Directory Domain Services. It's not enabled by default, you need to turn it on, and it provides the ability to join virtual machines to managed domain where the user identities are stored in Azure AD, but the VMs can use legacy authentication methods like Kerberos and NTLM. When you set up Azure Active Directory Domain Services, Azure deploys two domain controllers into your selected Azure region, and you don't need to manage or update them. It's handled for you. Information from Azure AD is synchronized into Azure AD Domain Services. Then, applications, services, and virtual machines connected to the managed domain can use common AD features like Domain Join, Group Policy, LDAP, Kerberos, and NTLM authentication. If you have Active Directory, you learned that you can synchronize users and groups from your Active Directory to Azure Active Directory. That's not the same thing as joining the two networks. They're still separate, there's just an agent installed in your environment that synchronizes the user accounts into Azure Active Directory on a continuous basis. Then those users can be granted access to applications in Azure. Even though they're accessing the application from it's their Azure AD identities that are used. They can still access applications hosted on those virtual machines that use legacy authentication protocols. Before we leave the discussion of virtual machines, let's talk about Azure Virtual Desktop, next.

Chapter 21 Azure Virtual Desktop Basics

Azure Virtual Desktop is a desktop and app virtualization service in Azure. It was previously called Windows Virtual Desktop. Azure Virtual Desktop is used to provide Windows desktops to users with computers actually running in Azure. The user logs into a computer in Azure where all their applications are installed and all their data is accessible, but none of it is stored on their local computer and all the processing by the applications is being done in the cloud. Users can access their remote desktop and applications from any device. There are native apps provided for Windows, Mac, iOS, Android, and an HTML5 interface is also provided, so the remote desktop can be accessed using a web browser too. What's the purpose of Azure Virtual Desktop? What problems does it solve? It separates operating systems, data, and apps from local hardware. That enables central management and security of user desktops with less IT management required. You don't need support because all the apps are running remotely in the cloud. Azure Virtual Desktop provides a separate compute environment for users outside of their local device, so the chance of confidential information being left on the user device is greatly reduced. It also lets you provide standard images to users with all the tools they need already configured without having to procure hardware, set it up, and ship individual computers to users. They just access the virtual computers from their existing devices. You can choose images for Windows 11, 10, and Windows 7 with extended support until 2023, and also Server operating systems like Windows Server 2022, 2019, 2016, and 2012 R2. You can also create or upload your own image with all the software and configuration needed for users. At sign in, the user profile is dynamically attached to the computing environment and appears just like a native user profile on a local machine. Users have access to their own data, and users with privileges can even add and remove programs without impacting other users on the same remote desktops. Azure Virtual Desktop is similar to Remote Desktop Services and Windows Server, but if you've ever set up that environment in an enterprise, you know there are multiple roles and multiple servers required for scalability. You can avoid all that configuration by using Azure Virtual Desktop. So, it's really a Platform as a Service offering to provide remote virtual machines. In the past, if you wanted to provide client operating system VMs to users in Remote Desktop Services, you had to have a single VM for each user. To have multiple users use the same VM and conserve resources, you needed to use a server operating system, but Azure Virtual Desktop supports Windows 10 or 11 which means you don't have to overprovision VMs. You can let users share the resources of a single VM. Users on a environment still have a unique secure experience, and they can use all their apps, like Office 365. The user's data and files are persisted on a separate disk that gets attached when the user logs in, so they get their desktop settings and application settings as if it's their own computer. But the user profile is separated from the operating system, so you can update the operating system and not lose the user's profile. Azure Virtual Desktop works with a couple of features that you've already learned about. You can domain join Azure Virtual Desktop VMs to Azure Active

Directory Domain Services or to an existing domain in Active Directory if you've created a hybrid cloud. Azure AD provides a secure, consistent experience that allows users to roam from device to device. And it also lets you use Azure authentication for another layer of security. Now let's move on from virtual machines and talk about hosting containers in Azure

Containers are a way to wrap up an application into its own isolated package. It's for applications and services, so web apps are a typical example. When an app is deployed using a container, everything the application needs to run successfully is included in the container, like runtimes and library dependencies. This makes it easy to move the container around from your local workstation to VMs in your environment that have the container runtime installed or to a managed container hosting service in Azure, like Azure Container Instances or the Azure Kubernetes Service. The main characteristic of a container is that it makes the environment the same across different deployments, so containers reduce problems with deploying applications. Let's talk about how containers are different from virtual machines. Virtual machines run on some sort of infrastructure, whether it's your laptop or it's a physical server in a datacenter in Azure. There's a host operating system that might be Windows, Linux or macOS. Then we have a hypervisor layer, and this is what runs the virtual machine and provides resources to it from the host operating system. is Microsoft's hypervisor technology, but there are others like VMware and KVM. Then there's the virtual machine. The virtual machine contains a full copy of an operating system, and it virtualizes the underlying hardware, meaning the CPU, memory, and storage. It also contains the application that you want to run. If you want true isolation of your applications, you'll have a copy of a VM for each application that you deploy, and that VM will need to have all the runtimes and libraries installed that the application needs. If you want to run three applications in isolation, then you'd be running three virtual machines on this hardware, each with a guest operating system that might be 800 MB in size, and each VM would require a certain amount of CPU and memory allocated to it because, again, virtual machines virtualize the hardware. Containers, on the other hand, virtualize the operating system. The host could be a physical or a virtual server, and on top of the operating system there's a runtime. This is kind of like the hypervisor for virtual machines, but it's for containers. On top of the runtime are the containers, which just contain the application along with any dependencies for that application, like frameworks and libraries for connecting storage, for example. These are the same types of things you would normally install on a VM to run your application. The containers emulate the underlying operating system rather than emulating the underlying hardware. This makes containers smaller in size than a virtual machine and quicker to spin up because you're only waiting for the app to launch, not the operating system. Because containers are so lightweight, you can host more containers on the host VM or physical server than using traditional virtual machines for each application, so there's obvious cost savings associated with that. A container is an instance of a container image. An image is a template with instructions on how to create the container, and the container is the runnable instance of the image. You can create your own container images by leveraging existing images and adding the frameworks, any dependencies, and finally the code for your application. Then you can deploy the container in a repeatable way across environments. Container images get stored in a container registry. A container registry is a service that stores and distributes container images. Docker Hub is a public container registry on the web that serves as a general catalog of images. Azure offers a similar service called Azure Container Registry, which provides users with direct control of their images, integrated authentication with Azure AD, and many other features that come along with its Azure integration. I just mentioned Docker Hub. A Docker container is a standard that describes the format of containers and provides a runtime for Docker containers. Docker is an open source project that automates the deployment of containers that can run in the cloud or Docker is also a company that promotes and evolves the technology, and they work in collaboration with cloud vendors like Microsoft. Docker has a runtime process that you can install on any workstation or VM, and there are services in Azure that provide that runtime for you. Remember that containers are portable, so they can be moved around to different hosts. Now let's talk about the different ways you can host containers. You can set up a local environment by installing the Docker runtime. Then you can develop your app locally and package up all its dependencies into the container image that you want to deploy. You could also host a container on your own hardware or virtual servers by installing the Docker runtime there. You can deploy containers on your own VMs in Azure. If you just need a small dev environment or you're not ready yet to move into services, you can still package your application into containers and deploy those onto VMs that you control. Of course, you'll need to maintain and patch those VMs, but it can at least get you started with some of the benefits that containers offer in terms of deployment and agility. With each of these approaches, you need to install the container runtime, but Azure has several Platform as a Service offerings for hosting containers. Azure Container Instances, or ACI, is a service that provides a way to host containers without having to maintain or patch the environment. It hosts a single container instance per image, so it's intended for smaller applications like simple web apps or DevTest scenarios, but it still has obvious advantages to deploying containers to your own virtual machines because you get a managed environment where you only pay for the containers. Azure Kubernetes is a fully managed container management system that can scale your application to meet demands by adding and removing container instances, as well as monitoring the deployed containers and fixing any issues that might occur. Kubernetes is an open source project, and it's one tool in a class of tools called container orchestrators. Azure Red Hat OpenShift is a service in Azure that's a partnership between Red Hat and Microsoft, and it allows for running Kubernetes powered OpenShift. If your organization is already using OpenShift, this is a way to move to a managed hosting environment in the cloud. Azure Spring Apps is for hosting containers that run Java Spring apps, so it's tailored to that specific platform. You can actually deploy containers to Azure App Service also. So in addition to deploying code onto Azure App Service, you can package web apps as containers and host them in App Service. You can also deploy containers to Azure Functions for applications. And a relatively new service in Azure for hosting containers is Azure Container Apps. This is a managed serverless container platform for running microservices. This service is also powered by Kubernetes, but it doesn't provide direct access to the underlying Kubernetes configuration, which makes management a lot easier. So the choice of how to host your containers comes down to the development platform your team uses, what orchestration platform they might be accustomed to, and how much control you want over the management of the service. Let's take a look at the simplest of the container hosting options. Azure container instances

From All services, I'll search for container instances. There are a few other services here, like Container Registry for storing your custom containers, and Container Apps, which is another service for hosting the running of your containers. But we'll choose Azure Container Instances, a simple service for running single containers. And let's create one from the menu.

As always, we need to choose a resource group to store the metadata about the container. I'll choose the one we created earlier. Let's give this container a name, and this name only needs to be unique within the resource group. There's a lot of regions we can choose from to deploy this container instance, but I'll just leave the default. Next we can choose to deploy our container from a container registry. Azure has its own container registry service, or you could choose another service like Docker Hub, and that can be a public or a private registry with a login. But let's choose a quickstart image just to get up and running. I'll select the helloworld Linux image. And you can change the resource requirements for the container if you'd like, so the number of virtual CPUs and RAM that the container uses. Let's close this and move to the Networking tab.

Here you can create a public IP address and DNS name. So this will get prepended on the Azure service URL. You can configure environment variables here and the restart policy for the container.

This is where an orchestration service like Azure Kubernetes offers a lot more functionality. We won't add any tags. And let's create this container instance. It'll take almost a minute to create this in Azure.

And once it's created, we can navigate into the container instance. We have some monitoring happening on the Overview page.

On the Containers page, there's the container we created. It says the state is Running. Let's go back to the Overview page, and the fully qualified domain name is here on the right. This is the DNS name we added with the rest of the URL provided by Azure. Let's paste this into another browser tab.

We get a basic web page that was served by the container, so we know the container is running in Azure Container Instances. Let's go back into the container on the Containers tab. And there are logs available here, so you have some visibility into the output from the container, and you can even remote into the container and get a command prompt so you can run shell commands here.

I can list out all the files and folders that are on this quickstart container.

So it's serving the default page using Node.js. That's a quick look at containers and one of the Azure services that can host them for you. Next, let's talk about Azure app service

I mentioned that you can use App Service to host containers, but it's also the Platform as a Service offering for hosting code directly, meaning the App Service is more like traditional web hosting where the frameworks are already installed on the servers, like .NET, PHP or Java, and you can deploy your code onto those servers. The difference with traditional web hosting is that Azure handles the management and patching of the underlying servers for you, but you do have lots of configuration options. Azure App Service can host web applications, API apps, which are web services that use the REST protocol, and it can host the code for mobile applications, which are really just web services. You can deploy containers to Azure App Service too, but you don't have to. And there's also a feature of App Service called WebJobs that let you run services on the underlying VMs of the App Service. WebJobs can run continuously or on a schedule. They can run as executable files or they can run scripts like PowerShell or Bash scripts. So if you're running Windows services on your web servers now and wonder how you can do that in Azure, WebJobs offer that kind of functionality. There are other services in Azure to accomplish those types of tasks, and we'll look at some of them in the serverless computing section. App Service started out as a service called Azure Websites, and when you create a new App Service, the default URL is still suffixed with azurewebsites.net. And yes, you can use your own custom domain name with Azure App Service. This is just the default URL that first gets created. So an App Service is basically an individual website or API web service or mobile back end that you host. They're all really the same thing, just code that's hosted on a web server. Before you can create an App Service, though, you need an App Service plan. The App Service plan defines the size of the underlying infrastructure, which are actually just virtual machines in Azure. But remember, you don't patch or maintain those VMs and you have limited access to them. You can run more than one App Service on a single App Service plan. When you create an App Service plan, you choose the size of the VMs, meaning the CPU, RAM, and storage by selecting the plan type, also known as the pricing tier. Depending on the pricing tier, you also have access to different features of an App Service plan. Let's create an App Service plan next and explore the features of Azure App Service in the process.

We won't create an App Service plan first. We'll just do it during the creation of the App Service. I'll do that by using the shortcut on the left menu for all App Services in this subscription. Let's create a new one. And the first thing is the resource group to put this App Service in.

Let's create a new resource group for this. I'll just give it a name. And now let's give this App Service a name. This name needs to be unique across all of Azure because it's suffixed with azurewebsites.net. So this part here can't be the same as any other website on all of Azure. But remember, you can add your own domain name later. This is just the default one for creating the App Service. Next, we can choose whether to publish code or a Docker container or a static web app. For code, you choose a runtime stack, and this will be available on all the underlying web servers that the code is deployed to. The runtime you choose here will dictate which operating systems are available below. You can deploy your own Docker container from a container registry. And the Static Web App option is when you're just deploying code. There's no code running on the server, so there's no runtime framework option.

This actually brings you to another service in Azure for hosting these types of static web apps, and that service uses Azure Functions for logic. Static web apps integrate directly with GitHub or Azure Pipelines to pull your code, so it's a serverless environment where you don't need an underlying App Service plan. You can create one of these apps from the All services menu. It's just here is an easier way to get to that service, but it does seem a bit confusing. Let's choose a traditional code deployment, and I'll choose .NET 6, which is available on Windows and Linux VMs. Then we need to select a region where this will get deployed. If my subscription had an existing App Service plan in this region, I could choose to deploy the App Service onto that plan. But since it doesn't, a new App Service plan will get created. Because I'm using the free trial, I don't have the option to change from the free pricing tier. They're all using the free pricing tier right now because I've scaled them down to that plan to save money. Let's create a new App Service plan, and I'll give it a name.

And now we can change the size, which is really changing the pricing tier. It defaults to the S1 pricing tier under the Production grouping of tiers, but we can switch to the Dev group. Below the pricing tiers are the options that are available for each one. As I change pricing tiers, features are added. With the D1 pricing tier, we can use custom domains. At B1, we can add VM instances manually when we want to scale out the resources to handle increased load. And at the S1 tier, we get autoscale and staging slots and all the features we need.

Going up from there just increases the amount of CPU, RAM, and storage on the underlying servers. So let's choose this S1 tier, and let's move to the Deployment tab. Here we can set up continuous deployment so our code gets pulled from a GitHub repository automatically.

Let's move on to Networking. We could make it so this App Service is able to call into resources in a virtual network.

Let's leave the default, though. And on the Monitoring tab, we could create an instance of Application Insights, which would collect all sorts of metrics from the App Service, like user behavior and performance of the app. But let's turn this off for now.

We can enable it after the App Service gets created. We won't create any tags, so let's create this App Service. Once it's ready, let's actually go to the tab with all the App Service plans. This is the plan that was created with the App Service. In the App Service plan, there's a tab for apps. There's only one App Service here, the one we just created. Further down the menu, you can change the pricing tier if there's features you need to use or you just need more powerful VMs.

And on the Scale out tab, depending on which pricing tier you're on, you can add VMs to the plan. And, of course, there are costs associated with that.

And you can also configure autoscaling. So when a certain metric is reached, more VMs will be added. In the list here, there are lots of metrics you can have the App Service plan watch, like the amount of CPU being used, the disk queue length, and the percentage of RAM being used. Then you can configure some logic to add or remove VMs under certain conditions.

Let's close this, and let's go back to the Apps tab and drill into the App Service we created. So this is a web app that can use .NET for its logic. We haven't deployed any code here, but there is a default page created for you that you can access from this Browse button. That opens up a tab with the URL that we chose during creation, so we know that the servers are running and serving content.

Back in the App Service, let's explore some of the things we can configure. You can add a custom domain.

You can either purchase that through a third party and just verify it here in which case you'd need to point your DNS provider to this IP address or to the URL that you saw on the browse page depending on the type of DNS record that you use, or you can actually buy an App Service domain through Azure right here in the portal. Let's look at the Configuration tab. This lets you add name value pairs that can be read by your application code. You can also add connection strings to databases here. This lets you keep configuration and secrets out of your code.

If you've written ASP.NET applications before, you know there's a configuration file in your project. Any values here with the same name as what's in that file will override the values in the file, so administrators can manage configuration in the portal. Notice there's a checkbox for deployment slot settings. Deployment slots let you create different environments like dev, user acceptance testing, and production.

So you can have a different version of your web app in each of them, and you can promote your web app through the environments from right here in the portal, and the application settings and database connection strings that we saw can be unique to each deployment slot. So the code gets promoted and the values change for each environment. Now let's take a look at authentication. One of the great things about Azure App Service is that you can let it handle authentication for you. You just choose the provider that your user base uses, and you can use multiple providers.

You could use Azure Active Directory, so accounts need to exist there. Or you could use outside authentication providers, like Facebook, Twitter, or pretty much any service that uses this protocol called OpenID Connect. That works with OAuth 2.0, which is a standard on the web. Remember we talked about that when we talked about Azure Active Directory Domain Services. One of the scary things about turning over management of your web servers to a third party, even Microsoft, is how do you troubleshoot that when there's a problem? Same with a deployment. You don't have access to the file system directly, but you can turn on quite a bit of logging, including logs from the web server and from the application.

Those can get stored in Azure Storage as files, or you can have them written onto the local server in the App Service plan, in which case you can actually stream them from here and see the logging in real time. You can stream the logs onto your local computer using Visual Studio or PowerShell also. There's a lot more here that can help you with troubleshooting. Under the advanced tools, there's a link to the Kudu portal. That's an application that gets installed with your App Service that provides all sorts of information about the environment like system information and environment variables on the servers.

Under the Tools menu, there's a way you can deploy your web app by dragging a zip file containing the website right into the browser here. There are a lot of other ways to deploy apps to Azure App Service too, but we'll stop here. Next, we'll look at serverless computing in Azure and Azure Functions in particular.

There's always servers involved in Azure. The term just really refers to how little you might need to interact with those servers. Serverless computing is about letting developers focus on the code and business logic that they're developing and not on the underlying infrastructure. The environment is set up for you, and it scales automatically to meet demand, but you don't need to do any configuration to make that happen, even the minimal config you need to do with App Service or virtual machine scale sets. Serverless computing also differs from the other compute models you've seen in that you're only charged when the code runs, so you don't need a virtual machine or an App Service running, waiting to do the work. The two main services in Azure that are considered serverless computing are Azure Functions and Azure Logic Apps. Logic Apps don't really fall under the Azure compute category. They are now categorized as part of the integration category of services, but they're used so often with Azure Functions that it's worth mentioning here. Both of these services can be used independently, but are often used together to build solutions. Azure Functions allow you to run small pieces of code that you write yourself. Functions are started by triggers, which could be an HTTP call to the function endpoint, an event that happens in another Azure service, like a blob getting created in Azure Storage, or you can run the code based on a timer event. You can write functions in C#, Java, JavaScript, TypeScript, Python, and even in PowerShell. Azure Functions can run completely serverless, and this is called the model. But if you already have an Azure App Service plan that you're paying for, you can also leverage that to host Azure Functions. Azure Logic Apps allow you to design workflows right in the Azure portal, so you don't need to write any code with Logic Apps. You can automate business processes when you need to integrate apps, data, and services. Logic Apps have a huge library of connectors to everything from SharePoint and Azure Storage to Zendesk and SAP. When there isn't a connector that suits your requirements, you can always write code in an Azure Function and call it from a Logic App. So even though Logic Apps are very powerful, it's always good to know that when you hit a wall in terms of functionality, there's a way to write code to accomplish what you need. So for an example of how these can work together, you could create a Logic App that watches an email account for an email with attachments, then cleanses the body of the email using an Azure Function. Then the Logic App could create a blob in an Azure storage account and store the email and the attachment there. In terms of choosing one over the other, if you need a solution that calls APIs, Logic Apps are a good place to start because of all the connectors available. If your solution needs to execute custom algorithms or do special data lookups, Azure Functions would be a starting place because you already know that you need to write code. Let's create an Azure Function next.

Chapter 27 How to Create an Azure Function

This will just be a simple HTTP trigger that returns HTML to the browser. I'll start by going to All services and searching for Function Apps.

A Function App is the container that holds multiple functions. Click Create, and we get brought to the creation screen. Let's create this Function App in the resource group.

We need to give this Function App a name. And similar to Azure App Services, this name needs to be unique across Azure because it's suffixed with azurewebsites.net. So you can start to see the relationship here between Function Apps and Web Apps in the Azure App Service. Next, we select to deploy code or a container. Because I'm using the free trial subscription here, only code is available. I'll select the runtime stack as .NET, but you can see there are other options here, like Node.js and Java. I'll leave the default framework version, .NET 6, and I'll leave the default region too. You can deploy Function Apps onto Linux or Windows. It just depends on the runtime stack you've selected. .NET runs on both, but not all the frameworks do. And the plan is the most important thing here.

The plan will take care of all the sizing and scaling of the VMs for us, and we'll only be charged based on when the functions are called. The other options are grayed out because I'm using the free Azure trial. But normally, you can select to create this Function App on an existing App Service plan right alongside your other web apps and API apps. Or you can choose the Function Premium plan, which adds some network and connectivity options and avoids having to warm up the underlying VMs, so the performance can be better.

And this is where the Function Premium plan provides networking options to restrict access to only virtual networks, not the public internet. On the Monitoring tab, we have the option to enable Application Insights for deep monitoring, just like with App Services.

Once it's created, let's navigate into the Function App. So this looks a lot like an App Service already. There are deployment slots, configuration, authentication, and custom domains. But under the Function grouping, there's this Functions tab, and there aren't any functions yet. So again, the Function App is the container, and you can have multiple functions here. Okay, now let's create the function.

The first thing is the development environment. You can develop functions using VS Code or another editor with core tools installed like Visual Studio. Or you can develop right here in the portal in the editor. So let's just stick with that. Next, we choose the type of trigger, so what's going to cause this function to run? It could be an HTTP call to the endpoint, it could run on a timer, or this Function App could watch for events in Azure, like when a blob is added to a specific container in Azure Storage or a document changes in a Cosmos DB database. Let's go with the HTTP trigger.

This is how you would call the function from another program, like a Logic App for example. We can change the function's name, and we can choose the authorization level. This has to do with whether or not the caller needs to supply a key, which is just a shared secret.

So you can prevent unauthorized callers from causing this function to run and costing you money. Let's just leave it wide open for the example though, and let's create this function. Once it's created, we're brought into the function. We've got some options along the left here.

Let's click Code + Test. That opens up the editor where you can modify the default code. This just gives you a starting place to see how the function is structured. The default code will write to the log, and it will look for a string in the query string value and send back a response over HTTP. You can write some really complex functions to interact with other services and perform whatever logic you want to code. But this is an example, so let's just stick with this. Next on the menu is the Integration tab where you can see how the function is laid out and make modifications here. This is just an overview, so we won't go into this. But let's go back to the code and test, and let's run this function. Let's copy the function URL. So this is the endpoint that the caller would use. They could do that programmatically to get the results back, but let's open up a new browser tab and paste this in. I'll increase the font size.

It says the function ran successfully and that we can pass a name in the query string. So functions are an easy way to deploy small packages of business logic onto a managed environment and can provide cost savings over hosting a app service. In summary, you learned about computer options in Azure, starting with the service delivery models, then looking at virtual machines, containers, Azure App Service, and Azure Functions. Next, we'll look at networking in Azure.

Azure has a number of products for networking that allow you to create secure networks for your virtual machines and other Azure resources so those resources can communicate with each other and with the internet. The underlying physical network and components are managed by Microsoft, and you configure virtual versions of everything that you need. An Azure virtual network is a fundamental building block for your private network. A VNet enables many types of Azure resources to communicate. A virtual network has an address space that you define in Azure, which is a group of IP addresses that can be assigned to resources like virtual machines. A VNet is segmented into one or more subnetworks called subnets, which are allocated a portion of the VNet's IP address space. Then you deploy Azure resources to a specific subnet. A VM is assigned to a subnet, and VMs can communicate with other VMs on the same network. But you can apply security rules to that traffic using network security groups, or NSGs. These allow you to filter network traffic by allowing or denying traffic into and out of the subnet. Virtual machines are deployed into virtual networks, but you can also deploy other Azure resources into a VNet, networking components, like Azure Firewall, Application Gateway, and VPN Gateway. You can deploy resources like Redis Cache and Azure SQL Managed Instances, and analytics resources, like Azure HDInsight and Azure

Databricks. And Azure Kubernetes Service gets deployed into a VNet also. You can also configure App Services to have a private IP on your VNet, which enables private connections to App Services, which have traditionally only been available over the internet. By default, resources assigned to one virtual network can't communicate with resources in another virtual network. So there's some inherent security controls built in, but you can enable that communication between virtual networks using a feature called VNet peering. You can enable VNet peering between virtual networks in the same region, as well as VNets in different Azure regions, and the traffic flows privately over Microsoft's backbone network. You can connect an network to an Azure virtual network also using a VPN gateway or using a service called ExpressRoute. Virtual machines on a VNet can communicate out to the internet by default. But in order for inbound communications to take place from the internet, the virtual machine needs to be assigned to public IP address. Technically, the public IP address gets attached to the network interface of the virtual machine. So each of these is a separate resource in Azure with their own configuration. I mentioned network security groups, or NSGs. You also use these to control the inbound and outbound traffic to the internet. You can assign a network security group to the subnet or directly to the network interface of a VM. Then you can filter traffic with rules based on the source and destination IP addresses, the ports being accessed, and the protocol being used, like TCP or UDP. Now let's talk about load balancing in Azure. In order to distribute traffic between virtual machines for high availability, you can create a load balancer. There are public load balancers in Azure, which load balance internet traffic to your VMs. You can actually use a public load balancer to allow traffic to your VM without needing to attach a public IP address to the VM. And there are also internal or private load balancers where traffic is coming from inside the network. A public load balancer can provide inbound connections to VMs for traffic coming from the internet. It can translate the public IP address to the private IP addresses of the VMs inside a VNet. It's a solution that can handle a lot of traffic, but it's just a load balancing and port forwarding engine. It doesn't interact with the traffic coming in. It just checks the health of the resources. When you're exposing resources to the internet, particularly servers on your internal virtual network, you usually want more control over the traffic. That's where Azure Application Gateway can offer more features and security for publishing applications to the internet. Application Gateway is a web traffic load balancer that exposes a public IP to the internet, and it can do things like SSL termination. So traffic between the client and the App Gateway is encrypted, but then the traffic between App Gateway and the virtual machines can flow unencrypted, which unburdens the VMs from costly encryption and decryption overhead. App Gateway supports autoscaling, so it can scale up and down depending on traffic load patterns. It supports session affinity for applications that require a user to return to the same web server after they've started a session. It can do rewriting of HTTP headers and can make routing decisions based on more than just the IP address and the port that was requested. And App Gateway also uses a service called Web Application Firewall, which protects your web applications from common exploits and vulnerabilities like SQL injection attacks and scripting. So again, Application Gateway is more than just a load balancer. If you search for load balancing in the Azure portal, you'll see descriptions of all the options.

Besides Load Balancer and Application Gateway, there are two other options that relate to load balancing across different regions.

Traffic Manager allows you to distribute traffic to services across global Azure regions. Front Door has more capabilities for application delivery. We'll look a little closer at some of the major components of virtual networking in Azure. We'll create a virtual network and subnets. Then we'll create a virtual machine and attach it to the existing VNet. Next, you'll see how to use network security groups to allow traffic to the VM from the internet. After that, we'll peer two virtual networks so the VMs on the VNets can communicate. Then we'll discuss the options for connecting networks to Azure using VPN Gateway and then ExpressRoute. Next, we'll discuss Azure DNS for managing DNS services alongside your other Azure resources. And finally, we'll talk about private endpoints in Azure, which bring platform services like App Services and storage accounts into your private virtual network. So next, let's create a virtual network.

Chapter 29 How to Create an Azure VNET

I'll start by going to All services and searching for virtual network.

Click this, and there aren't any created yet, so I'll click Create. As always, we need a resource group. I'm going to create a new one, so I'll just give it a name, and let's call this vnet1, and I'll place it in the closest region to me. Next, let's configure the IP address space for this VNet.

This is called CIDR notation. The number after the slash tells how many addresses are in the range, starting at the number before the slash. So a /16 means there are about 64,000 IPs available in the address space. We can assign these IP addresses to virtual machines and other services that can be addressed within a VNet. When you configure your IP address space, the IP addresses are private to your VNet. The only time it matters is when you want to connect this VNet with another network, like another VNet in Azure using peering or with your local network using VPN Gateway or ExpressRoute. In that case, those IP addresses can't overlap with this address space. We can break this up into smaller blocks of IP addresses using subnets, and then we can apply security to a subnet. I'll call this WebSubnet because I want to put web servers in this subnet. I'll give it an address range that starts within the range of the VNet and has a smaller block of IP addresses. In CIDR notation, a larger number means a smaller group of IPs. So a /27 is only 32 IP addresses. We can attach service endpoints to the subnet.

Service endpoints allow traffic to specific services in Azure over the Microsoft backbone. So VMs on the subnet could connect to Azure SQL or Azure Storage without having to connect to the public endpoints on the internet. Let's create this subnet, and let's create another subnet. Let's call this AppSubnet. So I might put application servers on this subnet and have them only accessible from VMs on the web subnet. Then you can use network security groups to enforce that. I'll give this subnet a range that starts higher than the highest IP available in the WebSubnet. And let's make this a /24, so there are 256 IP addresses available on this subnet.

Let's add this, and let's move on to the Security tab. Here, you can enable a bastion host, which is a VM that lets you remote into the virtual machines in this VNet without having to connect to them directly, so that's for security. We can enable DDOS standard protection.

Every VNet comes with basic protection against distributed denial of service attacks. By enabling standard protection, you get additional metrics and access to experts within Microsoft if an attack is launched against one of your applications. That comes for an additional charge, which is why you have to enable this. Azure Firewall is an intelligent firewall security service. It can watch for patterns and alert you to traffic coming from known malicious IP addresses and domains and deny that traffic. But let's leave these off, and let's move ahead and create this VNet.

Once the VNet is created, I'll go to the shortcut to all VNets. On the menu here, you can see the IP address space we configured, and you can modify it from here.

You can create and remove subnets. And down here, you can specify the DNS servers to use. You can let Azure handle DNS resolution for you, or you can add the IP address of your own DNS servers.

DNS resolves domain names to the IP addresses of servers, so you might create your own network in a VNet with a VM for Active Directory, VMs for applications, and a VM for hosting DNS services. You might do that if you're setting up a lab environment in Azure here, for example. Setting that VM as the DNS server here will allow all the VMs on the network to resolve your internal domain names to the IPs of the servers, but you can actually use a service called Azure DNS for that also. And on the tab for peerings, you can peer this VNet with other VNets in Azure, so the VMs and resources can communicate. If you have any resources that have been assigned IP addresses on the VNet, their network interfaces will show up here. We don't have any, so let's create a virtual machine next and add it to this VNet.

Chapter 30 How to Add Virtual Machine to VNET

So we have a VNet, but it doesn't have anything on it. So let's create a virtual machine just like we did earlier, but this time, we won't create a new virtual network at the same time. We'll add the VM to the existing VNet. We went through all this earlier, so I'll move quickly through this. Let's put this VM in the same resource group as the VNet. We don't have to, but that'll make it easier to delete everything at once later. But this VM does need to be in the same region as the VNet. Let's turn off the availability options and change the VM image from Linux to Windows Server.

I need to enter a username and password for the local administrator account. Now let's move to the Disks tab, and I'll leave the defaults.

And Networking is where I want to assign this VM to the existing VNet. Remember the VM is being created in the same Azure region as the VNet is in so that VNet is available in the list here.

You also need to select which subnet to put this virtual machine on. Technically, it's the network interface attached to this VM that will get the IP address from the subnet. The default for security is that a network security group will get created and assigned to the network interface of this VM. But we're going to create a separate network security group and assign it to the subnet, not to the

VM. So I'll turn this off for now. That's all I want to change in the defaults, so let's skip ahead through these tabs and create this VM. It'll take a minute or so to create this, and once the VM is created, let's navigate into it from here. I'll go to the Connect screen.

This is where we downloaded the RDP file to connect into the VM. It says here that the port prerequisite is not met. That's because there's no network security group that will allow traffic to this VM from the internet. Let's download the RDP file and try to connect anyway just to be sure. I'll open the file and click Connect, and I don't even get to the login screen because Azure won't allow the connection to the VM over port 3389.

So let's fix that in the next chapter.

Chapter 31 How to Create a Network Security Group (NSG)

We can't connect into this VM from outside the Azure VNet. Let's fix that by creating a network security group, or NSG, and opening up port 3389 to the internet. I'll go to All services again and search for network security. Click on Network security groups, and there aren't any yet in the subscription. This is a different subscription from the free trial that I used earlier in the course when I created that VM. Otherwise, the network security group attached to the network interface of that VM would show up here. I'll create a new network security group. We need to put this in a resource group, so I'll use the same one as the VNet and the VM, and that updates the region for me. And I'll give this a descriptive name.

That's all the configuration you can do when you create the NSG, so let's skip tags and create this. Once the NSG is created, let's navigate into the resource. On the Overview tab, it shows the default inbound and outbound security rules.

Network security groups allow you to permit or deny traffic between sources and destinations and to be specific about which ports and which protocol are permitted or denied. Before we add an inbound rule to allow port 3389 from the internet, let's just verify that this NSG isn't attached to a network interface for a virtual machine. If it was, that would show here. And this isn't associated with any subnets either, so let's do that first.

I'll select the VNet that's in the same region as this NSG, which is the one we want and then associate this with the subnet that the VM is on, which is the WebSubnet. Now the security rules of this network security group are being applied to the subnet that the VM is on. So let's go to Inbound security rules, and let's add a new rule. I won't go through each of these options. But if you wanted to allow HTTP traffic from the internet to a web server in the subnet associated with this NSG, you would allow ports 80 to the IP address of the VM or to any VM on the subnet, which is the default here, Let's do something similar, but allowing RDP traffic, which changes the port to 3389.

And that's all we need to do. So let's just change the name of the security rule and add it to the NSG. Now it shows at the top of the list because it has higher priority than the other rules.

These rules are processed based on priority, the lower number taking precedence. So this RDP rule overrides the rule at the bottom to deny all inbound traffic. Now let's go back to the list of virtual machines and drill into this VM and go to the Connect tab.

Now it shows that the inbound access port check past, so we should be able to connect using RDP. Let's download the RDP file, although we could use the one we downloaded before. Nothing's really changed. And I'm getting brought to the login screen, so that's progress. I'll enter the credentials of the local administrator on this VM. Okay, I can accept the certificate errors, and I'm brought into the remote desktop of the virtual machine in Azure. So the network security group enabled access from the internet from my local computer. Let's just wait until the connection is completed. Now I'll open up the command prompt, and let's run the ipconfig command. This shows us the IP address of this VM. It's been assigned an IP address on the subnet of vnet1.

Next, let's see how to set up VNet peering, so this VM can access a VM on another VNet.

Chapter 32 How to Peer Virtual Networks

Now let's see how to allow resources in different virtual networks to communicate with each other by peering the virtual networks. I've created another VNet for this demo. The type is kind of small here, but the VNets have different address spaces, and the IP addresses in each address space don't overlap.

I've also created another virtual machine. And if I drill into it, it shows here that the VM network interface is attached to a different VNet and subnet. The private IP address of this VM on the VNet is 10.0.0.4. Let's remember that.

I've still got the Remote Desktop window open to the VM on the other network, the one that we created earlier. Let's open up the command prompt. It still shows the results of ipconfig, which shows that this VM is on the first VNet, vnet1. Let's try and ping the VM on the other VNet. I've turned off the firewall on that VM by the way, so it will allow ICMP traffic, which is what ping uses. The request is timing out as expected.

Let's go to all virtual networks, and let's go into vent1. You can actually do this from either direction, vnet1 or vnet2. Let's go to Peerings on the menu and add a peering.

I'll give this link a descriptive name to show what it's meant to do, and the default is to allow traffic between the VNets. It's going to create a peering in the other VNet, so we need to give that one a name too. And now let's select the VNet to peer with. I'll select vnet2 and accept the defaults, and let's add this peering. That's all there is to it.

It says the status is connected. Let's navigate to vnet2 from the link here, and let's go to Peerings, and there's the other side of the peering.

Now let's see if we can ping the virtual machine in vnet2 from the virtual machine in vnet1. I'll open up the Remote Desktop session again, and I'll just press the up arrow key to show the last command and hit Enter. Now the ping is working, so we successfully peered the two VNets.

So that's how you can allow VNets in Azure to communicate with each other. Next, let's see how to allow networks to communicate with resources in Azure VNets.

Chapter 33 Azure VPN Gateway Basics

When you want to connect your network to an Azure VNet, there are a couple of ways to do it. In this chapter, we'll talk about VPN gateways, and in the next, we'll talk about Azure ExpressRoute. What does it mean to connect your networks? It means that from the computers and servers joined to your network, you can access the virtual machines and other Azure resources that have private IP addresses on that VNet in Azure. To the users on your local network, there's no difference accessing an application on a web server in Azure than there is accessing one on the local network, and that web server in Azure doesn't need to be exposed to the internet. The connection from is taking place over a private secure connection. To make that connection between networks, you can use a VPN gateway in Azure. VPN Gateway creates a private encrypted tunnel over the public internet. If you connect to your local office now using a VPN, it's basically the same thing. Azure VPN Gateway is made up of one or more VMs that get deployed into a subnet in your Azure VNet. That subnet needs to have a specific name, and you can't configure the VMs for the gateway subnet. You connect to the VPN gateway through its public IP address. If you're connecting your entire network to Azure, then the VPN gateway needs to connect to a VPN device on your network that has a public IP address to the internet. The traffic between the network and the Azure VNet flows through the gateway. There are a few different types of connections. Let's take a look at the documentation to see some diagrams.

You can create a VPN gateway that connects to your network. One major stipulation is that you have to make sure that the IP address ranges of the VNet in Azure don't overlap with your IP addresses. A virtual network in Azure can only have one VPN gateway, but you can make multiple connections to it.

So if you have different regional offices with different networks, you can connect them to the same VPN gateway and the same VNet in Azure. There's a second type of connection possible, which is called a VPN. This is where a single computer connects to the VPN gateway.

You might use this if you're working from home. It doesn't require the local computer to have a public IP address or a VPN server, like with the VPN, but you need to authenticate using certificates uploaded to Azure and on your local computer. It's also possible to use VPN Gateway to set up a connection between two VNets in Azure.

That's an alternate to VNet peering. You might do this if you have older VNets in Azure that were created with the classic deployment model, and you want to join them with newer VNets that use the resource manager model. You can't have more than one VPN gateway in an Azure VNet, but you can connect to a VNet using both VPN Gateway and ExpressRoute. The difference is that VPN Gateway uses the public internet, and ExpressRoute uses a private connection that's not over the public internet.

You might do this in order to use VPN Gateway as a failover if, for some reason, the ExpressRoute connection isn't available. That's a lot cheaper than maintaining a backup ExpressRoute connection.

The other type of connection you can make from your network to Azure is an ExpressRoute connection. The connection is made through a service provider that's partnered with Microsoft. You connect to the service provider, and they connect directly to the Microsoft edge servers. So the traffic doesn't get routed over the public internet. There are actually two ExpressRoute circuits created, so there's redundancy. ExpressRoute can connect your network to multiple VNets in Azure, which is called private peering. You can also connect to Azure public services, like App Service endpoints and storage accounts, as well as Microsoft 365, which used to be called Office 365, and Dynamics 365, which is a Software as a Service customer relationship management system. This type of connection used to be called public peering, but it's now called Microsoft peering. ExpressRoute requires you to work with a provider, and they're partnered with Microsoft to connect to Azure. These providers have infrastructure at data centers where they're collocated with Microsoft edge servers. Examples of these providers are companies like AT&T or Verizon, but there are many regional providers, and they're listed on docs.microsoft.com by location. Each ExpressRoute circuit has a fixed bandwidth, and you choose a plan between 50 Mbps and 10 Gbps. The speeds available depend on the service provider that you work with. There's also something called ExpressRoute Direct where you can establish a direct connection to Microsoft's global network at peering locations around the world. This gives you increased speed and encryption options and, of course, increased cost. So ExpressRoute direct is for big corporate clients with major security requirements like banks and government. You can choose to be charged based on how much data you transfer out of Azure, which is metered billing. Inbound data is always free, or you can pay a monthly fee for unlimited data. The standard ExpressRoute plan gives you access to all the regions in a geopolitical area, but there are two other plans available. The local plan gives you access to only one or two Azure regions near the location where you're peering. You don't pay additional charges for egress for the data coming out of Azure, so this can be economical for large data transfers. And the ExpressRoute Premium gives you global connectivity to any region in the world. So if there's a lot of data moving between your network and Azure, ExpressRoute can give you the best performance. Let's talk about Azure DNS next.

Azure DNS is a way to manage your DNS records right in Azure alongside all your other Azure resources. First, let's review what DNS is. When you create a service in Azure, like an App Service or an Azure Function, they get a name that's suffixed with azurewebsites.net. And when you create a VM, there's a public IP address. You can also add a DNS name label to the public IP address of the VM right in the Azure portal, so you don't have to use the IP address to access it. But again, it ends in an address that you don't control. It's the region name, then cloudapp.azure.com. But when you're publishing your applications for clients, you'll want a custom domain name. Usually, your application name .com or .net or ending in a suffix, like .ca for Canada. You purchase the custom domain name from a domain registrar, and they're responsible to make sure that no one else owns the domain name. If it's available, you pay an annual fee to reserve the use of the domain name. There are a lot of domain registrars, and the biggest one is GoDaddy. Once you own the domain name, it needs to be hosted by a DNS provider. DNS stands for domain name system, and it's the network of DNS servers all over the world that resolve domain names to the IP addresses of the servers that host the corresponding applications. Your DNS provider makes sure that all those servers can find your domain name and resolve it. Often, when you purchase a domain name, that same company, like GoDaddy, will often let you host the DNS entries with them, but you don't have to. You can host them with any DNS service, and that's what Azure DNS is. Azure DNS is a hosting service for DNS domains that provides name resolution by using Microsoft Azure infrastructure. That's the definition right from the Microsoft docs. Azure isn't a domain registrar. Azure doesn't register your domain name, but they can manage the DNS for it. You can actually purchase domains in Azure using a service called App Service Domains. But App Service Domains actually use GoDaddy for the domain registration and Azure DNS to host the domain name resolution. So you can purchase app service domains for Azure App Services, and everything gets configured for you to point the domain name to the app service. But because they're managed by Azure DNS, you can actually modify the DNS records to point the domain to another Azure service, like a virtual machine or an Azure storage account. If you host your domain name with another DNS provider, you can transfer it to Azure DNS. Then you can manage the DNS records alongside all your other Azure resources, which means you can enable access control to control who in your organization has access, and you can get activity logs to monitor when records are modified, which can help with troubleshooting. DNS domains are hosted on Azure's global network of DNS name servers so that helps with reliability and performance. Besides managing DNS for public domains so they can be resolved from the internet, Azure DNS can also support private DNS domains. That means you can use your own custom DNS domains in your private virtual networks rather than have to set up your own DNS servers on your VM on your VNet. It's great to be able to manage public and private domains in one place. Azure DNS also supports alias record sets. That means you can use an alias record to refer to an Azure resource, like the public IP address of a VM or a content delivery endpoint. Then, if the IP address changes, Azure automatically updates the records. If you manage your DNS records outside of Azure, you might not update those records right away if there's a change in Azure, and then users won't be able to access your service. So having that be automatic with Azure DNS can be really helpful. Next, let's talk about private endpoints.

Private endpoints allow you to essentially bring a public Azure service into your own VNet, so the service can get referenced with a private IP address. When you create a service in Azure, like an App Service, it has public endpoints. That means there's an address on the internet where that resource can be reached. For App Service, that's a URL, the name of your App Service, then azurewebsites.net. That resolves to an IP address. But in certain situations, that IP address can actually change, like during the renewal of an SSL certificate. So really it's the URL that's the endpoint. Azure takes care of the resolution to the underlying IP address for other services, like Azure Storage. You can go to the Endpoints tab and see the public endpoints for your instance of the service. In this case, the endpoint is the name of your storage account, then the individual service in Azure Storage, like the Blob Service, File Service, or Queue Service, and then it's always core.windows.net. These endpoints allow people to reach the blobs and files in the particular service from the internet, but you'll see that you can apply security to the endpoint But, what about if you don't want your App Service or storage account to be accessible from the internet? You only want resources on your own Azure VNet to be able to access the service, so virtual machines on your VNet. Or if you've connected your network to an Azure VNet, you want your users to be able to access the App Service or storage account only through that secure connection and then disable access from the public internet. You can do that by creating a private endpoint for the Azure service. A private endpoint is a network interface that uses a private IP address on your virtual network. So the App Service or storage account gets a private IP address on your VNet, and you can access it privately and securely using that IP address. For some services, like Azure App Service, creating a private endpoint will automatically prevent access to the public endpoint, so from the internet. For other services like Azure storage, the public internet access isn't automatically disabled, but you can configure that yourself. Private endpoints use a service called Azure Private Link. With Azure Private Link, you can create connections to Azure Platform as a Service offering, and your connection between your VNet and the service travels over the Microsoft backbone network. In order to create a private link to an Azure service, the service has to support it. On docs.microsoft.com, you can see a list of all the resources in Azure that support private link.

So besides App Services and storage accounts, Azure Cosmos DB supports private endpoints, Azure Container Registry, Azure Service Bus, Azure SQL Database. So chances are if you want to create a private endpoint to a platform as a service offering in Azure, it's probably possible to do it. Let's quickly see how to set up a private endpoint for an App Service. I'll open up the App Service we created earlier. If you go to the Networking menu item, you can configure some options here, including private endpoints.

This is available because this App Service was created on the standard pricing tier. If you created yours on the free tier, private endpoints will be grayed out. Let's click this, and here we can create the private endpoint. I'll click Add and select the VNet where I want the private endpoint created and the subnet.

Then I just need to give this a name because it will get created as a separate resource. It'll take a minute or so to create this, and you can watch the status from the Notifications tab at the top. Once it's created, we could navigate to the resource from here, but let's go to the VNet where it was created. That was vnet1. Under Connected devices, you can see there's the network interface for the virtual machine that was added earlier, and here's the network interface for the App Service.

And you can see they both have private IP addresses from the VNet, and they're part of the web subnet. So that's how you can create a private endpoint for an App Service. In summary, you learned about networking in Azure, starting with an overview. Then you saw how to configure some of the main services, first virtual networks and subnets, then network security groups. After that, you saw how to peer virtual networks in Azure, and then you learned about the ways to connect your network to Azure using VPN Gateway and ExpressRoute. Then we talked about Azure DNS and how to configure private endpoints to services in Azure. Next, we're going to look at data storage options in Azure.

Chapter 37 Azure Data Storage Options

Modern applications require data to be available quickly and securely from all over the world, and users expect to be able to access, share, and update their data from different devices at any time. Organizations are creating more data than ever, so storing data in the cloud requires addressing new problems in a flexible way, as well as solving old problems in new ways. Azure provides a variety of cloud storage services for different types of data that allows you to choose the storage service that's best optimized for your data and to include several strategies in the same solution, if needed. But common to all the storage solutions in Azure are important benefits like automated backup and recovery, replication across the world to protect your data against unplanned events and failures, encryption capabilities, and security through things like integration with Azure Active Directory for authentication, and developer packages, libraries, and APIs that can make data accessible to a variety of application types and platforms. Data generally falls into one of three general categories. Structured data is data that adheres to a schema, usually data stored in a database with rows and columns. It's generally referred to as relational data. Azure lets you host databases on virtual machines just like you would where you're responsible for managing and patching the database product, but Azure also has managed offerings which provide convenience and scalability. For SQL Server, there is Azure SQL Database, and there is also Azure

Database for MySQL and Azure Database for PostgreSQL, which are all managed Platform as a Service offerings. Unstructured data is data that doesn't adhere to a schema and is usually data stored in different file formats, so PDF documents, JPEG images, video files and JSON files. For that data, Azure Storage provides highly scalable solutions with Azure Blob storage and Azure File storage. File storage can be attached to virtual machines using the SMB protocol, similar to file shares, but both types of storage also offer REST APIs, so data can be securely accessed over the internet. Azure Storage also stores large files like disk images and SQL databases. data doesn't fit neatly into tables, rows, and columns. It's often called NoSQL or data, and it usually uses tags or keys that organize the data and provide a hierarchy. For this type of data, Azure offers Cosmos DB, which is a globally distributed service to store data that's constantly being updated by users around the world. Being able to provision these different types of storage solutions quickly and in a cost effective way helps you respond to business change without the need to procure and manage the costly storage media and networking components required to connect it all together. This makes data storage a very strong value proposition for moving to Azure. We'll be looking at Azure Storage accounts. This is the focus of the data storage portion of the exam. We'll look at redundancy options for storage accounts, so making copies of your data in different locations. We'll create a storage account and explore the features of blobs and files in Azure Storage. Then you'll learn about some of the options for transferring data into Azure using online methods with tools like Azure Storage Explorer and a utility called AzCopy. And for transferring larger amounts of data, there is a service called Azure Data Box where Microsoft will send you hard drives for you to copy your data onto and send back to be copied into Azure. We'll discuss migrating other types of workloads to Azure also, like servers and applications using a service called Azure Migrate. Let's get started by looking at Azure Storage.

Chapter 38 Azure Storage Accounts

Azure Storage is a set of services in Azure that provides storage for a variety of data types using a few different services. Those services are managed under an Azure Storage account. The Blob storage service is for unstructured data like files and documents. Then there is File storage that's similar to Blob storage, except that it supports the SMB protocol, so it can be attached to virtual machines like a network drive, and this makes migrating traditional applications to the cloud much easier. There is Disk storage which stores the virtual machine disks used by Infrastructure as a Service VMs. There is the Table storage service that lets you store structured data in the form of NoSQL data, similar to the data you can store using Cosmos DB. And finally, there is the Queue service that's used to store and retrieve messages to help you build asynchronous reliable applications that pass messages. Let's talk about some of the general features of Azure Storage. Azure Storage is durable and highly available. Your data is stored three times in the primary data center by default, and you can choose other replication options that copy the data automatically to other regions in Azure. Your data in Azure Storage can be reached over HTTPS from the internet and each of the storage services in an Azure Storage account has its own REST endpoint, but of course, you can apply security controls to those endpoints to prevent unauthorized access. Security is a big topic for storage accounts. At a high level, when you want to control access to the data plane of a storage account to allow access to the data, you can provide access using access control for users with identities stored in Azure Active Directory and that works for the blob, file, and table services in your storage account. Or you can provide a storage account key that gives access to the entire storage account. You can also provide a user with something called a shared access signature. A shared access signature is a security token string, and it can scope access to a particular service like only the Blob service, as well as to a particular container or even an individual blob, and it can also scope access to a range of time and a particular set of permissions like only allowing reads or updates or deletes. A shared access signature gets appended to the end of the URL to a blob or file in Azure Storage so you're able to have pretty access control to data in Azure Storage using shared access signatures, and data in your storage account is encrypted. You can even use your own encryption keys. Besides accessing your data using the REST endpoints, there are SDKs for a variety of languages like .NET, Java, PHP, and others, as well as support for scripting in PowerShell and the Azure CLI. Microsoft also offers free tools like Azure Storage Explorer, which provides a graphical user interface and a utility called AzCopy to make it easy to move data into and out of your storage account. There are four types of storage accounts, standard general purpose v2 storage accounts support blobs, file shares, queues, and tables. This is the recommended storage account type for most situations. It offers the most redundancy options, meaning you can have copies of your data in other regions. Premium Block Blob storage is for storing blobs only and it's for scenarios when you need high transaction rates and low latency. You're limited to only storing your data within a single Azure region though. Premium file shares are for high performance file storage, but you could only store files with this type of account, and again, you're limited to storage within a single Azure region. Premium page blobs are for storing larger blobs like databases and VMs for disks. You can store these types of files in general purpose v2 storage accounts too, but the premium page blob account type gives you better performance when it's needed. Again, your redundancy options are limited. All of the premium account types use drives for low latency and high throughput. You can't change the storage account type after it's been created. You would need to create a storage account of a different type and move your data over. Next, let's talk about redundancy options for Azure Storage accounts.

Redundancy with Azure Storage is about protecting your data from unplanned events like hardware failures, network outages, and even natural disasters. You do that by making copies of your data, which is called replication. Your data is always replicated in the primary data center, you can just expand that with other options. There are three categories that group the redundancy options: redundancy in the primary region, redundancy in a secondary region, and read access to data in the secondary region. Let's start with the primary region. Locally redundant storage is the lowest cost replication option. Your data is copied three times within a single physical data center. It protects you from failures of a server rack or a disk drive within the data center, but because all the data is within a single data center, there is still the risk of a data center level disaster like a fire or a flood. To help mitigate that risk, the next storage type is storage. This storage replicates your data across three availability zones in a single region. Availability zones are data centers within a region that have their own separate power, cooling, and networking. storage isn't available in every Azure region. Locally redundant storage and storage provide redundancy in the primary Azure region that your storage account is located in, but in the event of a regional disaster where multiple availability zones are affected, there are other options that allow you to copy your data to another Azure region. storage copies your data three times in the primary region within a single data center and also copies the data asynchronously to a single location in a secondary region. The data is copied three times in the secondary data center, it's basically locally redundant storage in two regions. The secondary region is decided by Microsoft and you can't change that, but it's selected to be hundreds of miles away from the primary region to prevent data loss in the event of a natural disaster. Microsoft lists the paired regions on their website. The second option for redundancy in a secondary region is storage. This replication option uses storage in the primary data center and locally redundant storage in the secondary data center. With and storage, the data in the secondary region is only available to be read if you or Microsoft initiates a failover from the primary region to the secondary. You might want to always have the ability for your applications to read the data in the secondary region. To enable this, there are two other account types, storage and storage. The two options are similar to the previous ones we discussed, and storage. They just add the ability to always be able to read the data from the secondary region. The replication options available depend on which storage account type you select. Let's create a storage account in the Azure portal.

Chapter 40 How to Create a Storage Account

Let's create an Azure Storage account so we can explore some of the features and configuration options. As usual, I'll go to All services and search for storage. Clicking on Storage Accounts brings us to the list of storage accounts in this subscription. We could have also gotten there from the shortcut on the left menu. From here, let's create a new storage account. The first thing we need to do is choose a resource group for the storage account, I have one created already. Then we need to give the storage account a name and this name needs to be unique across all of Azure because it will become part of the URL to each of the storage endpoints for the blob, file, table, and queue services. Now we choose the region. Before I change the default, I just want to show you how the choice of region affects the redundancy options available.

For the East US region, the options are so just in a single region, so with failover to a secondary region, and storage. If I change the region to Canada Central, we have all the same choices plus storage, so that's not available for all Azure regions, but let's go with storage, the cheapest option. Next, we choose the storage account type which is labeled as performance. The standard selection is for a v2 storage account type, but we can change this to Premium and choose from Block blobs, File shares, or Page blobs.

These account types store data on drives, but they're limited to just the type of data in the description, so file storage won't let you store blobs, tables, or queues. Each of these options offers better performance and might be suitable depending on your business requirements, but you'll notice that choosing any of these limits the redundancy options to just storage. Let's change this back to Standard and move to the Advanced tab.

Here, you can choose from some security options like requiring HTTPS on the REST API to the storage account allowing anonymous access to Blob storage, then there is options to Enable hierarchical namespace for the Blob service, which will allow it to be used for Data Lake Storage Gen 2. That adds some security options, and the hierarchical namespace makes it easier to do the type of bulk operations required by big data analysis tools. You can choose the default Access tier when uploading blobs.

Access tiers can have a significant impact on the cost of storing your data, and you can enable large files for Azure File storage.

Normally, file shares are limited to 5 TB in size, but by choosing this option, you can create file shares up to 100 TB. Let's move on to networking.

The default is to Enable public access, so this includes the internet and this is for the REST endpoints to all the storage services, which of course, you can secure by requiring the caller to authenticate or you can specify virtual networks within Azure and public IP addresses on the internet that you want to restrict access to. And you can also select to only have the storage account available over a private endpoint. Let's use the default and move on to data protection. You have the ability to Enable soft delete for blobs, containers, which hold blobs, and for file shares.

This essentially gives you a recycle bin where you can recover deleted files and folders. Then there are some options for versioning of blobs and enabling the change feed to record when blobs are created, modified, and deleted. Let's move on to encryption. The data in your storage account is encrypted by default, but you might have a need to manage your own encryption keys, maybe for regulatory compliance. You can do that using an encryption key that you store in Azure Key Vault.

We haven't talked about Key Vault, but it's a central service in Azure to securely store secrets like encryption keys, SSL certificates for web apps, and strings, like database connection strings. They can all be managed in one place and accessed by applications that have permissions. Let's leave the default and let's go to Tags. We won't create any tags. So let's review the configuration and create this storage account.

Once the storage account is created, we can navigate to it by going to all storage accounts and there is the new storage account we created. Before we explore the storage account, let's talk about blob access tiers next.

Chapter 41 Azure Blobs and Access Tiers

Now let's talk about blobs and how you can save on storage costs by using a feature called blob access tiers. BLOB is an acronym for binary large object. A blob can be any type of file, including documents, video files, text files, even virtual machine disks. The Blob service is optimized for storing massive amounts of unstructured data, and by unstructured data, I mean data that doesn't adhere to a particular data model or definition. There are three types of blobs you can store: block blobs, stored text, and binary data. They're called block blobs because a single blob is made up of multiple blocks and that helps you optimize uploading. Append blobs are made up of blocks also, but they're optimized for appending only, so they're a good choice for logs where you only add to the file. And page blobs store random access files up to 8 TB in size, so they're used to store disks for virtual machines and databases. Page blobs are made up of pages and are designed for frequent random read/write applications, so they're the foundation of Azure Disks. Block Blob storage is the most cost effective way to store a large number of files, and one of the features that helps you save is blob access tiers The cost of Azure storage comes from the amount of storage, as well as transaction costs related to accessing the data. So there are three blob access tiers that you can choose from to tailor these costs to the way that you use your data. The hot access tier is for data that's accessed frequently so it has the highest storage cost, but the lowest transaction costs. Cool storage is for storing data that you don't access frequently, so the storage cost is lower, but the transaction cost is higher when compared to the hot access tier. And the archive access tier is for storing data that you rarely access. It's very inexpensive to store a large amount of data, but you have to be willing to wait hours to rehydrate the data if you do need to access it, but for organizations that have requirements to archive large amounts of data, there can be big cost savings by using the archive tier. With some data, the need to access it drops as the data ages, so Blob storage also has a feature called lifecycle management, which lets you set policies to move blobs between access tiers so you can design policies that provide the least expensive storage for your needs without having to manually move the data around. Blob storage has a lot of other features too, like creating snapshots of blobs, leasing blobs to prevent other people from modifying them. You can enable soft delete to essentially provide a recycle bin for your blobs, and you can even host static websites directly in Blob storage so you don't have to host simple HTML and JavaScript sites in Azure App service or on a virtual machine. Blob storage also integrates with other Azure services like the content delivery network, so you can optimize the delivery of blobs to clients all over the world. Azure Search integrates with Blob storage too so you can index the contents of blobs, which enables searching inside the contents of the documents like Word docs, PDFs, Excel spreadsheets, PowerPoint files, and lots of other types. So the Blob service and Azure Storage accounts can be an integral part of applications that use unstructured data. Let's talk about File storage next.

You can attach File storage to virtual machines to act like network drives. The file share will show up as a drive letter in the virtual machine, just like storage. When you're moving applications from to the cloud, there is inevitably going to be some applications that rely on data or configuration being stored on a file share. With the SMB support that Azure Files brings, you can migrate those apps much easier. Something that distinguishes Azure file storage from traditional file shares is that you can make the files accessible from anywhere in the world using a URL to the file with a shared access signature appended on the end. Azure file shares can be mounted concurrently by cloud or deployments of Windows, Linux, and macOS. In order to map an Azure file share to using the SMB protocol, you need to open port 445, which is used by SMB. If your organization requires that port 445 be blocked, you can use Azure VPN Gateway or ExpressRoute to tunnel traffic. You'll need to set up a private endpoint for your storage account to do that though. Moreover, Azure File shares can be cached on Windows Servers with Azure File Sync. That provides you fast access near where the data is being used. It actually allows you to tier files based on how they're used. You can keep recently accessed files on your servers while seamlessly moving old and files that aren't used as frequently to Azure. This helps you manage unpredictable storage growth, and essentially turns your file server into a quick cache of your Azure file share. You do that by installing a sync agent on the local server. Azure

Files has different storage tiers. There are premium file shares, which are part of the File Storage storage account type. We talked about storage account types in the overview on Azure Storage accounts. Premium file shares run on drives so you get high performance and low latency, so milliseconds for most input/output operations. Transaction optimized file shares are backed by hard disk drives, and therefore, transaction heavy workloads that don't require the low latency of the premium tier. These are good for applications that require File storage as a store. Hot file shares are for file sharing, like team shares, and it works well as the storage for Azure File Sync also. Cool file shares offer the most storage for offline archive storage. Hot and cool file shares are similar to the access tiers that you learned about for Blob storage. You choose the storage tier when you create the file share, but with the tiers on the v2 storage account type, you can change the storage tier after the file share has been created. So next, let's explore the storage account that we created earlier and upload some data.

Chapter 43 How to Explore Azure Storage Accounts

Let's explore some of the features of the Blob and File services in the Azure Storage account that we created earlier. From the list of all storage accounts, I'll open that one. From the menu on the left, you can access each of the services in this general purpose v2 storage account.

Containers are the Blob service, File shares are the File service, and the Queue and Table services. Below those are general settings for the storage account. Let's look at blob containers. Containers are the logical groupings of blobs. You can think of them like folders, but they're really just a path in the URL to the file. Let's create a new container. I'll just give it a name and you can specify an access level.

This means that you can require everyone accessing blobs in this container to authenticate, which is the private option, or you can allow anonymous read access to just the blobs and not to be able to list the container contents or the container option allows anonymous access to both the blobs and the container. Let's leave this as private and create the container. I'll navigate into the container and it's empty, so let's upload some data. I'll go to my desktop and there are some files in this folder. I'll select them all and click Open. Under Advanced, we can specify the blob type, and we can also specify the access tier.

So if this data is meant to be archived, we can send it right to the archive tier during the upload. We can also specify a folder, which is similar to the container in that it's just part of the path in the URL. Once the upload is complete, I'll close this window and let's click on one of these files. So this just opens the metadata about the file, not the file itself. From here, we can change the access tier of this blob. On the right side, there is a menu for each blob.

This is where you can download the file or create a snapshot so a version of the file. You can acquire a lease to prevent other users from modifying the file and you can generate a shared access signature. Let's click that. This lets us choose the parameters for the token, like the permissions we want to give, how long it's valid for, and the IP addresses the user must be coming from in order to access this blob.

Let's generate this with the defaults. That gives us a URL to the blob with the shared access signature token appended on the end, but let's close out of the container and go back to the Storage Account menu. Now let's go to File shares and let's create a file share.

You need to give the file share a name and then you can choose the storage tier. Premium isn't available because the storage account was created with a general purpose v2 storage account type, but you can choose between Transaction optimized, Hot, and Cool storage tiers. Let's create this file share. And once it's created, I'll click to open it. From here, we can upload files and change the storage tier. We can also generate a script to run on a virtual machine in Azure, then it will show up on the VM with a drive letter. We can do that for Windows, Linux, and macOS because remember, it's also possible to connect to this file share from Let's close out of this and let's go back to the root of the storage account. I want to show you some of the configuration options here. Under configuration, you can change some of the settings we chose when setting up the storage account like the default Blob access tier and the Replication option.

We could change this to storage. Farther up the menu, there are a number of options under Security and networking. You can create a shared access signature to allow access to the services within the storage account, so this provides more access than the token we generated to the individual blob, and you could also provide the access key to the entire storage account.

This isn't recommended though because it provides full control of the storage account. You can rotate the primary and backup keys here which would remove that access, and the shared access signatures are actually generated using these keys too, so if you wanted to revoke the shared access signatures before they expire, you could just rotate the keys here. On the Networking tab, you can disable access from the internet and you might do this if you set up a private endpoint or you can enable access from your virtual networks.

On the Azure CDN tab, you can integrate the storage account with the Azure Content Delivery Network, which would cache the data in locations around the world for faster access times. We won't go into the Azure CDN here, but it can really increase speed and availability of files to users in locations outside the region where the storage account is located.

Farther down the menu, there is an option for Static website. This allows you to host static HTML pages and scripts.

This is less expensive than hosting a full app service when all you need is a website without any processing or frameworks installed. Azure Search lets you integrate your storage account with the Azure Search service so your blobs can be indexed and searchable, that includes being able to search the contents of a variety of file types like Word documents and PDFs.

You need to create a search service in order to do that though. We won't go into Azure Search in this book. When you want to access a blob in the Blob service, it would be at this endpoint, followed by the container name and the blob name. Uploading files manually using the Azure portal can be quite so in the next chapter, let's look at other ways to manage files in Azure.

Chapter 44 Azure Data Transfer Options

Let's talk about some of the options for moving data into Azure Storage. The approach you choose depends on a few factors, the amount of data you need to transfer, the frequency, meaning is this a transfer or will you be periodically pushing data into the Azure Storage account, and the last factor is network bandwidth. If you've got a slow connection, transferring large amounts of data can take quite a while. Even if you have a larger connection, you might not want to use up all the network bandwidth for data transfer. For smaller amounts of data, so I'm talking about gigabytes or terabytes of data, if you've got a decent network connection, there are a few tools available. Of course, you can use the Azure portal to upload data to Azure Storage like you saw it earlier, but there is also Azure Storage Explorer, which offers a graphical user interface and makes moving data as easy as using File Explorer, so it's a good choice if you need to delegate some tasks to business users, but it also offers management capabilities for a storage account so it can be used by administrators too. Behind the scenes, Azure Storage Explorer uses a tool called AzCopy. It's actually a command line tool that you can use directly. AzCopy provides features for uploading in a very performant way by leveraging multiple connections at once, and you can integrate AzCopy commands into your scripting activities to copy data to and from Azure. PowerShell and the Azure command line interface also have commands for managing data in Azure. For developers, there are also the client SDKs available in a variety of languages, so you can integrate data ingestion to Azure Storage as part of application development. Earlier, I mentioned Azure File Sync as a way to extend your file shares to Azure. This way, data that you access frequently is kept and data that you use less often is automatically stored in Azure. Azure File Sync is always online, and you don't need to manually move the data to Azure. These are some of the ways that you can get data into Azure using a connection over the internet. Sometimes that's not convenient or even possible due to the amount of data or the throughput of your internet connection. So we'll talk about some options for offline data transfer later, but first, let's take a look at Azure Storage Explorer and AzCopy.

Chapter 45 Azure Storage Explorer

Now let's talk about a tool for managing the data in your Azure Storage account. Azure Storage Explorer is a tool that you can download for Windows, Mac, and Linux and it runs on your local desktop. I've already installed it on my local computer, so let's search for it and open it up.

We need to log into Azure, so let's add an account. There are a number of ways to authenticate. You can log into the whole Azure subscription so the administrator would likely do this, or you can scope the login to just the resource the user needs access to like a blob container or a file share. This is great if you want to give access to an end user to upload files to Azure Storage. You don't want them to have any more access than they need so you set the permissions in Azure Storage and have them log into the resource that they need access to.

Let's use the subscription option and hit Next. That opens a browser and I'll select my administrator account, enter my password, and I have multifactor authentication enabled so I need to send a code to my phone and then enter that. Now I'm logged in and my account shows here along with the subscriptions that I have access to.

I'll go to the Explorer tab and expand the subscription. All the storage accounts are listed and below that are disks. This is a big advantage to using Storage Explorer. You can manage the disks used by virtual machines in your subscription, and I'll show you that shortly. First, let's expand the storage account we created earlier. If I on that storage account, I have some options to manage it like copying the access keys and setting the default blob access tier. Let's expand the Blob service and click on this container.

The blobs show on the left, and by on the container, I can do things like copy the container, get a shared access signature to provide someone else access to the container, and set the public access level.

You can manage the blobs in the container from here also. you can change the blob access tier for the individual blob. You can acquire a lease, create a snapshot, and edit tags.

Of course, you can upload blobs from here also with the same options you saw in the portal like setting the access tier. Let's cancel out and look at file shares. From here, you can create file shares and upload and download data, and you can get the script to attach this file share to a Windows VM, but let's close this and look at the Tables service. There are some tables here because classic metrics are enabled on this storage account. In the second table here, there is some data, and you can run queries against this data from here if you like.

With the Queues service, you can create a new queue, so this might be for applications or components that need to pass messages. And you can create queue messages from here also, which you might want to do for testing during development.

Let's close this storage account and now let's look at disks. I have this resource group that's storing the operating system disk for the VM that I created earlier. From here, you can download the disk image and also create a snapshot, and you can also upload disk images from to Azure.

You select the operating system type, you can select whether it should be stored on a drive or a hard drive, and the generation of the image.

Azure Storage Explorer gives you a graphical user interface that lets you do some management of your Azure Storage account, and it's great for transferring data to and from Azure. It also lets you delegate access to other people with an interface. Azure Storage Explorer actually uses a utility called AzCopy to transfer the files, and you can use AzCopy directly too, so let's look at that next.

Chapter 46 How to Use AzCopy to Upload & Manage Blobs

Let's take a look at the AzCopy command line utility. You can download it from docs.microsoft.com, and it's available for Windows, Mac, and Linux. I've already downloaded it, and you just need to unzip it and navigate to the folder where it's stored, it doesn't get installed. I've copied it to the root of my C drive. So I'll open up the Windows command prompt and I'll change directories to the root of the C drive where the AzCopy file is stored. We need to authenticate to Azure, so I'll type azcopy login.

It says I need to open up a browser and go to microsoft.com/devicelogin and paste in this code, so let's do that. And I'll paste in the code I copied and select my administrator account and just click Continue to log in. Now we can run some commands, but before we do that, I need to show you something in the Azure portal. I'll navigate into the storage account that we're going to be working against with AzCopy. Let's go to Access Control and view my access. I had to give my administrator account this permission called Storage Blob Data Contributor.

That will give the account access to create and modify blobs in the storage account. You don't actually have to log in using Azure AD. You can just attach a shared access signature on all of your calls with AzCopy, but using the Azure AD login is just more convenient. Let's open up the Containers tab and now go back to the command prompt and run some commands. Azcopy make will create the container name at the end of the URL to the Blob service in my storage account.

It says it's being created, so let's go to the portal and refresh the list, and there is the new container. Let's open it up. And now let's upload some files back in the command prompt. I have these files on my local computer. Let's copy the DOCX file. You do that with the azcopy command specifying the source and destination.

The source can be a local file or folder or even a container in another storage account, and you can change the name of the file in the destination, so it's the same file just with a different name. It shows that it was successful, so let's go to the portal and refresh the container, and there is the file with the new name. The last thing I want to do is to use AzCopy to change the access tier of one of these files. I want to change it from hot to cool because I know I won't need to access this file, and I'd like to save on storage costs. So back at the command prompt, I'll use azcopy with a path to the file and the parameter set to cool.

No failures, so let's take a look in the portal. And the blob access tier has been changed.

AzCopy is a powerful tool that you can use for managing files in Azure and you can use when creating scripts for batch jobs. Next, let's talk about managed database products in Azure.

Chapter 47 Managed Database Products in Azure

Azure offers managed database solutions for storing structured data in relational databases. Let's start by talking about Microsoft's own relational database management system, SQL Server. There are three offerings for SQL Server in Azure that make up the SQL Server family of products. You can host SQL Server on virtual machines, which gives you full control over the product and all the features you're accustomed to hosting SQL Server in your own data center, but you can also provision a virtual machine with SQL Server already installed by using the Azure Marketplace, and you can take advantage of pricing so you don't have the costly upfront licensing fees. You even have the ability to configure a maintenance window for some automated patching, and you can configure backups using a managed backup service in Azure. Then there is a version of SQL Server called Azure SQL Database. Most database management functions are handled for you like upgrading, patching, backups, and monitoring. Azure SQL Database is always running the latest stable version of SQL Server with high availability guarantees. There is also flexible pricing models based on either the number of virtual cores or using a unit of measurement called DTUs, which stands for database transaction units, and makes up a combination of CPU, memory, and data throughput. Azure SQL database also has flexible deployment options. You can provision a single isolated database or what's called an elastic pool, which is a collection of databases with a shared set of resources. With single databases, you can still harness the elasticity of the cloud by scaling database resources up and down when needed. There are different service tiers available too like the standard tier for common workloads, the business critical premium tier for applications with high transaction rates, and the hyperscale tier for very large transactional databases. Running SQL Server on a virtual machine, you get access to all the features of the product so there are some limitations to using Azure SQL Database. The majority of core features are available in the managed version, but if you have some specific requirements, you can verify compatibility with Azure SQL Database in the Microsoft documentation. If you have compatibility concerns, there is also a third offering called Azure SQL Managed Instance. It combines the broadest set of SQL Server capabilities with the benefits of a platform. It allows you to deploy a managed VM with SQL Server onto your own virtual network. Some organizations have security concerns about deploying databases onto a managed public cloud platform, so SQL Server Managed Instance lets you lift and shift your databases to the cloud with minimal changes and into an isolated environment with the networking controls while also getting the advantages of automatic patching and version updates, automated backups, and high availability. Those are some of the options for using SQL Server in Azure, but there are other database options available in Azure. Using the Azure Marketplace, you can provision a variety of virtual machines with various relational database systems preinstalled, but of course, you'll be managing those servers and databases yourself. In terms of offerings, Azure offers Azure Database for my SQL and Azure Database for PostgreSQL. These are hosted versions with pricing, and you get high availability and automated patching of the underlying database engine. Next, let's talk about how you can migrate different types of workloads, including SQL Server from your environment into Azure.

Azure Migrate is a unified migration platform that allows you to start, run, and track your migrations to Azure. It lets you assess your infrastructure, data, and platforms to determine how to migrate them to Azure. Azure Migrate can assess your physical and virtual servers for migrating them to Azure virtual machines. It can assess SQL Server instances to migrate them to SQL running on a VM, an Azure SQL Database, or an Azure SQL Managed Instance. It can assess web applications running and migrate them to run on Azure App service or the Azure Kubernetes service. For migrating large amounts of unstructured data offline, Databox is a service that's part of Azure Migrate. You can track your data migrations in the Azure Migrate portal, and you can also assess your virtual desktop infrastructure and migrate it to Azure Virtual Desktop. Azure Migrate is made up of several tools, let's look at some of them. Let's start with migrating servers. The Azure Migrate: Discovery and Assessment tool looks at physical servers hosted and virtual machines running on and VMware. It assesses whether the VMs are ready for migration to Azure, including the web apps and SQL servers running on the VMs. The tool estimates the size of the virtual machine that will be needed in Azure and estimates the costs for running the servers that you need. The tool also identifies dependencies and makes recommendations for optimization. You download a virtual appliance to your environment to do the assessment, and it can run on a physical or virtual server. It discovers servers and sends metadata and performance data to Azure Migrate. Then you can use the Azure Migrate Server Migration tool to actually replicate your servers into Azure. You can also migrate servers hosted in other cloud environments. You download and install a replication appliance in your environment and install the mobility service agent on the servers you want to replicate to Azure, Then you can replicate up to 10 servers at once and track the progress in the Azure Migrate portal. The server migration tool also attracts incremental changes to the disks after the initial replication and makes updates to the disks in Azure. For migrating SQL Server databases, there is the Data Migration Assistant and Azure Data Migration service. The migration assistant detects compatibility issues that can impact functionality when migrating to newer versions of SQL Server in the cloud or to Azure SQL Database. It can also recommend performance and reliability improvements. For smaller databases, the migration assistant can move the schema data and objects from your source server to your target database in Azure. The Azure Database Migration service is for large migrations in terms of the number of databases and the size of those databases. Both of the tools can move your SQL Server databases to SQL Server hosted on virtual machines, Azure SQL Database, or Azure SQL Managed Instances. Azure Migrate also has tools to assess and migrate.NET and Java web applications hosted They can be moved to Azure App Service as containers or as code. You can also use the Azure Migrate App Containerization tool to containerize existing ASP.NET and Java applications and move them to the Azure Kubernetes service. A Docker file gets created and the container is pushed to the Azure Container Registry for deployment to Kubernetes. The process doesn't even require access to your code base. So next, let's take a look at Azure Migrate in the Azure portal.

I'll go to All services and search for migrate. This isn't a service that you create, it's a portal inside the Azure portal to organize all your migration projects.

The different migration options are listed on the left menu. Let's go to Servers. I've already created a project, so the instructions for assessment and migration are listed here. Let's click on Discover.

It asks if the servers we want to migrate are virtual or physical, and notice that physical includes servers hosted on other cloud platforms like Amazon or Google. Let's select The next screen is where you can download the appliance to your environment that will do the discovery and assessment.

We won't do this. Let's close out of this and let's see what's involved in creating a new project. There is only a few things you need to enter here, the resource group, give the project a name, and you need to select a geography.

Notice this isn't an Azure region. You learned earlier that geography is organized groups of Azure regions. Under Advanced, you can select whether the migration will be over the public internet or a private endpoint. That's all you need to create a project. Let's close out of this though and let's look at another type of project for migrating web apps. I'll create a new project here, and this looks exactly like the servers project, but let's go ahead and create this. I'll use an existing resource group and enter a project name. Now I'll select my Azure geography, and I'll leave the public endpoint.

The project gets created and the steps here are different than for the server project. It says to download the App Service Migration Assistant.

Let's click this link. That brings us to the overview page for the tool. Scroll down and the tool is available here.

Let's download this. I just have to accept the license terms and an MSI gets downloaded to my local computer. Clicking on the file actually runs the installer, although there is no wizard here to step through. So when this is done, I'll minimize this window and there has been a shortcut installed on my desktop. I'll click this to open up the tool. So I have a web server installed on this Windows 10 computer, Internet Information Services, or IIS, is running the same web server that you would install on a production server in your environment. So the tool detects that and scans the web server. There is only the default site running which is just a web page actually, but it did discover the site so let's proceed with this.

Once the assessment report is complete, it shows all the things that were scanned and it passed all the checks. Next, I'm told to log into Azure. I'll click this and paste in the code that was copied to the clipboard. I need to choose an account, and I'll skip filming all the login stuff because I have multifactor authentication enabled. And once I'm logged in, it asks if I'm trying to sign into the App Service Migration Assistant.

So let's continue and go back to the assistant. Now we can select a project in Azure Migrate. I have several. Let's choose the one I just created. Next, we enter the target in Azure. I'll choose an existing resource group and give this app service a name, which remember, needs to be unique across Azure.

I'll choose an existing app service plan and that's it, let's hit Migrate. And while this is creating the app service in Azure and migrating the code, you can export the Azure Resource Manager template for this app service configuration.

Once the migration is complete, you can navigate to the website deployed in Azure.

This is just a default web page that gets installed locally with IIS, but it shows that the files were successfully copied into Azure App service. So Azure Migrate makes it pretty easy to move resources from and other cloud providers into Azure. Next, let's talk about Azure Data Box for moving large amounts of data into Azure.

Chapter 50 How to Migrate Data with Azure Data Box

Azure Data Box is a service where Microsoft will ship you a secure device that you copy data onto and send back to Microsoft to load into at Azure. Data Box is actually part of Azure Migrate. So let's go to the Azure Migrate portal and go down to Data Box on the menu.

You select a resource group for the order, then a source country or region, and a destination. Click Apply and the options are here.

There are three tiers to the service depending on how much data you need to transfer. The standard Data Box service is a secure device that can move up to 80 TB of data. It gets transported to you and you copy your data onto the device and ship it back to Microsoft. Or you can use it to export your data from Azure and copy onto your local network. You might use this to move some virtual machines or databases to Azure in a migration. Keep in mind, this isn't just a hard drive that Microsoft is shipping you. These are pictures of the device from the documentation.

It's a rugged device with fast transfer speeds and multiple layers of security and encryption. You can track the status of the transfer right in the Azure portal. For smaller transfers, Azure Data Box Disk can be used. Microsoft can ship you 1 to 5 disk drives that can hold up to 35 TB of data. The process is the same, and one of the advantages is that it can use USB 3.0 to copy the data locally, so you don't need a network interface like you do with the standard Data Box. You might use this to send backups to Azure Backup without having to transfer them over the network or to seed files in Azure File Sync. On the other end of the spectrum, if you need to transfer more data than the standard Data Box service, you can go with Azure Data Box Heavy. This is a device with 800 TB storage capacity, and it uses a 40 Gbps network interface. You might use this to move your virtual machine farm or SQL Servers to Azure in a lift and shift type scenario. There is another service available for Microsoft for drive shipping also called the Import/Export service. This allows you to ship your own disk drives to Microsoft to load the data into Azure Blob or File storage. You can also ship disks to Microsoft and have them load your blob data onto the disks to be shipped back to you. There is another product with Azure Data Box and it moves data online so it's not storage that you ship back and forth to Azure. Azure Data Box Gateway is a virtual device hosted in your environment. You write data to the device like using a file share and then as the data is written to the gateway device, the device uploads the data to Azure Storage. You might still use Data Box for the initial offline transfer and then use Data Box Gateway for incremental ongoing transfers over the network. In summary, you learned about storage options in Azure. We went into depth on storage accounts and their configuration options. We created a storage account and explored blobs and files. You learned about data migration options like using Azure Storage Explorer to upload small amounts of data and Azure Data Box for transferring large amounts of data offline. You also learned about using Azure

Migrate to move different types of workloads to Azure. Next, we'll look at ways to manage Azure other than using the Azure portal, as well as monitoring tools in Azure to ensure the health and performance of your resources.

So far, we've been creating resources in Azure using the Azure portal. That's a pretty intuitive way to do it, but there are also several tools for managing Azure from the command line and using infrastructure as code. Central to all of Azure Management is Azure Resource Manager, which goes by the acronym ARM. ARM is the deployment and management service for Azure and it's central to all the creation, deletion, and modification of resources that you do in Azure. When you're using the Azure portal, you're really just using a website that sends requests to the ARM endpoint. ARM handles authentication using Azure Active Directory and authorizes that you can perform the action that you're attempting to perform. ARM then sends the request to the Azure Service that you're attempting to create or manipulate. That could be an app service, a virtual machine, an Azure SQL Database, a machine learning workspace, anything in Azure that's a resource, which is basically everything in Azure. ARM is used by all the tools that you use to manage Azure. The Azure portal is an obvious tool, but you can also use PowerShell to create and manage resources in Azure. It's actually done through a set of cmdlets that you install as the Azure PowerShell module. PowerShell works from a Windows, macOS, or Linux computer, and with PowerShell, you can write scripts to automate a series of tasks, so it's really powerful. There is also the Azure interface, or Azure CLI. The Azure CLI is a set of commands used to create and manage Azure resources and it's also available for Windows, macOS, and Linux. It runs in the Windows command prompt on Windows or the Bash Shell on Linux. You can download and install PowerShell or the Azure CLI onto your local workstation, but there is also something called the Cloud Shell in the Azure portal that lets you use these scripting tools right from within the portal in your browser, so you don't need anything installed locally. There is a mobile app for managing Azure that lets you use a graphical user interface on your phone to create and manage Azure resources and receive alerts. There are also SDKs for different programming languages that allow you to call the Azure Resource Manager endpoints so you can build Azure management into a custom solution. Azure Resource Manager was introduced in 2014. Before that, there was the classic deployment model where every resource existed independently. You couldn't group resources together. The concept of resource groups was a major addition that came with ARM. Azure Resource Manager also brought the ability to use Resource Manager templates, which allow you to define your infrastructure using JavaScript Object Notation, or JSON. That lets you deploy infrastructure as code to create the resources for your solutions. And the Azure Resource Manager model also brought the concept of tags which allows you to logically group the resources in your subscription. We're going to look at different ways to manage Azure using the Azure CLI, Azure PowerShell, and we'll see those in the Azure Cloud Shell also. Then you'll learn about Resource Manager templates for deploying infrastructure in a repeatable way. Then I want to show you some of the services in Azure that can help with monitoring and troubleshooting your deployed solutions. Azure Service Health gives you a view of the health of the overall Azure service, so you'll know if there are problems with the platform that could be impacting your applications. Azure Monitor integrates with all the Azure services to provide monitoring of different metrics with alerts you can set up to know when there is an issue. Azure Monitor also contains log analytics which provides extensive logging capabilities that can help with troubleshooting. You'll learn about Microsoft Defender for Cloud which assesses the security of your cloud workloads, provides recommendations, and alerts you to security events. After that, we'll discuss Azure Advisor which provides recommendations for configuration of your deployed resources. Next, you'll see the Azure mobile app, which is a native app for your phone to manage your Azure subscription and receive alerts. Finally, we'll talk about Azure Arc, which lets you monitor resources that are deployed outside of Azure. We've got a lot to get through, so let's start by looking at the Azure CLI next.

Chapter 52 Azure Command Line Interface

The Azure CLI lets you manage Azure resources from the command line. You can download it to your local workstation, and it's available for Windows, Mac, and Linux, and I'll show you later that it's also available right in the Azure portal using the Cloud Shell. All Azure CLI commands start with az, then the command. Before you can use the CLI with your Azure subscription, you need to log in. That's done with the az login command. The first command I'll run is to get the list of resource groups in this subscription. Azure CLI commands are organized into groups and subgroups. Az is actually the parent group, and group is the name of the set of commands for resource groups, then list is the actual command. I'll hit Enter to run this.

That gives us back a list of the resource groups and their properties, and this is showing a JavaScript Object Notation format, or JSON. Let's run this again, but this time, we'll use a global argument called output, and this lets you format the output of any query. You can modify this to include whatever properties you want, but let's move on. Another useful global argument is help, this will give you information at whatever level you use it. So by typing help after the subgroup name called group, we get a list of all the commands that are available in that subgroup.

There is the list command we used. Let's try using the help argument at the root, the az group. That gives us a list of all the subgroups which have commands for the different services in Azure.

There is the appservice subgroup, the Cosmos DB subgroup, the group subgroup that we've been using for resource groups, and there is a subgroup called resource for managing resources. Let's use that one. And we'll list out the resources in a resource group. Let's use the second resource group here. I'll use az resource list, then we need to use some parameters. is the name of the group we want it to list the contents of, and let's format the output as a table again. So there is just two resources in this resource group, an app service plan, which is actually called a serverFarm type behind the scenes in Azure, and a site, which is an appservice.

Now let's create some resources. First, I'll create a resource group using az group create, then the location parameter, and I'll use canadacentral, and the name of this resource group will be cli_rg. The JSON that's returned indicates that it was created successfully.

Otherwise, there would be an error showing. Now let's create an appservice, but first we need an app service plan, so I'll use az appservice plan create then the resource group I want it created in, the name I want to call the appservice plan, and we need to provide a sku, which is the code for the pricing tier. I'll use the standard S1 pricing tier, the same one we used when we created an appservice plan earlier in the portal. It'll take a second to provision this, but then the JSON returns to indicate that it worked.

Now let's create an appservice for the plan. This is actually a different subgroup. We use the create command in the webapp subgroup passing the resource group name, the appservice plan name, and the name of this app service.

It looks like it was created, so let's go to the Azure portal and check. I'll go to Resource groups, and there is the resource group we created using the CLI. It only shows the appservice plan, but sometimes there is just a delay in it showing up in the portal, so I'll hit Refresh, and there is the app service.

Let's click the Browse button and make sure it's working with the default page.

That's how to use the Azure CLI to query and manage resources in Azure. Next, let's look at Azure PowerShell.

Chapter 53 Azure PowerShell

Another way to manage Azure resources is using Azure PowerShell, which is a module for PowerShell that you can install. PowerShell runs on Windows, Mac, and Linux, and Azure PowerShell requires PowerShell version 7 or higher, so you might have to install that first, which is what I had to do on my Windows 10 computer. Then you can install the Azure PowerShell module from right within PowerShell using the install module command. Let's open up PowerShell 7 which runs alongside previous versions of PowerShell. Just like with the Azure CLI, the first thing you need to do is authenticate to Azure. In PowerShell, that's done with connect az account. A browser opens up just like with the CLI allowing us to enter our credentials. I'll use the administrator account in my Azure Active Directory tenant, and I'm already logged in so I don't have to enter the password. Back in PowerShell, it shows that I'm authenticated. Now let's run some commands against Azure. First, let's list the resource groups in the subscription. That's done with with no parameters.

PowerShell commands always start with the action verb so get, in this case. That returns the list, but we can format this in PowerShell too. To do that, you send the output of the first command into another PowerShell command using the pipe operator. So we'll use the command with the AutoSize parameter. That's easier to read.

Let's clear this, and now let's list the contents of the resource group we created using the CLI. That's with the ResourceGroupName parameter, and we'll pipe this to So there is the app service and app service plan we created.

Now let's add a storage account to this resource group. First, I'll create a variable to hold the name of the region where I want the storage account created. You can start to see how PowerShell can be used for scripting. Now let's call passing in the name of the resource group, the name we want to give the storage account, then for the location, we'll use the variable we created. Next is the sku name. I'll use the locally redundant storage option, and finally, the kind of storage account which will be StorageV2.

It shows that the provisioning succeeded, so let's go to the Azure portal and take a look. I'll go to resource groups again and I'll open up the cli_rg, and there are the app service resources.

So using Azure PowerShell or the Azure CLI, you can use commands to manage Azure and also integrate those commands into scripts for more complex operations and deployments, but you do need to install the tools locally. If you're working remotely or don't have permissions to install applications on your local computer, there is an easier way to use the Azure CLI and PowerShell right in the Azure portal using your browser. Let's look at that next.

Chapter 54 How to Use Azure Cloud Shell in Azure Portal

You can run Azure CLI and PowerShell commands right in the Azure portal using the Cloud Shell. Cloud Shell runs on a temporary host container in the background and it requires an Azure file share in an Azure storage account. You only need to create this once, and it will get mounted each time you use the Cloud Shell, so you can persist files that you upload between sessions. You choose either the Bash shell or PowerShell here, but you can change between them any time after this gets created. You can change the options for the storage account creation if you like, the region and resource names, but I'll just leave the defaults.

Let's create the storage account for the Cloud Shell. Once the storage account is created, it'll connect the terminal. You can resize the window, and let's make the text a little bigger from the Settings menu. Let's clear this, and since we're in the Bash shell, it's the clear command. Because I'm already logged into the Azure portal, I don't need to authenticate like I did with the Azure CLI installed on my local computer, so let's run a command against this subscription. I'll just run the az group list command.

Now let's look at the menu across the top. If you have problems starting the Cloud Shell, like if it hangs during connection, you can restart it from here. This can come in handy. You can upload and download files. You might have scripts you want to run here or you might upload a file with variables that you want to use when running commands. Remember, there is a file share attached, so those files are only accessible to you. You can actually see the files and edit them right here too. If I expand this and click on a file, it opens in the editor on the right, and I can modify the contents, but let's hide this and now let's switch over to the PowerShell version of the Cloud Shell.

I'll just confirm, and the terminal connects again. I'm already authenticated in the portal, so I can run PowerShell commands here.

Before we leave the Cloud Shell, I just want to show you that besides accessing it here in the Azure portal, you can also go to shell.azure.com, and that opens a full screen version of the Azure Cloud Shell. Depending on your device format, that might be easier to use.

Next, let's talk about Azure Resource Manager templates.

Now let's talk about using Azure Resource Manager templates to deploy resources in a repeatable way. Many development teams are adopting agile methods and quick iterations where they want to deploy repeatedly and know that their infrastructure is in a reliable state. That's a big part of DevOps where the traditional division between developers and IT operations roles has disappeared. Teams are now managing infrastructure using code, so those definitions can be stored in code repositories alongside the source code, and they can be deployed in repeatable ways, sometimes using the same continuous integration continuous deployment process that's used to deploy web applications and database code. To implement infrastructure as code, Azure has Resource Manager templates. These are files written using JavaScript Object Notation, or JSON, and the contents define the infrastructure and configuration for all the Azure resources in your solution. It uses a declarative syntax, which means you state what you intend to deploy without having to write a series of programming commands to create it. Once you write the code in the template, you can deploy it in a variety of ways. In the Azure DevOps service, Azure Pipelines allow you to automate code deployments to hosting environments and you can deploy Resource Manager templates as part of an Azure Pipeline too. You can also deploy templates from within GitHub using GitHub Actions, which is a service that's similar to Azure Pipelines. It's also possible to deploy templates using PowerShell or the Azure

CLI, and you can also deploy templates using the Azure portal. Let's take a look at how to do that. I'll open up the list of resource groups and drill into the one we created using the Azure CLI. There are three resources in this resource group, an app service, an app service plan, and a storage account.

Every resource in Azure is defined by an ARM template. Let's look at this storage account. If we go down to the export template tab at the bottom, Azure will generate an ARM template based on the current configuration.

Parameters are broken out by default, so it's easy to change the name of the storage account when you deploy a new one using this template, but you can turn off that feature and have the name generated inline in the JSON.

Often, you'll want to deploy groups of resources though, so let's go back to the resource group and there is a tab on the menu here too that will generate an ARM template with all the resources in this resource group, but it's also possible to select just the resources you want and export the template from the menu here at the top. Now we've got just the app service plan, which is called serverFarms in the JSON, and the app service whose resource type is actually called microsoft.web/sites. From this screen, you can download the template and that creates a zip file with the template and a file with the parameters.

If I the template file, it opens in Visual Studio Code, which is the default editor on my local computer. So you could use this as a starting place and modify it to configure or add resources.

Let's go back to the browser though and let's see how we can deploy the template from here. Now when you're generating a template like this from the existing state of deployed resources, you often need to massage it a bit.

Besides changing the names of resources to deploy new instances, sometimes there are things included that actually can't get deployed. Let's edit this template. I'll scroll down to the bottom, and there are two entries for snapshots.

The documentation says that these are which means they can't be deployed, so let's remove them and let's save these changes. Okay, now let's create a new resource group to deploy this template to. The parameters from the template are showing here, so let's change the name of the app service and the app service plan.

That's all we need to do. The validation passed, so let's create this.

It'll take about a minute to deploy the resources in the template.

And it says the deployment is complete. Let's go to the Resource groups tab and there is the new resource group. There is an app service and an app service plan. Let's drill into the app service, and I'll just click browse to make sure it's running.

So that's how you can use Azure Resource Manager templates to deploy resources in a repeatable way. Next, let's talk about monitoring the health of the Azure platform.

Chapter 56 Azure Service Health

Azure Service Health keeps you informed about the health of your cloud resources. This includes information about current and upcoming issues that might impact your deployed resources like outages and planned maintenance. There is actually three services that make up service health. Azure status gives you information on service outages across all of Azure, Service Health is a personalized view of the services and regions that you're actually using, and Resource health provides information on your specific resources. Let's look at these. Azure status isn't part of the Azure portal. You go to azure.status.microsoft. It gives you an overview of all the Azure services and their current status. So this is a view of Azure.

The regions are organized into groupings with each region and columns and Azure services and rows, it's quite a long list of services. Let's go to the Azure portal and look at Service Health, which is the recommended way to check the health of your resources. I'll search for it under All services.

So Service Health scopes the affected services to just the ones that you use, so you might not be impacted by an outage in Azure Front Door, for example, if you're not using that service. Azure Service Health will trim those notifications to just what matters to you. You can find out about planned maintenance in Azure that might affect you, so you might want to notify clients of an upcoming event or reschedule an application deployment.

There is some planned maintenance here for Azure App Service in regions where I have app services deployed. Notice the impact category says that there is no impact expected.

Health advisories are changes in Azure services that require your attention. For example, if features in a service that you use are being deprecated or you need to upgrade your web applications because the framework version in Azure App Service is being updated.

There is also a link to Health history here which has one of the same links from planned maintenance, but also this historical entry about a DNS failure with Azure Monitor. Security Advisories are notifications or violations that might affect the availability of your Azure services.

The Resource health tab lets you scope to just certain resource types in your subscriptions.

The first two app services are grayed out because they're on a free pricing tier where resource health isn't available, but these other two are on the standard pricing tier, so you can get a quick summary of the overall health and drill in to see more.

If there were issues here, there would also be information on actions that Microsoft is taking to fix the problems and it would identify things that you can do to address them.

You can see a history of the health of the resource if you need to do some historical troubleshooting, and you can add a health alert from here also. Let's back out and let's add a service health alert. You can use this to be notified when there are any changes to a particular service, you can filter the alerts to just service issues or health advisories, security alerts, or planned maintenance, and you can filter the services and regions that you want to be notified about.

The alerts get sent to an action group. Let's open this up and create a new action group. Let's just move ahead to the Notifications tab.

You can just have an email sent to the people in the Resource Manager role, or you can set up a custom notification for email, text message, and automated voice message, and there is an option here for Azure app push notifications.

You can get notified on your mobile device through the Azure app too. So Azure Service Health can make you aware of when there is an issue with the underlying platform that can prevent you from chasing down a problem with your application when it isn't your application at all. But sometimes the problem isn't with the entire service, it's with your resource. So let's look at how to monitor resources next.

Chapter 57 How to use Azure Monitor

Azure Monitor is a service in Azure that collects metrics and logs from the Azure resources in your subscription. You can use these to check on the performance and availability of your applications and services. Metrics are numerical values that describe some aspect of a system at a particular point in time, and they're constantly being collected. This could be things like the response time of a web application, the amount of CPU being used on a VM, the amount of data coming out of a storage account. Metrics are good for alerting and fast detection of issues. The tool in Azure Monitor that helps you explore the collected metrics is called Metrics Explorer, and it's available inside each Azure resource. Logs, on the other hand, are different kinds of data that are organized into records with different properties for each type of log entry. Logs are good for troubleshooting issues and for analyzing trends. Azure Monitor includes a tool called Log Analytics, which is used to edit and run queries on the log data. It uses a powerful query language called the Kusto Query Language that's kind of like SQL, and it lets you sort, filter, and visualize the data in charts. Another service that's part of Azure Monitor is Application Insights. This monitors the availability, performance, and usage of your web applications. For Azure App Service, you can turn on Application Insights and it will monitor your app from the hosting environment, so things like performance counters on the servers, Docker logs, and you can set up web tests to send requests to your application. You can track API calls and dependencies outside of your application also. For deeper monitoring, you can use the Application Insights SDK to include instrumentation right in your code, and it's available for a number of programming languages. So your application doesn't need to be hosted in Azure to send data to Application Insights from the SDK. Let's take a look at the documentation.

Metrics and logs are collected from different sources inside Azure, and you can collect data from virtual machines outside Azure by installing agents on the machines. Then the data is stored as metrics, logs, and traces, and traces refers to distributed traces. When you have an application with different components deployed to different virtual machines or app services or containers, distributed traces are the data gathered from each of those hosts so it traces a web request through the different tiers of the application and all that data is linked together through some correlation id, which is all possible by using Application Insights. The data that gets collected by Azure Monitor gets used in different ways. Experiences are visualizations and queries that are organized for you already. You'll see shortly in Azure Monitor that these are also called Insights with Application Insights being one of them. You can visualize the data in Azure Monitor using dashboards and a Microsoft service called Power BI. There are tools in the Azure portal to inspect the metrics and log analytics for querying the logs, and you can respond to changes in the metrics by setting up alerts and performing actions like autoscaling an app service to add virtual machines when the load is heavy or calling other services to perform an action like a Logic app.

Chapter 58 How to Explore Azure Monitor

Now let's take a look at Azure Monitor in the Azure portal. Let's look at the central Azure monitor service in this subscription.

On the overview page, there are shortcuts to the same things as on the menu on the left. Metrics are all the metrics collected by the different resources we've deployed. You need to drill into individual resources in order to view the metrics. If I apply this, the metrics are scoped to the ones that are relevant just to this app service, like average response time and the number of requests. Logs are where you can query the logs sent to your Log Analytics workspace.

There are queries here to get you started and they fall into categories like alerts, audit logs for Azure Active Directory. You can actually send metrics to Log Analytics too so they can be queried like other logs. And there are performance queries here related to Azure functions. Let's just pick a query and run it. This shows you the syntax of the Kusto Query Language so you can start to write your own queries.

You can access Service Health from within Azure Monitor too. Insights are referred to as curated visualizations. It's a customized monitoring experience for a particular service. Application Insights is for web applications, and it can collect information from outside the application from the hosting platform, as well as from inside the code of your web application.

The virtual machine insights allows you to monitor the health and performance of your windows and Linux VMs inside Azure, as well as virtual machines hosted or even in other cloud environments. To enable that, you need to install an agent on the virtual machines to send data to Azure Monitor. You can monitor the overall health of your storage accounts. There are some metrics here, and the Capacity tab tells you the amount of storage being used by each service in the storage account. Network insights gives you a view of the health and metrics for all your deployed networking resources.

This is good for checking on application gateways and load balancers, but not all networking resources have health checks. You can also set up connectivity tests to monitor things like latency between VMs and storage and applications hosted in Azure and which also requires installing agents to run the connectivity tests. Let's take a look at setting up alerts for metrics in Azure Monitor. Up at the top of the menu is the Alerts tab. From here, you can create an alert rule. Let's create an alert for an app service.

Next, we set up conditions for the alert. There is a number of metrics to choose from here. Let's select CPU time.

This is the amount of CPU consumed by the app in seconds. We could scope this metric to a particular virtual machine in the underlying app service plan if there is more than one. And we can set the alert logic so when the total CPU time is greater than 80 seconds, then the alert will get fired. Let's finish this part. And next, you decide what happens when the alert is fired. Just like with the service health alerts, you do that with an action group. We don't have any created, so let's create one.

I'll create it in the same resource group and give this action group a name. And next, we set up the notification type. I'll choose the push notification option, and this can be an email, text message, voice message, or we can push out a notification to the Azure app on the mobile device of this user. Let's give this notification a name.

And besides sending a notification, you can also configure an action when the alert is fired. That could be calling an Azure function or a Logic app, calling a webhook in an outside web service.

There is lots of options for taking actions when an alert gets fired. We'll skip tags and let's create this action group. And now let's create the alert rule. Now let's move forward and create this alert.

You can see all the alerts from the Alert Rules tab at the top. Next, let's look at using Azure Monitor metrics and logs from right inside Azure resources.

Chapter 59 How to Use Azure Monitor Metrics in a Resource

Azure Monitor collects metrics for Azure services by default, it's turned on automatically. So let's navigate into a service that we've created like this app service we deployed earlier using an ARM template.

Right on the overview page, there are charts showing things like HTTP server errors, the amount of data moving in and out of the app service, the number of requests, and the average response time. These are coming from Azure Monitor and these are just predefined views of the metrics that are being collected. You can dig deeper by going down the menu to the Monitoring group and clicking Metrics.

Most Azure resources have a Monitoring section on the menu, although the exact tabs might be different depending on the service. We have a blank slate here. Because we're inside a resource already, some of these items are already populated like the scope of the resource and the metric namespace. The same metrics are here that you saw on the charts on the overview page, but there are others too like CPU time for the underlying virtual machines in the app service plan. Let's select that. It shows a slight spike here around 6 PM.

If we scroll further down the list, there are all sorts of metrics around IO, and there is the number of requests. Let's choose that.

It shows 125 total requests. With these charts, you can choose a different aggregation. So instead of summing all the requests, we could show the count at different time periods or the average requests. Let's try a different metric like Private Bytes, which is the working memory set on the server that the app service has allocated.

If this number keeps growing, you might have a memory leak in your application. Depending on the resource, you can apply splitting to the graph. In the case of an app service, I've increased the instance count of the underlying app service plan, so there are two virtual machines, and we can split the graph to show how the metric, Private Bytes in this case, applies to each VM. Let's see what that looks like for a different metric. Let's try Response Time.

It looks like there is quite a difference here, but that's probably because I just added the second instance a short time ago, so it hasn't serviced many requests. Across the top, you can give the chart a name, and you can pin it to the dashboard so it will show up on the main dashboard page along with any other charts and graphs that you want to see every time you log in. That's a quick look at metrics in this app service. Let's look at logs next.

Chapter 60 Log Analytics in Azure Monitor

Now let's look at Logs. I'm still inside the app service. There are a bunch of predefined queries that you can run against the log data that's been collected, but before we can do that, we have to enable that log collection.

You do that from the Diagnostic settings. Let's add a diagnostic setting. Depending on the resource you're in, there are different types of logs available, and you can choose to send metrics also.

We need to give this diagnostic setting a name and then you choose where you want to send the logs. Log Analytics is the obvious choice where you can aggregate all your logs together and run queries. There is a default workspace created automatically, but you can have multiple Log Analytics workspaces. You can send these logs to a storage account and specify how long you want to retain each of the log types for. You can stream the logs to an event hub where they can get ingested by another Azure service or you can send the logs to a tool, maybe you have one in your organization already and you want to keep all the logs together. Let's just send these logs to Log Analytics and save this setting. Now that we have a diagnostic setting, we can go to the Logs tab, which lets us run queries against the data. You can choose a query from here, and these queries fall into the categories on the left menu.

Let's scroll down and choose the query related to response times. This is actually a metric that's being sent to Log Analytics. You can see the Kusto Query Language syntax here, and this is a good way to learn how to write queries by using these ones.

We just turned on logging so there won't be any data yet. I'll go to the default page in the app service, which is already open, and keep refreshing it so there is data generated. It won't show up right away in the logs, but it will come back later when the data is available. There is a chart generated from the results of this query, and the individual data is showing on the Results tab.

So even though this is metric data like we saw in the metrics demo, because it was sent to Log Analytics, there is individual records for each web response here, it's not aggregated like in the Metrics Explorer. Let's try a different query, the one for HTTP response codes. It's showing that all the response codes were in the 200s, which indicates successful HTTP responses.

You can do some formatting of the chart from right here, and across the top, you have some options. You can share this query, you can create an alert based on this query, and you can export the query results to a CSV file, Excel, or to Power BI, and you can also pin the query to the dashboard just like on the Metrics tab. Let's navigate out of here and look at another resource group. This one has a storage account. Let's open that up. And down the menu on the storage account, there is a Monitoring section here too.

Some of the items are different from the app service, but we've got logs and metrics here too. You can scope the metrics to each of the services in the storage account and there are different metrics here that apply to Azure storage, so things like the count of blobs in the containers or the ingress of the data that I uploaded.

On the Logs tab, there are custom queries here related to storage, and you still need to turn on logging from the Diagnostic settings, but here you can set up logging on the individual services in the storage account. That's a quick look at logs. Remember, in the central Azure Monitor service, you can access the logs for all the resources you've configured to send logs to Azure Monitor, then you can write custom queries to correlate data across services. Next, let's look at Azure Advisor.

Chapter 61 How to Optimize Resources using Azure Advisor

Microsoft refers to Azure Advisor as a personalized cloud consultant that helps you follow best practices to optimize your Azure deployments. It's actually a great tool to provide recommendations on how to improve performance, availability, and security of your Azure resources, as well as recommending ways that you can save on costs in Azure. Let's go to All services and search for advisor, and click on here to open up Azure Advisor. It refreshes your recommendations when it loads.

These are personalized recommendations, so Azure is looking at the resources that you have deployed, it's not just providing a list of generic recommendations. This dashboard provides a summary of the recommendations broken down by five categories, cost, security, reliability, operational excellence, and performance. You can click on the tiles at the bottom to get links to each of the recommendations or you can use the menu on the left. Let's look at the security recommendations.

The first one says Accounts with owner permissions should be MFA enabled. So it's telling me to enable multifactor authentication on all the administrator accounts. There is a medium impact recommendation that storage accounts should use a private link connection. Some of these might not make sense for the design of your solution, so you can actually turn these off individually so you don't keep seeing them. Let's go to the next page. Here is one that has a quick fix.

It says web application should only be accessible over HTTPS. Let's click this. On the remediation steps, it says I can just select the app service and click the Fix button.

It will turn on a setting in the app service that only allows incoming traffic over HTTPS so HTTP is disabled. So in some cases, Azure Advisor can make the required changes for you, otherwise, it will just describe what you need to do. Let's go back to the main screen and let's look at the Reliability tab.

There is just one recommendation and it's for the virtual network being used by the virtual machine that I created. It says add NAT gateway to your subnets to go outbound. If I click on the link, it brings me to a screen to create a NAT gateway.

In this case, it isn't fixing the problem for me, but it's making it easier to fix. We won't complete this though. Let's go back and move on to cost recommendations. There actually aren't any, but there is this link to the list of cost recommendations that Azure Advisor uses.

This is a way to be proactive and configure your resources to best save on costs. The first recommendation is for compute and it says to use standard storage for disk snapshots rather than using premium storage. For another service like Azure storage, it has a recommendation about retention policies for log data so you're not storing old data you'll never use. Operational excellence has to do with deployment best practices and things like creating service health alerts. I don't have any recommendations here and performances to help improve the speed of the applications I have deployed. Again, you can view the standard list of recommendations here to get an idea of what Azure Advisor looks for. The last thing I want to show you is on the Overview page.

At the bottom, you can download the recommendations in PDF and CSV format, so you could share this report with other team members who might not have access to the Azure portal. Next, let's step outside of the Azure portal and look at the Azure app, which allows managing Azure from your mobile device.

Let's take a look at the Azure app. This is a tool that lets you monitor the health and status of your Azure resources, quickly diagnose and fix issues, and you can even run commands using the Cloud Shell.

You can download the app from the Apple App Store and from Google Play. I've already installed the app on my iPhone so let's open it up. I'm logged in and I've chosen a subscription. On the home page here, any alerts would show right away. I don't have any so let's scroll down, and I have access to Service Health from here too.

There aren't any service issues, but there is a maintenance notification. It's something about routine maintenance on app services. So I can see the Maintenance window and the impacted regions, and there is more detail further down. It says there is no impact expected.

Back on the home screen, you can create shortcuts to resources that you frequently check on. Let's open up all the resource groups. I'm going to open up a resource group where I know there is a storage account. There is cost management information here showing how much this resource group is costing me.

At the bottom are the resources, so I'll drill into this storage account. I can see some metrics that show me the health of the storage account, and these are coming from Azure Monitor. The Resource health tells me that the storage account is available.

There is some information about the resource, and at the bottom is Access Control. So I can give someone access to the storage account from here, which can be handy if you're out of the office and there is an issue or a new client needs to upload files.

Let's back out of here and go back to the list of resource groups. I'll choose one that has a virtual machine in it. I'll open up this virtual machine. You can see the metrics for the virtual machine and you can restart the VM from here if there is a problem.

I'll actually shut it down so I don't incur compute charges. And you can even connect to this VM. This button will launch another app, Microsoft Remote Desktop. Let's back out of here and go back to the home screen. At the bottom, you can open up the Cloud Shell.

You can choose between the Bash Shell and PowerShell, and from here, you can type in PowerShell and Azure CLI commands to manage your resources. Let's run this az group list command, and we get back information on all the resource groups in the subscription.

We can switch to PowerShell and it will restart the Cloud Shell. The last thing I want to show you is that from the menu at the top left, you can manage your log in and change directories, and you can even access support requests from here.

So the Azure app provides an easy way to perform some Azure management tasks from your mobile device. Next, let's talk about Azure Arc.

Chapter 63 How to Manage Resources Outside Azure using Azure Arc

Azure Arc is a service in Azure that allows you to manage resources outside of Azure. So resources that you host or in other cloud platforms like Amazon Web Services or Google Cloud. You can manage a few different types of resources hosted outside of Azure. You can manage Windows and Linux physical servers and virtual machines, that means being able to monitor them, secure them, and update them from within Azure Arc. When you're hosting your virtual machines on private cloud platforms like VMware vSphere or Azure Stack HCI, you get additional integration with Azure Arc like the ability to perform lifecycle operations like provisioning, restarting, resizing, and deleting virtual machines as if they were hosted in Azure. SQL Server instances hosted outside of Azure can be managed using Azure Arc also, and with Azure Arc, you can manage Kubernetes clusters running and with other cloud providers. Remember, Kubernetes is an orchestration service for containers. You can apply Azure policies to the Kubernetes clusters to enforce configuration and compliance. Once you have Kubernetes clusters being managed, you can run other Azure services on them like data services. SQL managed instances and postgreSQL hyperscale databases are available running on Kubernetes. You can deploy Azure machine learning workloads onto those clusters also. And you can deploy Azure App Services on Azure Kubernetes clusters, including web apps, function apps, and even Logic apps so your developers can leverage the features of app service while you maintain corporate compliance by hosting the app services on internal infrastructure or leveraging your existing investment with other cloud providers.

You get features of Azure Resource Manager when your resources are managed using Azure Arc. That includes organizing resources using management groups and tags, searching and indexing them using Azure Resource Graph, security and access control through access control and subscriptions, automation using templates and extensions, and update management. For physical and virtual machines hosted outside Azure that you want to manage with Azure Arc, you install the Azure connected machine agent on the servers. That lets you proactively monitor the operating system and workloads running on the machine, and you can leverage Azure features like update management to manage operating system updates. You can apply Azure policies to audit settings inside the machine. You can leverage Microsoft Defender for threat detection and Microsoft Sentinel to collect events, and you can collect log data using the Log Analytics agent. The data gets sent to a Log Analytics workspace.

Chapter 64 How to Add Local Server to Azure Arc

Let's look at Azure Arc in the Azure portal. I'll go to All services and search for Arc. When Azure Arc opens, all the infrastructure services that can be hosted are listed on the menu.

The data services are below and the application services are below that. Let's go to Servers. These are the physical and virtual servers that you host or in cloud environments other than Azure. There aren't any being hosted, so let's add one. I'll add a single server, and this will be a virtual machine running on my local computer using

It says the server will need HTTPS access to Azure services for outbound connectivity. It'll need local admin permissions and the server can connect over a public endpoint, so over the internet, or using a private endpoint. We also need an existing resource group to add the server to. I'll click Next, and I've already created a resource group for this which will change the region to Canada Central.

This VM will be using the Windows operating system, but it could use Linux. And connectivity will be over the internet. Next, we can fill out some tags. The default ones define the location of the server. You can add your own custom ones also. On the next page, a script is generated. We need to run this script on the local server in order to download and install the agent and connect the server to Azure Arc, so I'll copy this script. And I have this virtual machine running on my local computer using I'll search for PowerShell and open it up as an administrator. Now I'll paste in the script I copied from the Azure portal, and let's run this script on the local VM. It'll take a few minutes because it's downloading the agent from Azure.

Next it's asking me to sign into Azure by going to microsoft.com/devicelogin and entering the code here. So let's open up a browser and navigate to the URL and it's asking for the code, so I'll switch back to PowerShell to enter it in. Now I need to authenticate. My administrator account has already logged into the browser so I'll use that and it has MFA enabled, so I get a code sent to my phone. I'll enter that in, and now it says Are you trying to sign in to Azure Connected Machine Agent?

So I'll hit Continue, and now we can go back to PowerShell, and it'll continue with the installation.

Once that's done, the machine should be getting managed by Azure Arc. Let's go back to the Azure portal. I'll close out of this and out of the Add server screen, and we're already on the Servers tab so I'll just hit Refresh. There is the server that was added and it says it's connected. There is information here about the operating system and tabs along the left with actions we can perform on this VM. Let's look at Security. Microsoft Defender is running on this virtual machine now, so it's scanning for threats, and there are also recommendations being made related to security.

One of them says that Log Analytics agent should be installed on Azure machines. So Log Analytics doesn't get installed by default, that's another agent we can install on the local VM. We can also manage operating system updates on this VM using Azure automation, and there is something called Automanage that will apply a preset configuration to the VM depending on whether it's being used for dev or production, and that includes things like backup and monitoring.

You can also assign policies to the VM so you can assess it for compliance to rules set up in Azure Policy. There is a lot of functionality here that makes it seem like this virtual machine is running right inside Azure, but of course, it's not, it's running on my local computer. So Azure Arc provides a lot of possibilities for simplifying your resource management across other clouds and Let's have a quick review of what you've learned. We started with understanding how Azure is implemented physically with regions and data centers and logically with subscriptions and resource groups. You saw how to use the Azure portal and learned a bit about Azure Active Directory for controlling access. Next you learned about Azure compute. We looked at virtual machines, containers, and Azure App Services for hosting web apps, as well as Azure Functions for smaller pieces of code. Then you saw some of the main features of networking in Azure, like virtual networks, network security groups, Azure DNS, and private endpoints. You also learned about connecting your network to

Azure using VPN Gateway and ExpressRoute. Then you learned about data storage in Azure with Azure Storage accounts, including how to copy files in Azure and migrate data into the cloud. Then you learned about managing and monitoring Azure using features like the Azure CLI and Resource Manager templates and services like Azure Monitor and Azure Arc. We've covered a lot in Azure, but there are a lot more features and services that are worth checking out like solutions for big data ingestion and analysis, solutions for the Internet of Things, machine learning services, and artificial intelligence. Those things actually used to be part of the Azure Fundamentals exam, but they were removed probably because they're pretty advanced and aren't relevant to as many people as these topics, but I encourage you to jump in and try Azure, create a free trial account, and take some of the services for a drive.

BOOK 2

MICROSOFT AZURE

SECURITY AND PRIVACY CONCEPTS

CLOUD DEPLOYMENT TOOLS AND TECHNIQUES, SECURITY & COMPLIANCE

RICHIE MILLER

Introduction to Azure Identity Services

In the following chapters, we'll be taking a look at Azure identity services. We will first focus on Azure identity services, so we're going to be taking a look at authentication and authorization, we're going discuss Azure Active Directory, and discuss the benefits of authentication. We'll cover Azure AD and discuss why it's important to a secure deployment inside Azure. We'll take a look at access control too. access control gives you granularity of permissions assignment inside Azure. We'll also take a big look Azure governance and policies. We'll look at practical things we can do to make sure that we stay within certain compliance standards, looking at both Azure governance features and documentation to back them up. We'll take a look at securing network access. Almost all the Azure projects I work on have some sort of virtual networking involved in them, and managing access to those virtual network can be a bit challenging. We'll also take a look at reporting and compliance. We'll detail some of common compliance standards your organization might be interested in and tools we can use to achieve them. You might be asking yourself, why is this important to you? Well, if you're thinking about deploying resources to the cloud, a good understanding how they can be securely deployed is important to you. The hat of that is authentication, you must have a good understanding of how your users, computers, and applications are authenticated and authorized to use Azure. Also, monitoring is so important to us today. Understanding how we monitor the security posture of our organization, how we comply to various security standards is vital for all deployments. So if you're interested in any of these areas of Azure, then this book is for you. I would recommend this book to anyone that's interested in securing compliance in Azure, anyone that's thinking about migrating or deploying workloads to Azure and anyone that's studying for exam because this book is part of a series of books that should prepare you well for that certification. There are some prerequisites that would be recommended before starting this book. First of all, a familiarity of cloud concepts, a basic understanding of cloud computing. Even if you don't meet these prerequisites, then as long as you have an interest in Microsoft Azure, then you'll benefit from this book. Let's now talk about authentication and authorization and how the two compare. When we think about authentication, we need to think about the act of proving who or what something is. Authentication is something we do on almost a daily basis in our lives. Each time you show your pass to get into a building, each time you provide a copy of your signature that can be compared with a copy on file, or each time you go through a passport control on your holiday, we're going through a process of authentication. Authentication works hand in hand with authorization. Once you can guarantee beyond a reasonable doubt who or what something is, you can then authorize them to do certain things. So think about the passport analogy. When I travel to some countries, my passport is required to prove who I am, but then a visa is required to show that I'm allowed to enter that country and what I can do when I'm there. The passport is authentication. The visa is authorization. Authorization is saying, now that we can prove who this person is, this is what they can then do. In Azure, authentication is provided by Azure AD and authorization is provided by access control. There may be some overlapping areas, but this is the list I came up with. A user logs in with a password is authentication. This is an example of something the user knows being used to authenticate them. A user uses their thumbprint to get access to a laptop. This is an example of using biometrics. The third example is a bit more general A user proves she's a member of your staff. Well, if they prove they're a member your staff, they've authenticated. The next three are authorization. Once you've been authenticated, you can be given the rights to create virtual machines, get access to files, allowed access to a building. Hopefully, you could see the flow here. Authentication has to take place first, and then authorization can occur.

Fundamental to authentication in Azure is Azure AD. So now, we're going to give you a good grounding in what Azure AD is. Azure Active Directory sits at the heart of authentication for the Microsoft cloud. If you've ever signed up for an Azure subscription, Office 365 or any of the Dynamics products, then you're using Azure AD. When using any of these products, if you've created a user, a group, you've granted permissions, then you've been using Azure Active Directory. Azure Active Directory underpins the security for all these products. When using these products, an Azure Active Directory tenant is created for you. This is dedicated for your company's exclusive use. Usually, an organization will have one Azure Active Directory tenant that manages the security for each of their Microsoft products, so for each Azure subscription or each Office 365 installation that they have. But Azure Active Directory is not just about securing access to Microsoft products, Azure Active Directory can also be used to secure access to your applications, applications, and applications provided by other cloud providers. When we think Azure Active Directory, think single sign on. A user will have a single user account that can be used to access all these different applications. If your company uses Microsoft products, then you're probably already using Active Directory Domain Services Azure AD is not the same product, so let's try and compare the two. Azure AD is all about user and computer registration and providing single sign on capabilities for Users and Computers. Active Directory Domain Services also performs Users and Computers registration. Azure AD does not give access to Group Policies, but Active Directory Domain Services does. Azure AD cannot perform trust relationships, whereas Active Directory Domain Services can. When thinking about Azure AD and single sign on, also think about application management. Applications can register for Azure AD, so your users can be given single sign on access. Active Directory Domain Services offers application and device management, as well as application deployment. Active Directory Domain Services supports both Kerberos and NTLM as authentication protocols. Active Directory Domain Services also gives you access to schema management so that you can add custom objects and attributes into the domain service. Finally, Active Directory Domain Services follows a hierarchical design using domains, trees, forests, and organizational units. It scales almost infinitely, whereas Azure AD is a flat structure, which offers limited scale. For most organizations that already use Microsoft products, when moving to Azure, you'll use a mixture of Active Directory Domain Services and Azure AD in the cloud. But there is a third option as well. In Azure, we have another domain service called Azure AD Domain Services. The names are very similar here, but we have three distinct products. We have Azure AD. This is for single sign on and application integration. We have Active Directory Domains Services. This is the full Active Directory Domain Service that we've used for years And sat between the two, we have Azure AD Domain Services. Azure AD Domain Services was introduced several years ago now. It was initially introduced to make it easier to migrate legacy applications as it supports both NTLM and Kerberos for authentication. But it also supports Group Policies, trust relationships, as well as several over domain service features. Azure AD Domain Services is a Platform as a Service offering provided by Microsoft. Instead of you having to manage the virtual machines, the operating systems, and the directory service, you just deploy Azure AD Domain Services and let Microsoft take care of the rest. The question I get asked most often is can Azure AD Domain Services replace Active Directory Domain Services? The answer right now is still no. It's not as as Active Directory Domain Services, and there's a little way for it to go before it can replace Active Directory. Do we need Azure AD? Well, giving a short answer, yes, it's a requirement. If you're thinking about working with Microsoft Azure, then you will need to work with Azure AD. But that doesn't mean we've wasted all our investment with Active Directory Domain Services. In fact, Azure AD and Active Directory Domain Services work very well together. There's a product called Azure AD Connect that we can deploy Think of Azure AD Connect as being like a synchronization tool. As I perform actions those actions are replicated into the cloud. For example, as I create a new user account that user account is replicated in the cloud. Most of your administration of users, of groups, will still be done We can then feel the benefit of those actions in Azure AD. When we use Azure AD, there are no domain controllers for us to manage, we just access a list of users and groups applications. Azure AD provides us with user management, application integration, and single sign on, and through the use of products like Azure AD Connect, we can integrate with other directory services. Imagine a typical organization. They have different categories of staff. For example, they have a IT staff, end users, as well as contractors who are brought in to work on various projects. What the company would like from us is an understanding of where the different user accounts will be created and managed for these different sets of users. Take a minute and have a think. Where will these user accounts be created for the company? Well, the IT staff can be created or in Azure AD. The majority of the IT staff will be created and their user accounts will then be replicated into Azure so they can be assigned access to perform Azure tasks. Some IT staffed will be That means that their user accounts will just be created in Azure AD. These tend to be administrators who will only need access to Azure and do not need access to services. Our users should have their user accounts created in Microsoft Active Directory Domain Services. Remember, Active Directory Domain Services is almost a limiting scale. It's hierarchical, so the bulk of your user management should be done through there. For the company's users that need access to Azure, well, their user accounts can also then be replicated with Azure AD Connect. Contractors will already have user accounts somewhere, either in an directory service themselves or in an email product like Outlook or Gmail. Using their existing user accounts, we can grant them access to perform admin tasks in Azure subscriptions. As a company, you might have to go through a project where you categorize your staff and discuss how those user accounts can get access to resources we deployed to Azure. When thinking about authentication, Microsoft strongly recommends that we use authentication. authentication involves providing several pieces of information to prove who you are. This information includes something you know, like a password, something you have, like a smart card with digital certificates installed, and some think you are, like biometric information. Microsoft Azure provides several different ways for us to enable authentication for users and to enforce it.

One of the key features of Azure AD is Azure AD single Single means the ability for users to use a single set of credentials to access a whole group of applications and services. This way, your users only have one username and one password to remember no matter which applications they're trying to access. We can integrate Azure AD with applications, applications, and custom applications that we and our teams develop The goal is that users should be asked for their credentials once and not prompted to provide their credentials over and over again. Some applications that we integrate in Azure AD even include the ability to provision users and assign levels of access to those applications directly from the Azure AD console. But controlling access to all of these integrated applications can be difficult. For example, do you want all of your Azure AD integrated applications to be accessed from outside your firewall? There might be some applications that give access to sensitive information that you only want to be used if the user is at one of your corporate centers. Are there times when applications should only be accessed if the user has used authentication to authenticate? Should access to applications be restricted to certain secure devices? There'll be plenty of scenarios when working with integrated applications in Azure where restricted access based on criteria like this will be a good thing. Azure AD Conditional Access can be used to secure access to Azure AD integrate applications based on the criteria previously discussed and more. Azure AD Conditional Access can be used to control access to applications no matter where our users are. We create Conditional Access policies, which at their heart are statements. If the user is authenticated in this way and is using this type of device, then grant access. If not, then deny access. In conditional access policies, signals are used to make decisions. These include IP location information, risk analysis based on the user's login, information about the device the user is using to try and get access to applications, and the application being accessed. Ultimately, the Conditional Access policy is trying to make one of two decisions, either to block access to an application if various conditions in a policy are met or grant access to an application. And even when we grant access, that access can be qualified by enforcing requirements like MFA required or that the user is accessing the application from an Active Directory joined device. Let's take a look at conditional access. In this demonstration, we're going to be working with Azure Active Directory to enable conditional access. We're going to be working with the Azure console and, as always to follow along, you will need the Azure subscription. But be warned, some of the features that we're enabling can incur costs. I'm in the Azure console. Specifically, I'm looking at my Azure AD tenant and a list of its enterprise applications. These are some of the applications that have been integrated with my Azure AD tenant for single

If I select Conditional Access here and then select New policy, we'll start off by giving our new policy a name.

I want this policy to be assigned to a user, so in the Users and group section, I click the blue writing, 0 users and groups selected, and I'm going to choose the Select users and groups radio button and the Users and groups tick box.

Using search, I'm going to search for the user that I want this policy to be assigned to. But don't forget, you can select groups of users, directory roles or guests and external users. Next, I'm going to choose Cloud apps or actions.

Here I can choose the application that I want this policy to affect. So if I choose the Select apps radio button, and for this demonstration I'll just choose Office 365, but you can select any of the applications that you've integrated with Azure AD.

With Office 365 selected, I click Select. We're starting to build up our policy. So far I've identified a user and the Office 365 application. Let's select Conditions. And for this demonstration, we'll choose Device platforms.

Then we'll select Yes to apply this policy to selective device platforms. And we'll say this policy should take effect if the user is using selected device platforms, such as Android, iOS, and macOS.

I'm happy with my selections, so I select Done. Let's select Locations, and here you'll select Yes to control access based on the physical location.

This time we'll leave the default any location selected, but you can build up a list of trusted locations based on IP addresses. I'm happy with the conditions selected, so if I scroll down a little bit, and on the side we have Access controls.

In the Grant section, if I select 0 controls selected, here we get to decide whether this policy will block access or grant access based on the conditions that we've input into this policy.

I want this policy to grant the user access to Office 365, but if the user is using one of the device platforms selected, then I require that he is authenticated using authentication. So if I select that tick box and then choose Select, notice here this policy can be turned On, Off, but the default is

This generates report information that would indicate whether this policy would have taken effect if it was turned on. I want this policy to be enforced, so I'm going to select On. And I'm happy with my policy, so I select Create. It should only take a second, and your policy is created. Now if the user tries to access Office 365 from any of the devices listed, he will be granted access as long as he's logged in with authentication. So far, we introduced our customer to a company. You learned the differences between authentication and authorization, and you were introduced to Azure Active Directory. You were shown how to create users and groups using Azure Active Directory, and you were shown how to enable Conditional Access. Next, we're going to look at Azure access control and Azure locks.

Chapter 3 How to Implement Azure Role Based Access Control

In this chapter we'll be looking at implementing Azure access control. We'll begin by discussing shared access to an Azure subscription and how difficult it is to manage multiple user accounts who have different requirements. We'll then move on and introduce you to Azure access control. We'll take a look at the different types of Azure roles and discuss using custom Azure roles. access control is used daily by your organization. It's central to access control in Azure. Azure provides shared access. By this I mean there are different types of users that require different access to Azure, and we can provide access to them all. Some users will require admin access to Azure while other users will require access to the resources we deploy. Each type of user has to be authenticated and authorized at the correct level, as well as managing each type of user. Each type of user also has to be monitored to make sure they have the correct level of access, but also to make sure they're not trying to breach the access levels they've been assigned. Azure access control is the tool we use to provide shared access. RBAC is made up of several different components, starting off with roles. Roles are groups of permissions that are needed to perform different administrative actions in Azure. We can make users or groups members of different roles that inherit all permissions that are assigned to that role. When using roles, we first choose a role or we create a custom role of our own. We then assign role members before configuring a scope for the role. A scope details where a role can be used. There are many roles, each giving different sets of permissions, but three roles are used more than any other. The Owner role, for example, is used a lot. If you are assigned the Owner role for a resource, you have full control of that resource, including the ability to assign other users and group access. We then have the Contributor role. The Contributor role allows you to do everything except manage permissions, so you would not be able to assign your friend access to the resource with the Contributor role, you would with the Owner role. We then have the Reader role. This role is It lets you view everything, but you can't make changes. These three roles are used most often in Azure, and these roles can be used to grant access at the subscription level, the resource group level, or to individual resources. Here we've got an example of using roles in Azure. We've got a user, and this user needs full control of the development resource group. The user does not need to assign permissions to the resources, he just needs full access for administration. Because the user will be making changes, the Reader role is no good. Because he doesn't want to assign permissions to other users, the Owner role is not required. The Contributor role fits for the user's requirements nicely, so you would assign the user the Contributor role and scope it to the development resource group, giving the user full control of that group. When using roles in Azure, start off by using the roles. There's dozens of these to choose from. Some grant access to Azure resources, some to Azure AD itself, some to applications like Office 365, but if you have a requirement to grant access, start off by looking for a role that meets those requirements. If you can't find the role, then create a custom role. You can use one of the roles as your template. So if you find a role that gives you 90% of what you need, you can copy that role to create your custom role, and then just change the final few percent. Always follow the principle of least privilege. Make users and groups members of roles that allow them to do their job, but no more. A company has different sets of users, like most organizations, and these different sets of users have different requirements. The company will have to go through a process of identifying each of their user's needs and then mapping those needs to the roles that will grant them access to Azure. The company has got three types of users. They have Azure administrators who are quite administrators that need a lot of control in Azure. They have Azure developers who will be working with projects and they will need full control of a subset of resources. And then they have Azure compliance officers. These compliance officers perform audits, making sure the resources that we deploy are compliant with the various standards that the company is trying to certify against. Your organization will go through a similar project to the company, identifying the different types of administrative access required and choosing appropriate roles to make your administration work.

In this chapter we're going to take a look at Implementing Azure Access and Governance Tools. We'll first discuss the importance of governance tools in Azure. We'll then take a look at Azure policies and initiatives before moving on to take a look at Azure Blueprint. We will demonstrate both sets of governance tools, and we'll discuss the importance of governance generally. We'll discuss how we can use Azure governance tools to restrict the sets of features that can be used in your subscriptions and how we can use Azure governance tools to enforce sets of security standards. Before we get into the Azure governance tools themselves, let's just take a minute to discuss why we need the governance tools in the first place. We will discuss with our security teams the different security requirements we need for our cloud deployments. The actual governance tools gives a way to enforce those requirements. We will have also discussed the technical requirements we need for our various deployments, and again our governance tools gives a way of enforcing those technical requirements. So instead of allowing engineers to make their own decisions that might impact security, scale, and cost of our deployment, the governance tools gives a way of putting guardrails in place that our different users in Azure must follow. One often overlooked component of governance and compliance in Azure it Azure tags. And because they are often overlooked, they're worth a special mention now. Azure tags are key value pairs that we assign to Azure resources. Tags might identify the department or project that particular Azure resource belongs to or identify the cost center that should be paying for the resource. All resources that you deploy to Azure should be tagged, but we shouldn't leave it up to the individual to apply any old tag. Instead, organizations should have a tagging policy enforced by Azure policies. This way, the tags applied to our resources in Azure will be consistent. Tags can be used to enforce security requirements, so access to resources will be granted only if certain tags are on the resources, tags can also be used to control costs. We can use Azure Cost Analysis to search resources that have a specific tag, let's say a tag for a particular department or a tag for a particular project, and use that information to build cost reports. We can also put in rules that say that if costs for resources associated with a particular tag go above a particular amount, that we'll be informed and even the automation steps can kick in to close those resources down. We can also use tags when we're deploying software. Our DevOps teams can deploy an application, but on its virtual machines they are tagged in a particular way. Two of the most powerful tools available to us are Azure policies and initiatives. What is Azure Policy? Well, at its heart, Azure Policy is a collection of rules. Each policy we create is assigned to a scope, such as an Azure subscription. By creating a set of rules that the user of that subscription has to then abide by, it will mean the resources they deploy will remain compliant with corporate standards. When using Azure Policy, we create a policy definition, a policy assignment, and policy parameters. When we create Azure policies, they can be used by themselves or they can be used with initiatives. Initiatives are a collection of policies. These policies tend to be grouped together to achieve a larger goal. It is the initiatives then that we assign to a scope, such as a management group, a subscription or resource group. To use initiatives, we create an initiative definition, an initiative assignment, and initiative parameters. To get us started, there are a set of Azure policies that we can use. We have a policy that controls the characteristics of storage accounts, a policy that controls resource types that can be used inside resource groups. One really useful policy is a policy that controls the locations that can be used by your subscription. So if you only want resources deployed to the UK, we can use this policy to make sure that's enforced with a policy that enforces tags and a policy that controls the size of virtual machines to get deployed. So these five, and several more policies, are available to give us a head start when using Azure policies. When working with Azure, think about your teams and think about the resources that they all need to deploy. You'll then create Azure policies so that those resources can be deployed, but nothing else. This will save you money, but also lead to a more secure Azure deployment.

Let's now take a look at Azure Blueprints. Azure Blueprints give us an advanced way of orchestrating the deployment of resources. You may have used ARM templates in the past to deploy virtual machines, virtual networks, entire resource groups. Think of Azure Blueprints as a big extension of what resource templates can already do. One of the benefits of using Blueprints is that they maintain a relationship between themselves and the resources that they deployed. If you use a resource template to deploy resources, changing the template later on will have no effect on the deployed resources. If you deploy your Blueprints on the other hand, a change to the Blueprints can affect the deployed resources. Imagine a situation where you've got set of deployed resources and you want to change the roles that associated with those resources. By changing the roles in the Blueprint, it will update those resources for you. Blueprints can include Azure policies and initiatives, as well as artifacts like Azure roles. These can be deployed along with ARM templates to set up your subscriptions or to deploy a set of resources to existing subscriptions. To use Blueprints, we require a Blueprint definition, we publish the Blueprint, and then assign it to a scope. When we create a Blueprint definition, we can provide details of resource groups that we wish to deploy. We could include the Azure resource manager templates that we want to use as part of that deployment, Azure policies to enforce compliance. We can also use Blueprint definitions to assign roles to the resources that Blueprints have deployed. So far, we've used governance tools that allow us to enforce our security and compliance standards on our subscriptions. Azure also has tools that will look at our subscriptions and provide recommendations to us. One of those tools is Azure Advisor, and part of Azure Advisor is Azure Advisor Security Assistance. Azure Advisor Security Assistance integrates with Security Center. Security Center has lots of information from lots of sources. The job of Azure Advisor Security Assistance then, is to filter through all the information and provide best practice security recommendations. Azure Advisor Security Assistance helps prevent, detect, and respond to security threats. You or your team should be using this tool every day to get the latest security recommendations. Configuration of this tool, the amount of information it is gathering, the type of information it is gathering, is controlled through Security Center. It's the results that we're seeing in Azure Advisor Security Assistance. In this demonstration, we're going to work with Azure blueprints before taking a look at Azure Advisor security assistance. Back in Azure portal, I'm in a Blueprints dashboard. If I scroll down here, we're going to create a new blueprint by clicking Create. Like most governance and compliance tools that Azure gives us, we have a choice of where we start from.

We can start from a blank blueprint, or with a set of samples. Again, if I scroll down, we can see some of the sample blueprints that we can choose from.

In this example, we're not going to use a sample; we are going to start with a blank blueprint. The first part of creating a blueprint is a very familiar wizard. We need to assign a name to the blueprint and provide a location. The location could be a management group or a subscription.

The location you choose dictates where this blueprint can be used. Once we have these basic properties in place, we can click Artifacts. Artifacts control what your blueprint can do.

Let's click Add artifact, and then click the for Artifact type. Here you can see that we have a choice of four different artifacts that this blueprint can work with; Azure policies, Azure roles, Resource Manager templates, and resource groups.

The first thing I want our blueprint to do is to deploy a resource group, so let's select that. We have to select a name for the artifact, and we can either fill in the properties of this artifact now, or say the properties will be filled in when the blueprint is deployed.

That's the default, and that's what I'll leave it as. So I'll say, add here. Notice how we've got two levels for our blueprint now, subscription and resource group.

So let's add a second artifact, but let's add it to our resource group. Notice the type of artifacts have changed. We can no longer select resource group because you cannot have one resource group inside another. What we can select, though, is role assignment, so let's select that.

And from the role let's choose a role. This time, we will untick the box that says this value should be specified when the blueprint is assigned, and in the Add user, app, or group, we'll select the user. With the role assignment properties filled in, we say Add.

I'm happy with this blueprint, so I want to save the changes. So at the bottom here, we say Save Draft. And it might take a minute, but our blueprint will be saved. We can view our blueprint from the Blueprint definitions section.

And we can see our blueprint. Draft blueprints cannot be deployed. Blueprints have to be published first. So let's select our blueprint, and at the top left, we can say Publish blueprint. We'll have to provide a version of this blueprint, and then we click Publish.

It should take no time at all to publish the blueprint. And once we have published it, we can then say Assign blueprint. During the assignment of the blueprint, we can finish off its configuration. So we can choose a location, we can select the version of the blueprint we want to assign, and then as we scroll to the bottom, we can fill in the parameters for the artifacts that we've chosen to deploy.

In our case, a resource group name and location. We don't have the choice of filling the role artifact because we did that when the artifact was added. It'll take a minute to create the blueprint assignment. Once created, we'll see the assignment under Assigned blueprints. Notice the provision section. Right now my blueprint is being deployed. Depending on the size of the blueprint and how much work it's got to do, it could take seconds, minutes, or even hours to deploy your resources. Here we can see my deployment succeeded.

The resource group has been created. If we go inside there and under Access control assignments, we can see that the user has been assigned a contributor role. Back over in Blueprints and Blueprint definitions, we have the option here to edit the blueprint. In artifacts, we'll say add artifacts to the resource group, and for artifact type ,let's choose policy assignment.

And here you can see the Azure policies that we created earlier. So let's choose the require tag Azure policy. Now we've made a change to the blueprint, we say Save Draft, and then we publish the blueprint again, this time with a different version number. Once the draft is published, if we go to Assign blueprint, in the version section we can now see we can assign both version 1.0 and version 1.1.

We've seen how we can create blueprints, publish and assign blueprints, and how we can create a new version of blueprints. One more thing to show here, on the side, you can see the section that says Track assignments.

Let's click Track there. Here we can see our original assignment. If we select that and say Update assignment, through here, we can change the version of the blueprint that's been assigned from version 1.0 to version 1.1. If we scroll down, we can now set a tag for this resource group and say Assign. Here we can see that our change succeeded, and now version 1.1 of our blueprint has been deployed.

For the second part of this demonstration, we're going to move away from Azure blueprints and take a look at Azure Advisor security assistance. This is Azure Advisor security assistance.

When you access this tool, it will perform a scan of your subscription, and it's going to make security recommendations. Let's take a look at some of those security recommendations. We can see that I've got 12 recommendations in total. We can see there's 36 security alerts for me to view.

So if we scroll down a bit more, you can see that I've been advised about all sorts of different areas.

The idea here, then, is that we would select one of these recommendations, view the more detailed information about that recommendation, and then decide whether we want to act on the recommendation or not.

For each recommendation, you should find a description, general information, and then, if you're lucky, remediation steps, which will give you information on how you can remediate this particular issue. We should be using security assistance every day, as our security parts change as resources are deployed and removed from our subscriptions. In summary, you have learned the importance of the Azure governance tools. You learned how to use Azure policies, initiatives and blueprints, and you learned the benefit of regularly using Azure Advisor security assistance. Next, we're going to take a look at securing Azure virtual networks.

Chapter 6 Securing Azure Virtual Networks using NSGs

In this chapter we'll take a look at Securing Azure Virtual Networks. You will learn about network security groups and their use for securing Azure Virtual Networks. We'll demonstrate network security groups before taking a look at a feature called application security group and how we can use application security groups to simplify the management of network security. We'll finish off by demonstrating application security groups. To help us relate network security groups to your corporate network. We will discuss a sample company's security group requirements. We'll also highlight areas where security groups can help secure your deployments, and hopefully by the end you'll have a good understanding of where network security groups and application security groups can help you. One of the key recommendations when planning your network security is to plan your security based around defense in depth. This means planning multiple layers of security so that if one layer is breached, other layers are still there to protect you. In Azure, defense in depth starts with physical security. This is managed by Microsoft. They will protect their physical data centers and the physical infrastructure inside those data centers. Another layer of your defense in depth is Identity and Access Control. This is managed by you by working with products like Azure AD and within a great suite of products like Active Directory Domain Services and Active Directory Federation Services. At the perimeter of your virtual networks in

Azure, standard distributed protection is enabled by default. You can choose to enable additional layers of DDoS protection if you feel your organization would benefit from the additional monitoring and security that those additional layers will provide. To protect our Azure Virtual Networks and applications, we can deploy network security groups, firewalls, and gateways to offer protection from layer 4 to layer 7 of the OSI model. To protect your compute and data, we would implement the appropriate operating system security and access controls and encryption. A typical deployment integer would implement all of these layers of protection, defense in depth. So let's start off by looking at network security groups then. Fundamentally, network security groups filter traffic. Each network security group has an inbound list and an outbound list. Inbound traffic is filtered through the inbound list and is either allowed or denied, outbound traffic filtered by the outbound list. Each list contains a series of rules, and each rule has a number, a private number with rule 100 having the highest priority and rule 4096 having the lowest priority. As you create rules in each inbound and outbound list, you must get the order right or you might end up with behavior that you didn't expect. Each network security group we create can be attached to subnets on network cards, and each network security group can be linked to multiple resources so we can reuse network security groups. Network security groups are stateful. That means if I allow traffic inbound, the return traffic will be allowed outbound automatically, and vice versa. If I allow certain traffic outbound, the return traffic will be allowed inbound. This makes network security groups relatively straightforward to administer. There are a lot of network security group properties, but they include a name for the network security group, a priority number, the source and destination of the traffic that we're trying to filter, the protocol we're trying to filter, so TCP, UDP, etc. We include a direction for each rule that we create, so inbound to outbound. We include a port range that rules should monitor for, and this could be an individual port or a set of ports. And finally, an action, either allow or deny. So for each rule that we create, we can either allow traffic or explicitly deny it. Imagine that we've got an Azure Virtual Network, and this network contains two subnets, Subject 1 and Subnet 2, and we can see that Subnet 1 contains two servers and Subnet 2 a single server named Server 3. We will use network security groups to control the flow of traffic inbound and outbound to these subnets and servers. If each subnet requires the same set of rules, then they will use a single network security group. We've got a network security group called NSG1, we created that group and associate it with both subnets. And as traffic flows into those subnets, it will be assessed by the inbound rules of NSG1. Server 2 needs slightly different rules, so I can create another network security group called NSG2, attach that to Server 2's network interface, and NSG2, assess its traffic inbound and outbound for that server. We can have granularity of access using different layers of network security groups.

Now we know about network security groups. Let's have a look at another feature that'll make it easier to work with them, application security groups. Network security groups are a great feature, but they can become complex to manage. Each network security group can contain lots of rules, and the more rules they have, the more complex they are to manage. Network security groups can also be difficult to maintain. The more resources we add to our virtual networks, the more we might have to go back and edit network security groups and sometimes several layers of network security groups at that. Anything we can do to simplify the management of network security groups, then, is a good thing and there are several things we can do. First of all, we can use service tags. Service tags are given to us by Azure. Service tags represent services like Azure load balancer, Azure API management, and locations like the internet. In general, if you want to allow strict default traffic, we can use service tags. We can also make use of default security rules. Default security rules allow common outbound traffic such as internet traffic. They also allow traffic from subnet to subnet and traffic from common services like the Azure load balancer. At the same time, the default security rules restrict the flow of inbound traffic. A third option for simplifying network security groups is to make use of application security groups. Application security groups tend to represent a tier of your application. And if we use them correctly, will mean that network security groups are configured once and do not have to be readjusted every time we add a new subnet or set of servers. So what are application security groups? Well, they allow us to reference a group of resources such as web servers, application servers, or database servers. They can be used as even a source or destination of traffic. They do not replace network security groups. They enhance them. So network security groups are still required. When working with application security groups, we create the application security group, we link the application security group to a resource, we then use the application security group when working with network security groups. Let’s say that we have an example of an Azure virtual network with two subnets. If we want to restrict access to these subnets, we can create a network security group and associate it with both subnets. When we create new subnets, we can go back to the network security group and adjust it to include the association with our new subnets. Network security group 1 now is associated with subnets 1, 2, 3, and 4. This generally isn't a problem because we're not adding new subnets every day, but resources are different. We've got two virtual machines, and we want to protect access to these virtual machines by using network security group. As we create the machines, we create a network security group called NSG2, and we associate it with the two virtual machines. These virtual machines are part of our web tier. So as I add more resources in our web tier across a range of subnets, we have to keep constantly going back and adjusting network security group 2. Resources like virtual machines will change much more frequently than subnets will. The more we change the virtual machines across these different subnets, the more we might have to change the network security group. So instead of associating network security group 2 with the constantly changing network interface cards of our resources, we're going to use application security groups. In another example, we've got a fore of nets, and we have an NSG called NSG1. And we create an application security group called web tier that's used by the network security group to allow the appropriate traffic inbound and outbound. With that network security group in place, we just add resources. As we add resources, we just make sure they use the appropriate application security group reference. There's no need to go back to the NSG every time we add a new set of resources. Have a think about your requirements. You might be deploying applications. In that case, each tier would get its own application security group. Resources in your DMZ would also have their own application security group, and as we build in automation pipelines to our deployment, the application security group will be part of that pipeline. So as I'm deploying new virtual machines, new containers, we make sure they include a reference to the appropriate application security group.

In this chapter, we will be taking a look at Azure firewalls and user defined routes. We'll first take a look at Azure Firewall and Azure's distributed denial of service protection. We will go on to discuss user defined routes and how user defined routes can help us route traffic through our security appliances. We will learn about the different firewall protection options available to us. Understanding of the different firewall options available to you will lead to a more secure Azure deployment and a more cost effective Azure deployment. Let's begin with Azure Firewall and DDoS. What is Azure Firewall? Well, Azure Firewall is a stateful firewall service provided by Azure. It's a virtual appliance configured at the virtual network level. It protects access to your virtual networks and is a highly available solution. Features of Azure Firewall include advanced threat intelligence, so the firewall service can learn about the traffic going in and out of your network to determine which traffic is good or bad. It supports both outbound and inbound NAT, reducing our reliance on public IP addresses. Azure Firewall integrates with Azure Monitor for all our porting needs. It includes support for network traffic filtering rules so that we can tightly control the traffic flowing through Azure Firewall, and it's almost unlimited in scale. If we want to support a small deployment of just a few services or hundreds or thousands of instances, Azure Firewall will scale. Imagine that we've got an Azure Firewall in place. We've got resource subnets and a separate subnet for Azure Firewall. The Azure Firewall will have a public IP address, and inbound traffic will be directed towards the Azure Firewall service. We'll then have NAT rules and intelligent protection looking at all that traffic that's coming in and then passing the good traffic towards our resources. Azure Firewall works hand in hand with Azure DDoS protection. Azure DDos protection provides DDoS mitigation for networks and applications. It's always on as a service. So right now if you're using Azure, you're using Azure DDoS protection. It provides protection all the way up to the application layer. And like Azure Firewall, integrates Azure Monitor for reporting services. Features offered by Azure DDoS protection include support, so protection from layer 4 attacks up to layer 7 attacks, attack analytics, ao we can get reports on attacks in progress, as well as post attack reports. Like the firewall service, we have scale and elasticity, so this service will go ahead and try and absorb the attacks that are in progress. Azure DDoS protection also provides protection against unplanned costs. If our service is of scale because of a DDoS attack, then those costs can be recouped. We do need to be aware that Azure DDoS comes in two different service tiers, basic and standard. It is the basic service that's always on for your account. The basic service offers availability guarantees, and it's backed by an SLA, and crucially, is free. The standards here offers everything the basic tier offers, but that includes features like metrics that will monitor when an attack in progress, reports so you can see details of attacks that have happened and where they came from and what the results were. The standards here also offer live support so you can have contact with DDoS experts at Azure during the attack to help you fight against it. The standards here also integrates with your SIEM systems, but the standard tier is not free. You are charged a monthly fee and a fee based on the usage of the feature of the standard tier. Not everyone will need Azure Firewall or the standard DDoS tier. So think about your network that you have now and think about the networks you'll deploy in the cloud. If you use network firewalls now, you'll probably need Azure Firewall in the cloud. If you decide that you need Azure Firewall, you'll have to go through a planning phase. During this planning phase, you will figure out what rules will need to be configured in Azure Firewall. As we said a moment ago, the basic tier is free, but you have to ask yourself, do you need that standard tier? As well as Azure Firewall and Azure DDoS protection, you can also deploy virtual appliances for the marketplace. These virtual appliances can add additional protection or be used in place of features like Azure Firewall.

Chapter 9 Azure User Defined Routes

When working with Azure firewall or virtual appliances, you have to think about how traffic is routed around your virtual networks. There may be a requirement to alter the default routing, so that traffic flows through your firewall or virtual appliance you are deploying. And that's where user defined routes come in. When traffic flows around your Azure Virtual networks, it is governed by a set of system routes. These system routes were enabled by default. System routes make sure that resources deployed to different subnets can communicate with each other and the resources deployed to your subnets can connect out to the internet. User defined routes allow us to override Azure's default system routes. They're most often used when we want to fill our outbound traffic through a virtual appliance. Imagine that we got three subnets, and I've got virtual machines deployed to Subnet 1 and Subnet 2. Using the system routes, if traffic wants to leave the internet, traffic rerouted directly from Subnet 1 to the internet and direct them to Subnet 2 to the internet. But let's say I want to deploy a virtual appliance to Subnet 3. I want all traffic going out to the internet to be filtered through that virtual appliance. This is where user defined routes come in. Or we've got the same three subnets, but now I've introduced user defined routes. Can we see how they user defined routes are now filtering the outbound traffic towards our virtual clients first? This virtual appliance can inspect the traffic. It may even alter the traffic before allowing that traffic outbound to the internet. Subnet 3 itself is governed by the system routes for internet access. So using user defined routes, we have a lot of control over the flow of traffic between our subnets and towards the internet. We've got a lot of Azure security options available to us. Let's see if we can try and differentiate between a few of them. So far, we've discussed Azure Firewall and Azure DDoS protection. So hopefully we've got an idea of where those two fit. We also have a feature called Azure Web Application Firewall. Azure Web Application Firewall is designed to publish your applications to the outside world, whether they're in Azure or and lures bound traffic towards them. We've discussed network security groups previously, so hopefully we have a good idea of where network security groups fit into our security. We also have a feature called forced tunneling. Forced tunneling allows the control of flow of traffic. So instead of traffic being routed directly into that, traffic is sent first where your security monitoring tools can assess that traffic and decide what is allowed or not. Finally, you can deploy any marketplace devices available. So if you already have an existing skill set and you want to take advantage of licenses that you already own, you might find a marketplace device that better suits your needs. To help determine when we need to use these six different types of security options, think about these three scenarios. In scenario one, you want to control the flow of internet traffic. You wish to control the flow of traffic heading to the internet so that it can be inspected at layer 7. In scenario two, you've got an SQL Server. Only traffic from your Azure subnets should be allowed to access the Azure SQL Server. In the final scenario, all traffic that's generated by your application servers must be routed through HQ. So take a minute, think about these three scenarios, and decide which of the previous six security solutions would you implement to satisfy the needs of each scenario. There can be a variety of answers here. If I was asked this question, for scenario one, I would say we deploy user defined routes. The user defined routes will allow us to control the flow of traffic, and then would even deploy Azure Firewall or a marketplace device to filter that traffic. For scenario two, well, it's network security groups. Network security groups will allow us to control which traffic is allowed to communicate with those SQL Servers by restricting the source of traffic that those SQL Servers will accept. And for scenario three, forced tunneling. If we want to force traffic to HQ, forced tunneling is what we need.

Chapter 10 Azure Information Protection & Security Monitoring Tools

In this chapter we're going to be working with Azure Security and Reporting Tools. We're first going to introduce you to Azure Information Protection. We're then going to discuss Azure Monitor and Service Health, and we're going to introduce you to Azure Key Vault. We're then going to introduce you to Azure Sentinel before introducing you to and demonstrating Azure Security Center. By the end, you'll understand the core monitoring tools available in Azure, and you will understand the benefits of these tools to your organization. Let's start off then with Azure Information Protection and security monitoring tools. Azure Information Protection is a pretty useful tool. We use Azure Information Protection to classify documents and emails. Azure gives us some classifications that we can use, and you and the stakeholders in your organization will create classifications relevant for you. Classifications are labels that are attached to documents Think of security levels like open, secret, these can all be classifications that your company uses. Each classification can be added as a label to a document. Once documents are labeled, they can be protected. This protection comes in the form of encryption, and in order to gain access to the documents in the future, you must be assigned the appropriate rights. These labels go beyond standard permissions because these labels will stay with the document no matter where it is, whether that document is on your SharePoint service in Azure Storage, or even if that document is downloaded onto removable media, the labels and its protection stay with the document. Azure Information Protection labels can be applied automatically, can be applied manually, and we can recommend the use of different labels to our users based on the content that they are creating. There are two sides to Azure Information Protection. First of all, the classification of documents. These classifications come in the form of metadata that can be attached to the header or added as watermarks to the documents you're trying to protect. Once classified, then the documents can be protected. Azure uses Azure Rights Management to encrypt the documents using Rights Management templates. Then when a user wants to again access to a document, they can apply for a certificate that will grant them the appropriate rights. This process tends to be automatic and performed by the applications that the user is using. Ideally, documents and emails will be classified when they're created. But you'll probably have a load of existing documents, even Azure or that need to be protected. For data stores, you can use Azure Information Protection scanner to scan those data stores and apply labels, and then those documents can be protected. For data stores, we can use Microsoft Cloud App Security. Just like Azure Information Protection scanner, this tool will scan your cloud stores, classify documents so we can apply protection to them. Azure gives us a lot of security and reporting tools. Three tools that are commonly used include Azure Monitor to collect and analyze metric information, both and in Azure, Azure Service Health that we can use to see the health status of the Azure services, and Azure Advanced Threat Protection that can be used to detect and investigate attacks against our users. Let's chat about these three tools in a bit more detail. Azure Monitor is used to collect, analyze, and act on telemetry. It acts as a central point where resources both in Azure and can report information. We can use the information reporting to troubleshoot issues and run in performance. There were two types of data collected by Azure Monitor, metric information from services that we've deployed and logs from services we've deployed. And again, these metrics and logs coming from both Azure deployed resources and Azure Service Health is a service that's automatically provided to us by Azure. It notifies us about the status of Azure services around the globe. Through Azure Service Health, we can see the reports of incidents that have occurred, such as downtime to services and downtime to regions and availability zones, and planned maintenance that might affect the services and resources that we're consuming in Azure. Azure Service Health offers personalized dashboards that you can use to view the health status of services that are relevant for you and your organization, configurable alerts so you don't have to be sat in front of a dashboard when a situation occurs. Instead, you can receive emails that will alert you to the changing of a service's status, and guidance and support that will help you navigate through a service issue. Azure Advanced Threat Protection is all about your users. It can be used to monitor and analyze user activity, it's used to identify suspicious activity events that you can then try and protect against. Azure Advanced Threat Protection works with both Azure and your Active Directory forests. It can be used to identify reconnaissance attacks, compromised credentials, lateral movements, when an attacker gains access to your network and then slowly moves through your network looking for vulnerable systems, and domain dominance where legitimate credentials have been used to gain access to your forest to perform malicious activity.

Chapter 11 Azure Key Vault Basics

A long time ago, it took a lot of effort to encrypt and, especially, decrypt information. Today, most secure operations like that is done in specialist hardware. So particularly in the cloud, we attempt to encrypt everything. The conversation today, then, is about secrets management. Where do we keep the secrets that are used to perform our encryption? Where do we keep the secrets that are used to access Azure secure resources? Azure Key Vault is a service we can use to protect our secrets. Secrets come in many forms like certificates, keys, connection strings, passwords, and it's the secure management of the secrets that can be an issue for IT. How do we control access to things like passwords, API keys, and of secrets? Have a little think. How do you control access to passwords and API keys right now? Are they stored securely? How do you create and control encryption keys? Where are they stored and who has access to them? And how do you provision, manage, and deploy digital certificates? You need to answer these questions, and your answer should involve some form of secure storage for your secrets, date audited, and only select people and applications have access to it. And this is where Azure Key Vault comes in. Azure Key Vault can be used to centralize the storage of application secrets. Azure Key Vault uses hardware. It uses hardware security modules. These hardware security modules have been validated to support the latest Federal Information Processing Standards. That means if you want secret storage that can be used while you're working on government contracts, Key Vault can work for you. When using Azure Key Vault, we can enable logging to monitor how and when our secrets are being used. And crucially, Azure Key Vault can just centralize administration of all our secrets. Azure has some recommendations for us when we're using Key Vault. Firstly, we should use a separate Key Vault for each application that requires centralized key management. This makes it easier for us to control access to sets of keys and secrets on an basis. We should also make sure that we take regular backups of all our Key Vault That way, if something goes horribly wrong, we can restore from those backups. We should turn on logging and set up alerts to inform us when certain events occur, and we should also enable soft delete and purge protection. This makes it even easier for us to recover secrets if they've been accidentally deleted and protects the Key Vaults themselves from accidental deletion.

Now, we're going to take a look at Azure Security Center and Azure Sentinel. Security is always a challenge, but in the cloud, it's even more of a challenge than In part, that's because services in the cloud changed rapidly. How do we know that changes to Azure services meet our security requirements? As security tools improve, attacks are becoming more sophisticated. But how do we keep up to date with new threats? We also have to contend with a skills shortage. We have lots of information available to us in the cloud, but do we have all the stuff we need to digest and act upon that information? Azure Security Center is designed with these challenges in mind. If you are using Platform as a Service services in Azure, like Azure Web Apps, then Azure Security Center is already working for you, monitoring those services and reporting on their security status. There's no need for you to do anything for those services. For services, like operating systems you deploy in the cloud, we can deploy monitoring agents to gather security information. As well as monitoring for security threats, Azure Security Center also reports our compliance status against certain standards. It provides continuous assessment of existing and new services that we deploy. It also provides threat protection for both infrastructure and Platforms as a Service services. One of the newer security services provided by Azure is Azure Sentinel. Azure Sentinel is both a security information and event management system and a Security Orchestration, Automated, and Response system, so both a SIEM and SOAR solution. Azure Sentinel offers a single solution for collecting data at cloud scale. It can then take that data and help detect previously undetected threats. Azure Sentinel looks for threats using artificial intelligence. It allows us to respond to incidents rapidly, much quicker than traditional monitoring tools would have highlighted the threat. With Azure Sentinel, you connect to your security sources using data connectors. These sources can be storage, virtual machines, security appliances that we want to collect information from. We can then analyze the data collected using Azure Monitor Workbooks and Analytics Workspaces. These are very powerful tools that allow us to search for and display information that's relevant for us. We could then take the information we've gathered and use that to trigger security automation using Orchestration Playbooks. Playbooks are an extension of Azure Logic Apps. Logic Apps are a serverless workflow service that allow us to automate a series of steps. In this example, perhaps a series of steps in response to a security threat that Azure Sentinel detected. With Azure Sentinel, we could perform deep investigation and hunting. This allows us to discover isolated pieces of information that by themselves might seem innocuous, but when brought together, help us discover a security threat.

Chapter 13 Azure Service Trust & Compliance

In this chapter we'll be taking a look at Azure compliance and data protection standards. We will first discuss some of the common industry and compliance standards that Azure supports and your company might be trying to achieve. We'll discuss Microsoft's privacy statement and what that statement means to you and your organization. We'll introduce Azure's Service Trust Portal, and we'll also discuss Azure special regions as well. Understanding compliance will lead to secure Azure deployments, save your company money, and help win contracts you may otherwise have lost. So we'll start off by talking about some of the industry compliance standards that Azure supports. A lot of us work in industries where some form of regulatory compliance is required. Regulatory compliance is the process of ensuring that you follow the standards or laws laid out by a governing body. As a company, we'll employ people and develop processes to help detect and prevent violations of our security regulation requirements. Compliance monitoring can be complex, particularly in today's hybrid world where we have both and cloud systems to manage. Azure provides several tools to help us assess and maintain our compliance posture. There are lots of compliance standards out there that you might have to adhere to. Here we've got a selection of some of the most common standards. First of all HIPPA, the Health Insurance Portability and Accountability Act. HIPPA is a US standard that lays out data privacy and security provisioning for safeguarding medical information. We have PCI, the Payment Card Industry Standard. PCI lays out standards for securing the handling of credit cards. In the EU, we have GDPR, General Data Protection Regulation. This is an EU standard for data protection and privacy. FedRAMP is the Federal Risk and Authorization Management Program and is an information security standard used by the US government. Any organization working with the US government at any level would have to adhere to FedRAMP. We then have standards laid out by the International Standards Organization, such as ISO 27001, which is part of their information security standard. These five are just examples. There are many more standards that you might have to adhere to. And although we will have to keep our own proofs and documentation proving that we're adhering to these standards, Azure gives a series of tools that we can use to measure our services against these and many more. Azure supports more than 90 global compliance standards. They provide documentation about their services and which standards their services are certified against. And Azure gives guidance on what areas of these standards are our responsibility and which areas are theirs. Azure supports over 35 offerings with documentation to support the rollout and management of compliance features. If our Azure subscriptions have to be deployed to support a particular standard we can use Azure Blueprints to deploy a complete environment configured to our compliance needs. As well as providing tools that allows us to honor our compliance posture, Azure itself is certified against all the standards that it supports. And Azure offers access to reports from orders that gives us proof of their compliance. One powerful compliance tool that we outlined in module 7 is Azure Security Center. As part of Security Center, we can see our current compliance posture measured against some of the most common standards like GDPR and PCI. One of the most important tools for modeling your compliance posture and for viewing information of Azure's compliance status is Azure Service Trust Portal and Service Trust Center. The Azure Service Trust Portal is probably the first tool you will use on your compliance journey. In this portal you will find details of Microsoft's implementation of controls and processes against Azure and other Microsoft products. Although anyone can access the portal, if you log in as an authenticated user you'll get access to information not visible to everyone. When you log in you need to log in with a Microsoft cloud service account. This could be any user in Azure Active Directory with the correct delegated permissions. In the portal you will find Compliance Manager. This is your section of the portal where you can document the steps you are taking to become compliant against certain standards. You'll find trust documents. These include Microsoft security information, as well as design information that will help you on your compliance journey. You will find compliance information specific to different industries and regions, as well as links to Microsoft's general trust center. You will also find My Library. This is where you can save and access your compliance documents so all your compliance information is in one place. Microsoft Trust Center is accessed through the Trust Portal. Through Trust Center you can view security, privacy, and compliance information, access to Microsoft product compliance information, and you'll find compliance tools such as compliance score, audit reports, and data protection resources. So let's take a look at the Service Trust Portal and Microsoft Trust Center.

Chapter 14 How to use Azure Trust Center & Compliance Manager

In this demonstration, we're going to be working with Azure Compliance Manager and the Azure Service Trust Portal. This is the Service Trust Portal.

Notice the URL, servicetrust.microsoft.com, and the fact that I'm logged in. I've logged in with a user account that's been assigned the compliance role in Azure AD. Through this portal then, we can access Service Trust, Compliance Manager, Trust Documents, and if we click on the next to More, we can see links, amongst other things, Industries and compliance information, Trust Center, and My Library.

If we scroll down on this first page, here we can see links to independent Audit Reports for Microsoft Cloud Services. These provide information about Microsoft's compliance status. Notice here FedRAMP, ISO 27001, and the link where you can View all Audit Reports. Scroll down a bit further, and we get to Documents and Resources.

Through here, we can see the results of penetration tests and security assessments carried out by third parties. We can see information on Azure Blueprints that we can use to rule out compliant subscriptions, as well as access to documentation, including White Papers, FAQs, and Compliance Guides. Let's go back up to the top of this page, and we'll select Compliance Manager.

If you're not logged in with your proper credentials, you'll get an error when you try to access Compliance Manager. If you're not familiar with Compliance Manager, you can take a tour. If you are, you can scroll down and get started. Compliance Manager is all about assessing your deployments against certain standards.

So here you can see I'm measuring my Office 365 deployment against GDPR and NIST, I'm measuring Azure against ISO 27001, as well as GDPR, FedRAMP, and the UKNHS standard.

If I select Azure GDPR, at the top here, we could see Assessed Controls, and mine indicates that I'm 36% Assessed, and that would be 47 out of 130 controls.

And the statement of this assessment is In Progress. Let's scroll down and see a few more details about the assessment. We should see three sections, Azure Cloud Services, Microsoft Managed Controls, and Custom Managed Controls.

Let's view the Azure Cloud Services. There's a long list here.

Each one of these services then can be secured against the GDPR standard. You should be able to see a similar list for each of the standards that Compliance Manager supports. Just because of service is on this list, it doesn't mean it's fully compliant. To be fully compliant with standard, we have to break down the responsibility between ourselves and Microsoft. If I minimize this list, we have Microsoft Managed Controls, these have passed the standards that Microsoft is responsible for, and we have Custom Managed Controls, these have passed the standards that you're responsible for implementing.

If we view the Microsoft Managed Controls, here we can see a number of assessments in different categories and the number of assessments that have been assessed.

If we take a look at Rights of individuals, for example, this contains one control detailed on the side, we can see a Test date, and the fact that this was tested by a independent auditor. Crucially, we can see the Test result. So Microsoft passed this test. You will be affined the independent audit, which provides proof that Microsoft passed this control. Let's collapse the Microsoft controls, and now take a look of the Customer Managed Controls.

These controls are your responsibility. In order to be GDPR compliant, you would have to make sure that all of these controls are enforced. Let's take a look Right of individuals, for example. So there are 11 controls as part of this section, and none of them have been assessed.

As an organization then, it would be up to you to research the relevant articles and put procedures in place to enforce the article controls ready for assessment. In this section, I can assign this control to an individual by using Assign option. I can also implement status information, and we can provide a date for status changes. Using Manage Documents, we can upload documents that are relevant for this article, that perhaps outline the steps we have taken to fulfill this article. We can also provide dates of tests that have been run to assess adherence to these controls and the results of those tests. Ideally, your tests would also be run by a independent auditor. What Compliance Manager is given us then is a way to visualize our readiness for assessment. Instead of you having to fight your way through all the different controls for a particular standard, and then document your adherence to those controls in a tool, we have all the information we need here in one place. Let's go back to the Service Trust Portal.

If you're trying to find the Microsoft Audit Reports, you can find that from the Trust Documents section, along with information and information on Azure Security and Compliance Blueprints. If we select More, one useful section is Resources.

Through here, we could see information about Microsoft's Global Datacenters, access Frequently Asked Questions, and access the Security and Compliance Center. The Security and Compliance Center offers more general information around security and compliance, including security information on Office 365. If we select More again, and select My Library, it's in My Library where you could save reports, as well as white papers and other resources that will help you document your compliance posture.

There's one other website that I want to show you in this demo. This website, privacy.microsoft.com, gives us access to Microsoft's Privacy Statement. Here we can find details on the type of personal data Microsoft collects about you.

It gives information on how Microsoft uses that personal data, reasons why they might share your personal data, and how we can access and control our personal data. This statement was last updated in January 2020, and it's well worth you getting to know this statement, so you and your organization have a good idea of how your personal data will be used by Microsoft. As this statement gets updated, it's also somewhere that you should visit on a regular basis.

Chapter 15 Azure Special Regions

When a deployment's launched in Azure, we deploy to Azure regions. Most regions are equal, but some are special. In this section, we're going to cover Azure special regions. Azure services are deployed to multiple data centers around the world. For ease of management, high availability, and disaster recovery, these data centers are grouped into regions such as West US and UK South. You can deploy your resources to a region that matches your requirements. Requirements are based on cost. Some regions are cheaper than others. You might choose regions that are closest to your customer base all because the services that you want are only available in certain regions. Azure special regions exist for compliance and legal reasons. These regions are not generally available, and you have to apply to Microsoft if you want to use one of these special regions. Currently, there are three categories of special regions. US Gov regions support US government agencies. This includes US Gov Virginia and US Gov Iowa. We have the China special regions. This includes China East, China North. The China regions are managed in partnership with 21Vianet. We also have the Germany regions, Germany Central and German Northeast. These regions are managed for a data trustee model. This model means that data in the Germany regions will reside in Germany only and be compliant with German data laws. You must request access to Azure special regions if you want to deploy resources to them. The US Gov regions are for additional compliance certifications such as FedRAMP and DISA Level 5 DoD approval. So if you're working on a federal, state, or local US government project and you need to adhere to the standards, you could apply for access to the US Gov region. Regarding the China regions, it's worth pointing out that Microsoft does not directly maintain these data centers. They are owned in partnership with 21Vianet. China has some very specific rules about how data enters and leaves the China regions, and by partnering with a Chinese organization to manage these data centers, Azure could adhere to these rules. Germany has very specific rules about data residency. To conform with these rules, all data in the Germany region stays under the control of is a company owned by Deutsche Telekom.

Chapter 16 Azure Compliance Resources

In this chapter, we're going to be discussing Azure compliance resources. The goal is to introduce additional compliance material that Microsoft makes available online. One of Microsoft's building blocks for their online services is Trusted Cloud. Trusted Cloud is built on a series of foundations, including security, privacy, and compliance. Building on the security foundation, Azure helps keep our customer data secure. Azure provides network and infrastructure security, data encryption, strong authentication and authorization, as well as tools that allow us to audit access to Azure subscriptions so that we can keep track of who is accessing data. One of the principles of the Privacy Foundation is that Azure customers own and control their own data. We get to decide the regions that our data is stored in, we have the ability to access our data from any location at any time, our data is encrypted both at rest and in transit, and there's even rules of how Microsoft will dispose of our data if we cancel our subscriptions. To help us satisfy compliance, Microsoft currently supports integration with over 90 international compliance standards. These include global, national, and compliance standards that we can certify against. To help you build secure, compliant architectures in Azure, you should spend time with the Microsoft Cloud Adoption Framework. This framework outlines cloud adoption best practices that have been brought together from Microsoft employees, partners, and customers. The guidance laid out in the Cloud Adoption Framework is not only for IT decisionmakers, but also for business leaders. The Cloud Adoption Framework helps you define a business strategy for cloud adoption. It provides guidance on best practiced governance of Microsoft Cloud Services. So if you're starting out on your Microsoft Cloud journey, the Cloud Adoption Framework should be one of the first resources that you use. It will keep us away from bad practices and will help all parts of the business, both technical and nontechnical, understand why the cloud is right for them and the best way to adopt cloud services for each department's particular requirements. If you are responsible for compliance for your organization, then I encourage you to seek out the Azure compliance documentation. This documentation is your starting point for learning about compliance in Azure. The Azure compliance documentation is organized into both regional and global compliance offerings, as well as several offerings. These include compliance standards aimed at financial services, the automotive industry, media, and energy companies. By using the Azure compliance documentation, you can find out how Microsoft can help you conform to compliance standards such as GDPR, HIPAA, and FedRAMP to name just three of over 90 compliance offerings that Microsoft Online Services supports today. Most organizations are keen to keep personal information private, so understanding what personal information Microsoft collections from us and how it uses that information is very important. The Microsoft Privacy Statement explains how Microsoft collects and processes personal data and for what purposes that personal data is collected. The privacy statement includes information. This means we can identify the type of personal information that each individual product is collecting from us. This helps understand what information might be exposed when we're using individual products. Microsoft collects and uses our personal data for things like improving and developing products, personalizing products and making recommendations, advertising and marketing, and performance analysis and research. You can find the Microsoft Privacy Statement online, and it's definitely something you and your team should seek out so you can have a clear understanding of what personal information each Microsoft Service collects and how it uses that data. When you start to use any Microsoft online service, such as Microsoft Azure, Microsoft 365 or Microsoft Dynamics, you are agreeing to a set of Microsoft terms and conditions. These terms and conditions are laid out in agreements such as the Microsoft Online Subscription Agreement. Amongst other things, this agreement lays out acceptable use and unacceptable use of Microsoft Online Services. This agreement is an agreement between Microsoft and the organization you're representing or Microsoft knew if you didn't specify an organization when you signed up to an online service like Azure. Depending on the industry you're in or the region you're in, there may be other subscription agreements that you're also agreeing to when you first sign up for Microsoft services. So again, a bit like the privacy statement, seek them out and make sure you're happy with the terms of agreement before you start consuming Microsoft Online Services. Another useful piece of documentation is the Online Service Terms. These lay out the terms by which each Microsoft Online product is offered. It's through the Online

Service Terms that you find the licensing agreements for each of Microsoft's online services. One addendum to the Online Service Terms is the Online Service Data Protection Addendum. This addendum lays out how data is protected, what Microsoft's responsibilities are for data protection, and what the customer's responsibilities are. In today's world where data protection is a key aspect of what we do, a good understanding to the addendum is vital. Microsoft also published their service level agreements for each of their services that are bound by an SLA. On a more technical note, when we initially think about cloud, we often think about shared resources, so shared compute, shared storage that our company might be sharing with other companies that are also using Azure. But if the compliance standards that we're hoping to certify against require additional levels of isolation, then we might consider using dedicated hosts. Azure dedicated hosts provide physical servers that we can use to host one or more virtual machines. Each dedicated host is dedicated to a single organization. This gives you isolation that can help you address certain compliance requirements. Dedicated hosts also give you visibility of the underlying cause, and this can help you meet software licensing requirements, which can help you bring your own licenses into the cloud. Dedicated hosts may not be the first thing we think about when we think about compliance in Azure, but by giving us our own dedicated servers that allow us to isolate our virtual machines from the virtual machines of all the other customers in Azure, dedicated hosts can be an important part of our compliance tool set. In summary we started off by looking at Identity and Access Management. We looked at Azure AD and discussed how important Azure AD is for Azure cloud security. But not only Azure, Azure AD underpins security of all

Microsoft cloud platforms. We mentioned Azure AD Domain Services, Microsoft's managed directory service in the cloud, we discussed access control and how it can be used to provide granular access to Microsoft cloud services, such as Azure and Office 365. We discussed the use of and custom roles in RBAC, as well as the three most commonly used roles, Owner, Contributor, and Reader. We also demoed how we could use these roles to provide access to resource groups. We then moved on to discuss governance tools and secure virtual networks. We discussed governance tools like Azure Policy, Azure Initiatives, and Azure Blueprint, and we saw how each one of these builds upon the other with Azure Policy enforcing controls during deployment and Azure Initiatives that we can use combine a group of policies so they can all be applied in one go. We then discussed Azure Blueprints and saw how they can be used to deploy a secure and compliant subscription. We looked at security and virtual networks. We have network security groups and application security groups, and we discussed how application security groups can simplify our network security group deployment. Next we focused on Azure firewalls and routes. We discussed Azure Firewall and its capabilities, we discussed Azure DDoS protection for our virtual networks, and then we discussed routes and how we can use routes to control the flow of traffic. We also showed you a selection of Azure security solutions and identified different use cases for those solutions. Finally, we discussed security and compliance. We talked about Azure Information Protection, we discussed and demonstrated Azure Key Vault, we discussed Azure Monitor, and we discussed and demonstrated Azure Security Center. We then moved on to discuss compliance. We discussed some of the common compliance standards, we discussed Azure special regions, and we had a look at Azure Trust Center and the Azure Trust Portal along with Compliance Manager and Microsoft's Privacy Statement. Next, you have learned about some of the common industry compliance standards. We discussed the Azure Service Trust Portal and how we interact with it, and you learned about Azure special regions.

BOOK 3

MICROSOFT AZURE

PRICING & SUPPORT OPTIONS

AZURE SUBSCRIPTIONS, MANAGEMENT

GROUPS & COST MANAGEMENT

RICHIE MILLER

Introduction to Azure Subscriptions

Azure subscriptions are the first construct of Azure, and in the following chapters, we're going to learn all about Azure subscriptions, how to use them, why they're there, and other aspects of Azure subscriptions. We're also going to learn about management groups and how those work with Azure subscriptions. The first thing is we're going to describe an Azure subscription and understanding what it is, understanding the different uses and options with Azure subscriptions, and then we're going to understand management groups, when you would use them, why you would use them, and how they fit with subscriptions. Let's look at the scenario first that we'll be covered and you'll see throughout the duration of this entire book. The scenario is that we work for a company that has decided to adopt cloud. The company is exploring different cloud providers, not just Azure, but Azure is one of those cloud providers. One of the very first steps of adopting Azure is to understand how the pricing and the support options work to make sure they fit your organization's needs. You have been tasked with learning the fundamentals about the pricing and support options. You need to document these and bring these back to the team and report on these to help the team make the decision on if they're going to go with Azure or if they're going to go with another cloud provider.

Chapter 1 How to create an Azure Subscription

The first thing with adopting Azure is creating a new subscription and then tying that subscription to an account and deploying your cloud resources that you will consume into the subscription. At the top level, you have the subscription, and then you have what's called a resource group and a resource group is used to group your resources that share a lifecycle. Each resource is going to belong to a resource group, and then the resource group is going to belong to a subscription. So regardless if your resources are web apps, databases, virtual machines, maybe a Kubernetes cluster, they're all going to be associated with some sort of resource group, and that resource group has to be associated with a subscription. An Azure account is used for contact information and billing details for an Azure subscription. Every time you spin up a new subscription, it has to be associated with an Azure account. There's an email tied to an Azure account, and the person that owns that email is responsible for the monthly cost of the Azure consumption that happens in that subscription. An Azure subscription is just a logical container and grouping of Azure resources and administration. The different elements of an Azure subscription are it's a legal agreement, it's a billing unit, it's a logical boundary of scale, and it's also the very first container that's created, and it's an administrative boundary. So let's walk back through these. So it is a legal agreement, so if you have charges against a subscription, you are legally bound to paying for those and also the use of the subscription and the things that are done within that subscription. It's a billing unit, so you can have many subscriptions, and you can have different billing tied to different accounts and go to different, let's say, like departments or people. So if you need to separate billing for any reason, a subscription might be a way to do that. It's also a logical boundary of scale, so you can only have a certain amount of VNets deployed into a subscription, or you could only have a certain amount of web apps deployed into a subscription. Microsoft has soft limits around different amounts of resources that can be deployed in a subscription. If you need a higher amount, you can reach out to Microsoft and ask to have that increased. But just keep in mind that the subscription can be a logical boundary of scale. Then the first container created is you create this subscription before you create any resource groups or before you create any resources. You have to have that subscription there. But let's talk about the relationship between Azure Active Directory and subscriptions. This is key, and this is important to understand. What is Azure Active Directory? It's Microsoft's identity and access management service that runs in Azure itself. It's used for being able to sign in and access cloud resources. A subscription is going to have a trust relationship with at least one Azure Active Directory. An Azure Active Directory can have trust with multiple subscriptions, but each subscription could only trust one single Azure Active Directory, and this is something important to keep in mind as you're working with subscriptions and you're deploying more subscriptions. In your Azure Active Directory is where your accounts will live, like your user accounts and your different groups. So when you're assigning permissions to subscriptions, you're assigning permissions to resource groups, etc., keep in mind the relationship between Azure Active Directory and your subscriptions, and that will help you understand how things work as you go to assign permissions.

But when should I add a new subscription? The first question that you'll ask when you want to know, should I add another subscription here or not is going to be do you have concerns of exceeding limits within a subscription? Are you going to add more VNets than Microsoft allows in a subscription? If the answer is yes, then you'll add a subscription. If the answer is no, then you'll move to the next question. Do I trust the subscription owners? When you spin up a subscription, you will be a subscription owner, but you can also assign that permission to someone else. That could be another department head or just another person on your team. Maybe you need to separate that security. Maybe you don't want that person having access to the resource groups and resources in your subscription. If the answer is no and you don't trust the subscription owners, then add another subscription, and you will add that person to that additional subscription. If the answer is yes, then move on to the next question. This question is, do you need to constrain or scope resource providers within a subscription? A resource provider is there in Azure behind all of the services. For example, virtual machines in Azure have their own resource provider, SQL databases for the PaaS SQL offering have their own resource provider. So maybe you want to have a subscription and you want to allow folks that will be consuming from that subscription to deploy VMs, but maybe you don't want them deploying Azure App Services to run web apps. Maybe you want to have that capability in a totally separate subscription. You could certainly do that, and you could scope the resource providers at a subscription level. So if the answer is yes, that you need to scope a subscription based on resource providers, then you'll add an additional subscription. If the answer is no, then you move on to the last question, and that is can administration be delegated through RBAC controls? If the answer is no on that, then you'll add an additional subscription. If it's yes, that's usually the criteria that we look at when deciding, do I need another subscription or not? Naming of your subscriptions is absolutely critical. This should be done as a part of your planning and really as the first step. This can help with things like chargeback, help IT teams find and managed resources, and just overall be an aid in the operations of your Azure environments. There's many ways to name subscriptions, and you may come up with your own method. At the top level we have the company name, and then we're going to add a department to the name, and then we're going to add a location to the name, and then we're going to tag it with an environment. We'll have that be a part of the name, and then we'll increment the instances with a number. So if we have several of the same names, we'll add an additional number, like 1, 2, 3. The pattern starts with company, department, location, environment, and then instance.

Chapter 3 How to Provision a New Azure Subscription

As far as provisioning a new Azure subscription, you have a couple of options. The first option is you could come out to this azure.microsoft.com/ site:

You can click on this Start free or you could click on the Or buy now below. I'm going to go ahead and click on the Start free. This is kind of the first way that you could sign up for a subscription. The other way is you could add a subscription from an existing account.

Here you would need to put in either an email account, or you would need to go ahead and create one, or you could sign up with your GitHub account. We're not going to go all the way through the process here, but we're going to take you partway through the process so you can see what it looks like. So I'm going to go New, and I'm just going to paste in just an example account, and then we'll go Next. Then it wants a password, so we'll give it a password, and we'll click Next. We put in our information, and then we verify our identity. The next step would be to identify by a credit card.

You do have to have a credit card for this. Note that this would be the free account, so you would have the $200 credit and you could use only free services on here, so you won't be charged. Then the last step would be to go ahead and accept the agreement, and then you would have your new free Azure account. We're going to switch gears, and we're going to go ahead and look at provisioning a new Azure subscription from an existing account. I'm going to click on

That's the offer that we're going to go with here and then it asks for payment information.

Then it's going to ask you what type of technical support do you need on this new subscription? I'm going to say no for none. Then you accept the agreement and then go ahead and click on Sign up, and it will provision your new subscription.

Setting up the account sometimes can take a few minutes. Then when it's done, it brings you into the Azure portal right into this Quickstart Center.

We can go ahead and click on Subscriptions, and it will bring us to the list of subscriptions. That was the process of provisioning a new subscription, to an existing account.

Management groups allow you to apply governance conditions a level above subscriptions. The governance conditions are talking about access, so RBAC and Azure policies. Azure policies are a way to either audit or enforce conditions on a subscription, resource groups, or resources in a subscription. For example, maybe you want to only allow people that have access to your subscription to deploy in a certain region. An Azure policy can be used to do that. So if you have many Azure subscriptions, you can use management groups to help make the management of those subscriptions easier. Let's look at the container hierarchy in Azure, and this is not to be confused with Docker containers. At the top level, you have your management group, and then you have your subscription that rolls into that, or you could have multiple subscriptions. Then you have a resource group that rolls into the subscription, and then you have resources that roll into a resource group. This is a breakdown of the overall container hierarchy from top to bottom. How to use management groups with Azure subscriptions. So when we deploy our first management group, that's what's known as the root management group. We have access and Azure policies applied to that. We can have additional management groups roll up to this root management group. Let's say for example we have a marketing management group, and we have separate access controls and we have separate Azure policies apply to that management group and we have a subscription there, and then resources inside of that.

We also have an IT management group, but we have groups under the IT management group. In there, we have one for development and one for an infrastructure team. The development management group and the infrastructure management group both have multiple subscriptions. Something to note is we're applying the access and the Azure policies at the IT management group level, and the development and the infrastructure management groups are inheriting those permissions and Azure policies and so are the subscriptions underneath those and the resources as long as we didn't break the inheritance. This is a visual of how management groups fit into the overall picture of your Azure container hierarchy and how subscriptions fit into that picture as well.

Chapter 5 Azure Planning & Management Costs

A key component of adopting Azure Cloud is knowing cost. It is important to map out the Azure products and services that will be used and their initial and costs. In the following chapters we will explore the various pricing options and calculators that are available to assist us when estimating cloud. Understanding options for purchasing Azure products and services will be the first topic and then we'll learn about the Azure free account and what that means and what that comes with. We'll look at different factors that can impact your cost in Azure. We'll learn about zones for billing purposes and then best practices for minimizing Azure costs. Then we'll take a look at the different Azure pricing calculators and how you can use those for estimating. The first one here is the Azure free account. With this account, you get $200 in Azure credit that can be used within 30 days. You also get access to free Azure services. The second type is the So you will be charged monthly for the services that you use in a billing period, which is the monthly billing period. The next one is the student account, and you get $100 in Azure credits that can be used in a period. No credit card is required to sign up or use this type of Azure subscription. The final one is the enterprise agreement, and this is where you purchase Azure services and Microsoft software under a single agreement, and this is usually used for organizations to purchase their Azure credit from Microsoft.

Chapter 6 Azure Free Subscription & Free Services Options

What are the options for purchasing Azure products and services? Well, let's go through the list. First, you have your enterprise agreement, which we just touched on, and these are usually designed for very large organizations. This comes with the premier support and dedicated Azure resources. There's generally an annual commit for spend with the Enterprise Agreement. What that means is that you have committed to spending a certain amount on Azure on an annual basis. This happens at your organizational level, and usually with this, you can get deep discounts on your Azure cost. The second is the Direct - from Microsoft. So you'll get a bill for Microsoft, you'll have a support plan through Microsoft, and you or you can use a partner to help you with your Azure provisioning, your deployment, and basically building out your cloud environment. The last is the Indirect - Cloud Solution Provider or also known as CSP. You will receive a bill from the CSP, not for Microsoft, and you will get your support through the CSP, not directly through Microsoft. And you'll work with the CSP for any Azure provisioning, deployment, and uses management. There are some benefits to working with the CSP. For example, a lot of times CSPs will have very talented folks as a part of their organization that you will be able to work with. Now, let's talk about the Azure free account. You can go and sign up for this and you will have access to many popular services for free for 12 months. Again, you also get that $200 credit for 30 days. Then you'll have access to 25+ services free forever. What are some of the popular services that are free for 12 months? You have Linux VMs, you have Windows VMs, and you have 750 hours for each one of those. You have other things like managed disks. You have access to 5GB of file and blob storage. You can deploy SQL Database, Cosmos DB, and also egress bandwidth. In regards to the free services that are free forever, here's a list. Keep in mind that this always changes. Microsoft is always releasing new services to Azure, and they often make services free and include them on this list. Let's go through some of the popular ones. You can have like Azure App Service where you can have up to 10 of web, mobile, or API apps. Azure Functions is another popular one where you can have up to a million requests per month. So you can get some good use out of that one. DevTest Labs, so if you're running like lab environments and you want some of the capabilities that come with DevTest Labs, that's a free service. If you're running VMs, if you're running other things in those labs, you will be charged for that. But you won't be charged for the DevTest Lab service itself. Also like AKS, so Azure Kubernetes Service. You've probably heard of Kubernetes. This is a managed Kubernetes service from Microsoft. The service itself is free, but keep in mind you will pay for any VMs that are run, any underlying storage that's consumed, you'll pay for those things, but you won't pay for the AKS service itself. Other valuable services are like Security Center. There's a free tier. So as you're deploying things on Azure, you definitely want to turn that on to give you insight into your security posture on your cloud environment. Event Grid, you can get up to 100,000 operations per month. There's some Active Directory B2C, authentications that you can you can leverage. Azure automation is another big one where you can get up to 500 minutes per month and then networking. The networking is usually free, so VNets, subnets, load balancers, that stuff is free, which is great. Keep in mind you'll usually be using VMs with your networking or PaaS services, and so you will be paying for those things, but you won't be paying for the underlying networking itself. Then another big one is App Insights and Azure Monitor. You get up to 1GB free per month of storage, but the services, you can just turn on and start using them. Then the one that I really like is the Azure DevOps where you can get up to five users with private repos with that Azure DevOps.

Let's now cover the factors that can affect your costs in Azure. Location is a big one. Azure is global with regions all over the world, and it does matter where you deploy your resources and what regions you select. Regions are going to have different pricing. For example, North Central US is going to have a lower cost than West US or even East US. South Central US is going to have a lower cost than US East or US West. Part of the reason is there's local factors that are impacting the costs such as staffing in those local areas, in those data centers, connectivity costs, cooling costs, things like that. That is all factored into the actual price that you receive as an Azure customer. The next item is the resource type. Costs are specific to each resource and the usage that the meter tracks, and the amount the meter has associated to the resource will depend on the resource type. So keep that in mind as well as you're consuming and the different types of resources. The next one is service. As your usage rates and building periods can be different if you're using, let's say, an enterprise agreement versus a direct or even a CSP. So, for example, with the enterprise agreement, you will probably have discounted services. As you're consuming services from Azure, you'll be receiving those discounts, so you'll have a lower cost because of your enterprise agreement and what you've negotiated with Microsoft. The last one is egress traffic. Sending data into Azure, aka ingress, also known as inbound, is always free. It's always free to get your data into the cloud. Pulling data out of the cloud, known as egress or outbound, has a cost. Bringing data into the cloud, ingress, is free. Sending data out of the cloud, egress, is going to have a cost. You really need to understand the traffic flow for your environment or the solution that you're running in Azure, and that will help you estimate the cost that you will be paying for any egress traffic. An Azure zone is a geographical grouping of Azure regions for billing purposes. Data transfer pricing is based on the zones. But what are the zones? Zone 1 is the United States, US Government, Europe, Canada, UK, France, Switzerland. Zone 2 is APAC, so it's East Asia, Southeast Asia, Japan, Australia, India, Korea. Zone 3 is Brazil, South Africa, and UAE. Then there's something called DE zone 1, which includes Germany. Keep in mind that these change over time, and Microsoft is adding new regions, new countries, to Azure all the time. So the zones will update and will expand. So make sure you check the Microsoft documentation on these for the latest and greatest.

The first one is reserved instances. Let's say, for example, you know, as an organization, you're going to run X amount of VMs for the year. You just know that. What you can do is you can for those virtual machines and get a discounted rate. It's a really great way to save money and minimize your cost when you know what you're going to consume. The next is Azure cost management. Once you're running workloads in Azure, you could take a look at the Azure cost management tools inside of Azure and start to and see what things are costing you, what resources are costing you, on a monthly basis, on a daily basis. There's a great breakdown. There is also some forecasting that could be done in the Azure cost management so you can predict what it's going to cost you for the next 30 days or into the future. This is a great way to go and analyze your spend and start to minimize your cost and make adjustments. The next is quotas. You can put quotas in place around the resources and the amount of resources that you're using. The next one is spending limits. You can actually put spending limits in place. If you are approaching that spending limit, you won't be able to deploy like more resources, and you can ensure you're not going to go over a budget. There's also Azure Hybrid Benefit. What this is if you have software assurance and you're bringing, let's say, servers from on premises, whether it's Windows servers or maybe even SQL servers up to Azure, you get deep discounts on the licensing for running in Azure. I've seen this change a little bit over time, so make sure you go out to the documentation to understand the latest and greatest with the Azure Hybrid Benefit and/or talk to your Microsoft representative, and they'll be able to give you the insight and the latest information on the Azure Hybrid Benefit and how it can apply to your specific scenario. The final item on this list is tags. When you're deploying resources in Azure, you will want to tag your resources. You can use this as a tool for showback or chargeback in your organization, and it's also useful just to identify and know what things are and what they're for. For example, I usually like to put tags in place to ask the question of when will this expire? Do we need this resource indefinitely? Can we spend this resource down in 60 days? It's good to ask that question, and having a policy for tagging is going to help you enforce that. Also, it's good to identify what things are for. If you have a resource that's costing you a lot of money per month, and it's tagged so you can identify it, identify who the application owner is, maybe who the team is, then you can go talk to that individual and find out do they really need that? Do they know it's costing that much money per month?

There's two calculators that you can use to help you estimate costs. The first one is known as the Azure pricing calculator, and you can use this to estimate cost of Azure products. The second is the total cost of ownership or TCO calculator, and this is good for estimating the cost of migration and the cost of ownership. The Azure pricing calculator is and you can use this to estimate the cost of Azure services and really go down to a granular level. If you're putting together a solution and you want to know how much that solution is going to cost to you, this is a great tool to help you get that cost. You can add services, you can add tiers, and you can get your estimated consumption out of this calculator. For example, let's say you are designing a solution that consists of maybe three web front ends, and you need to run them on Azure App Service. Maybe you need to run SQL Server, and you need to run that on an IaaS VM for whatever reason, and you need some storage, and maybe you need something like Event Hub. Well, you can go and add all of those individual components to this calculator, and then you can see a breakdown of the cost per service, and you can see the breakdown or the summary of the cost of all of these services together, and you can see the cost on a monthly basis and the cost on an annual basis. You also can add different levels of support to the calculator. You can see that right in the cost as well. Then you can save an estimate to come back to it later, or you could even share it with colleagues if you need to. You can export the estimate from a calculation on this calculator to Excel and then work with it from there. It's pretty flexible, and it's really easy to use. The TCO calculator is good when you're planning a migration to Azure from your data center. This is very helpful when you need to see what it's going to cost to run X amount of workloads in Azure over a period of 5 years. This helps you understand and estimate your operations costs or your run costs. You can use that data to compare and contrast with your cost for running the same workloads on premises. This is good to see how much things are going to cost you in Azure for the same type of workloads and help you understand if you have a savings there.

Chapter 10 How to use the Azure Price Calculator

In this demonstration, we're going to go ahead and price an solution using the Azure pricing calculator, and then we're going to go into the total cost of ownership calculator, and we're going to calculate how much it's going to cost us to run this solution over a period of 5 years. We are out on the Microsoft Azure site at Azure Pricing.

This is where it's easy to find both calculators. On the pricing calculator, you'll notice several things. The first thing is you will want to log in with your Windows Live account. You'll know that by looking in the upper corner, you'll see that you're actually logged in. The reason we want to log in is because we're able to save estimates, and were able to share estimates. So if you want to get back to your estimate, you'll need to be logged in so it can be saved to your account. At the top of the pricing calculator, we have several tabs that we could work through.

We have products here, and this is where we would actually add products to our estimate so we could flip through the different products in Azure Services like so. We could actually click on a product here, and it would add it to our estimate. Here we have Example Scenarios and under Example Scenarios, you could create an entire estimate from a pattern that Microsoft has developed and that they have out there on the Microsoft documentation on their docs site.

There's several patterns here, and it doesn't cover all the patterns that Microsoft has created, but there's a lot of them here. So if we wanted to do CI/CD for containers, we could see the solution, which is the pattern.

We could see the products that are involved in that. Then we could go ahead and add this to an estimate. What that will do is create a brand new estimate, and it will actually add all of these products to that estimate. Then you could go and tweak the cost for each of those products for your specific needs. Then you would have an estimate that goes along with this pattern. You could also go and click on this and that would bring you out to the Microsoft docs site where it has full information about this specific pattern. We could also click on Saved Estimates too and this will show any estimates that you have that you've done in the past that you saved. You could actually open them from here, you could delete them or you can export them. You could even copy and then work from there and then there's an FAQs tab that just kind of gives you information on how to use the calculator. You can add the amount of hours you will require and it's going to reflect the actual cost. You can change some other things too, like you could say, you could go to months, you could go to days as far as how often you think you're going to be running this service in Azure. You can also do some things like change the tier, and that's also going to impact the price. You really need to understand the services that you're planning to use and then come in here and modify accordingly and tweak so that you can get the accurate pricing. You'll have different options for different services and products that you have added. We could also change the operating system. For example when I switch the requirements to Linux look the pricing will decrease because there is no licensing cost. So keep in mind if you're going to run Windows servers on Azure in your environment, you will want to look at the hybrid benefit to see if you can save some money there or even your enterprise agreement. Then we can change the instance types. So you see all of the different virtual machine types, so you could change that. You could make it a bigger one, and it's going to increase the cost. Then under OS, we could say that we have the hybrid benefit, and that brings the cost down as well.

Or if you don't have a license for this server yet, you can say license included, and then when you deploy that server, you get the license with your Azure subscription. You also have an option of reserved instances. You could also configure some other options, like managed disks and etc. So in our solution, we have virtual machines, and then we have App Service. App Service is actually PaaS, so Platform as a Service. And let's say we wanted to add that because we're running some web servers and we're running SQL Server so we want to keep that server in this solution. We're going to have three instances of the App Service because we're running three servers on premises, and we want to continue to have three web servers so we can do load balancing. You have some options to configure this App Service, which will impact your pricing. And you have different options, so things like SSL and IP for the App Service that you don't have with the virtual machines. If we keep going, I have virtual network added here. We can calculate our inbound and outbound, so our egress and our ingress. I've also added an application gateway with WAF, or web application firewall.

That is just to protect my traffic coming into these app services, or my web servers. It's there to protect it, and there's a cost with this as well. You can go ahead and estimate this is well, and there's options to configure that. As you tweak and move the dials up and down, you'll impact the hours. If we keep going, we have some storage accounts.

Then as we keep moving down to the bottom, we have support added.

With support, we have the option to check different levels of support. I have it on Professional Direct but look at what happens to the price as I change it to Developer.

So it decreased the price by a lot. We've gone over the pricing and support options, so keep in mind the proper level of support that you need for your solution that you're going to deploy on Azure. At the very bottom of your estimate, you have several options.

You can export this out to Excel or you could save this. When you save this, it will save it to the Azure site under your account and that's why it's important to make sure that you're logged in. You could also do a save as if you want to change the name instead of having a generic name. Then you can also share the link. What this does, is that it generates a custom link for this estimate, and you could go ahead and share this with the rest of your IT team or someone else if they need to look at the cost, and they would be able to come in and adjust the resources as well so that the cost could go up or go down and you could collaborate on a calculation on an estimate. Then you have some options to display SKUs and/or display resources which you could turn on or off.

Another part of adopting Azure is to understand all of the available support options and channels so in the following chapters we will explore the various support channels and leave with an understanding of where to go for help when needed. First, we'll look at understanding the support plans that are available in Azure. Then, we'll understand how to open a support ticket and what that process looks like. Then we'll look at the available support channels that are outside of the standard support plan channels in the Azure portal. Then we'll take a look at the Azure Knowledge Center and how to use that. The first one we have to cover is the basic support plan. This comes with all Azure subscriptions, and it's a basic support plan that includes the following: 24/7 access to billing and subscription support, so that's billing and subscription support. It doesn't necessarily mean you get support if you're having issues with a virtual network or a network security group or some other service. It does include online Azure products and services, documentation and whitepapers around those, and support forums. Other Azure support plans include the following. There's a developer plan. There's a standard plan. There's the professional direct. Then finally, there's the premier. What are the details of all of these plans? These consist of the following. The developer plan is really for From a tech support perspective, it's email only and it goes up to a severity C, and you can get a response within 8 business hours. From an architecture standpoint, Microsoft will give you some general guidance. The standard plan is really meant for production subscriptions, so if you're running production workloads, you'll want to have the standard plan at a minimum. This gives you tech support access 24 x 7 via email or phone. This can go anywhere from severity C up to severity A, and you can get a response within a timeframe. From an architecture standpoint, Microsoft will give you some guidance based on best practices. From an operation standpoint, they'll give you some support for onboarding services, service reviews, Azure Advisor, consultations, and such. From a training perspective, you'll have access to some web seminars that are led by Azure engineering. Then you do get some proactive guidance with this as well from a ProDirect delivery manager. The next one up is the professional direct, and this is for those workloads or subscriptions in Azure. The tech support and the response times are going to be identical to what you get with standard - same thing with the architecture, operations, training, and the proactive guidance. With the premier, that's kind of the top tier that you have available for you, and this gives you some extra things. The tech support and the response times are going to look the same. When you get to architecture, you'll have architectural guidance. Microsoft will actually help you with your design. They'll help you with some reviews there and give you tips, best practices. They'll help you with some performance tuning if you need it around certain services. Think about like databases, web apps, etc., and some help with configuration. From an operations perspective, there's a TAM, that's a technical account manager, that's dedicated to you and will help lead service reviews and if you have support tickets in and you need to know what's happening with those, you can reach out to that dedicated person for those things. The TAM is very, very helpful is what I've found. From a training perspective, you've got access to the online web seminars, but you also have access to some training. Then from the proactive guidance, you also have the ProDirect delivery manager, but you also have your TAM that you can reach out to for help there if you need some guidance there. Then you also have access to launch support. That's for an additional fee though. That covers the comparison and the details of all the different plans that you have available with your Azure subscription. But how do you open a ticket in Azure? The first step is you need to go to the Azure portal. The second step is you need to click on the help and support. The third step is click on a new support request. When you click on the support request, you will be presented with a form, and go ahead and fill out the form with the details like what is the service that you're having an issue with or you need help with, what are the details, what's the severity level of the support request. Once you have that all completed, you go ahead and submit it, and then Microsoft will get back to you within the designated response time. It's pretty straightforward.

Now let's talk about the different support channels that are available outside of the support plan, the typical standard support plan. The first few are actually forums. You have the MSDN forum, you have the StackOverflow forum, and you also have the Serverfault forum. You will find Microsoft MVPs out on these forums, answering questions, other community folks, and you may find Microsoft employees from time to time out on these forums answering questions as well. So they are a really great resource. But keep in mind, there's no official SLA around response times, but they're still very, very helpful. With the StackOverflow, that's more geared towards developers. And the Serverfault is more geared towards the IT pro. And MSDN is kind of a mix of all. The last one here is Azure Support on Twitter. So you can actually tweet @AzureSupport, and Microsoft will respond to you. I've found this to be very, very useful as well, and they're very responsive. Again, there's no guaranteed response time or SLA on this. It's just kind of like you tweet at this, and Microsoft will get back to you. These are the support channels that are outside of the standard support channels that you can leverage when you're looking for support on Azure. Let's now look into the Knowledge Center. This is an online website where you can go for general questions and get those answers and get them quickly. The knowledge is comprised from community, Azure experts, developers, and customers all brought into a single place. You can search in this Knowledge Center, you can browse, or you can filter by products. You can scope it down to what you need to get to quickly. So if you're looking for something around networking or maybe DNS in Azure or websites or whatever, you can filter down and get to the answers very quickly and this will link out to documentation or other resources. The Knowledge Center is very helpful and a great resource for you to use as you're looking for support around your Azure subscription in your Azure environment.

Chapter 13 How to open a Support Ticket on Azure Knowledge Center

In this demo we're going to cover how to search the Knowledge Center, and how to open a support ticket within the Azure portal. You can get to the Azure Knowledge Center website on the URL;

From here, you can scroll down and you can see the Popular questions, and you can click on any one of these.

What I want to do is target a specific scenario just to kind of give you an idea on how this works. I could search, but what I'm going to do is use the filter here, and I'm going to click on Browse and then I want to check on something in regards to networking.

I'm going to click on Networking, and then it gives me this additional flyout where I want to click on Virtual Network. When I click on Virtual Network, it gives me options at the bottom of the screen but I'm going to scroll down again and you'll see the options here, and maybe you don't see what you're looking for yet, so you can click on the View more button, and that's going to expand.

What I'm actually looking for is that I need to increase the number of network security groups in my subscription. When I click on that, that's going to bring me to another page, and from here what it gives me is this Learn More, and if I click on this, I'll open it in another tab, this is going to link me out to the Microsoft documentation on how to do this and give me more information on my network security group limits and my subscription.

On the documentation and I can scroll through and you would see the different limits for your network security groups and you would be able to go from there and get the information you need.

Back in the Azure portal, I went ahead and I clicked on Help + support, and it brought me to the dashboard.

I have a few options here. I can look at Support Plans, Service Health, Advisor, All support request that maybe I have something that I've already submitted and I want to see what the status of those are. But what I want to do here in this scenario is just open a new support request, so we'll go ahead and click on that, and it's going to go ahead and open up and ask what's my issue type?

What I want to do is increase the limit of network security groups on my subscription. That's my goal. So I need to go ahead and select a subscription here, and the quota type and then I'm going to click Next on Solutions and this will give us our details that we need to fill out for the form.

It's a severity C, so it’s not a big deal. I'm going to do email for my contact, and I'm going to leave everything default there. If you needed to describe the issue if you were putting in a support ticket for something else, you would be able to describe the details there. I'm actually going to select the resource that I want to increase the limit on. We're looking for Network Security Groups, so we select that and we can see the current limit is 5000, so let's just say we want to request this to increase to 7000.

We're going to save this and continue, and then we'll go ahead and click on Next for the review and create.

Here is a summary of what you're going to submit with this ticket, and then you would go ahead and click Create to go ahead and submit the ticket.

Chapter 14 Azure Service Level Agreements

A part of planning and supporting cloud environments is understanding the available service level agreements and it's important to understand the SLAs and how they apply to the cloud services that you will consume. So, in this chapter, we're going to take you through the ins and outs of SLAs and how they apply in Azure specifically. Let's look at the topics we're going to cover specifically. The first topic is we're going to dive into what an SLA is and make sure we have a clear understanding of SLAs. Then, we're going to talk about something called composite SLAs and how they work and how you would figure out a composite SLA for a solution that you have in Azure. Then finally, we'll finish off with understanding how you can determine the appropriate SLA for the services you plan to consume and/or the solution that you plan to spin up in Azure. SLA stands for service level agreement, and a service level agreement really comes from ITIL, which is a framework for managing IT that's been around for quite some time. An SLA is a commitment between a service provider, so think of Azure, think of Microsoft as that service provider, and its internal or external customers. You, as someone that's consuming cloud services, would be that external customer. Microsoft is the service provider, and they're providing you an SLA as an external customer. The SLA itself outlines what the service provider, in this case Microsoft, will provide to its customers and the standards that the provider will meet. Azure SLAs are a little bit different, so let's talk about this. In the context of Azure, an SLA is going to detail the commitments for uptime and connectivity, and this is what Microsoft is going to guarantee around its various services. The different services within Azure are going to have different SLAs, and so it's important for you to know what the SLAs are of those services before you just go ahead and consume it and to make sure that those SLAs fit with your business and meet the needs of the business. Before you're consuming a service within Azure, make sure you go out and look at the SLAs that are available and what they are and what the connectivity and uptime commitments are for Microsoft around those services and make sure they align with your business needs. Now let's talk about a composite SLA. What is a composite SLA? Well, in Azure, this is when it involves more than one service that's really supporting an application. It's rare that you will have one service that you're consuming in Azure, and you need that SLA around that service. Chances are in cloud, in Azure, you're going to have multiple services that you're consuming from Azure that are powering an application. This is where the composite SLA comes into play because each of those services are going to have their own SLA, and those together make up the composite SLA that's supporting that application. This is going to be centered around availability and connectivity of those services. Let's look a little bit more at a composite SLA, and let's look at an example here. So let's say, for example, a company is planning to have a web app as the front end of an application and planning to have some databases or a database as the back end of an application. Within Azure, we'll have a web app, and it has 99.95% for it's SLA. That's what's guaranteed from Microsoft around that service. That would be a web app running on Azure App Service. For the database, let's just go with a SQL database, and this is managed SQL, so it's PaaS SQL database, and that's going to have 99.99% for the SLA. What you have to do for the composite SLA is basically calculate these two together to come up with the composite SLA percentage, and this is what that would look like; your web app and your database together. So we have 99.95% guaranteed for the SLA for the web app and 99.99% for the database. We multiply those two together, and we get 99.94% for the composite SLA. For your business needs, that 99.94% SLA might be good enough. Maybe it's not. There are methods to increase that SLA. You can do things like replication with the SQL back end. You could look at replicating the front end. That all goes into the architecture and the design and the way you design your application, and that will enhance or modify that SLA to meet those business needs.

Now let's talk about things you should look at to help you determine the appropriate SLA for your business. Let's kind of start from the bottom up because we just got done talking about a composite SLA. What you want to do is look at the included services and their SLAs and how that's going to make up the entire overall SLA of the application that you're planning to run. As your architecting or designing an application, you'll map out what services you're planning to use. You need to go look up those SLAs. Then just like we just went through, you start to calculate, what is the composite SLA, and does that really meet my business needs or not? The next thing is you want to look at the availability metrics, so your mean time to recover, your mean time between failures, and we have the definition of what those are there. Just make sure that you understand what those are against the services and make sure those meet your business needs. Same thing with your recovery metrics; those are critical as well, so your RTO and your RPO. So your RTO, your recovery time objective, and that's what is the maximum that's acceptable, for an application to be unavailable after an incident. Then your RPO, the duration of data that you can afford to lose if there's a disaster. You want to take all of this into consideration when you're designing your application and when you're looking at services within Azure, and you're looking at the SLAs. You want to look at the SLAs and see if the SLAs meet your objectives and your goals of that application and the business needs. If they don't, what adjustments do you need to maybe make in regards to replication or disaster recovery technologies there to help meet that? The next one is dependencies. It's critical to understand all internal and external dependencies of the application, and then you can understand any SLAs that are around any of those dependencies. Chances are your dependencies are all going to live in Azure. You have the URL to go look up the SLAs, but there may be dependencies on other services that are outside of Azure, and it's important to understand those SLAs as well. You can take the same steps that you took to calculate the composite SLA even with external dependencies and use that formula to figure out the composite SLA with external dependencies as well. The last one is really critical, and it's cost and complexity. We can get better SLAs if we need to, but chances are that's going to come at a cost. Is the better SLA really worth the increased costs that you may incur by increasing that SLA? Or does it make sense to go with what you get out of the box from Azure, and can the business live with that? This warrants really understanding the SLAs that are there around the services, understanding the composite SLA, understanding the business needs, and understanding the cost. You may need to go have a conversation with the business to understand if the SLAs are good enough, and maybe they are. Maybe they meet what has already been determined in regards to SLA needs. If not, go have that cost and find out if the business is willing to pay a higher cost to get increased SLAs.

Chapter 16 Azure Service Lifecycle

Azure is constantly changing at a very, very fast rate. It's important to be able to identify what features and services are and plan around the future roadmap of Azure. In this chapter, we're going to dive into how you do that, how you plan for the services and the feature updates. We're going to talk about preview for public and private, as well as general availability, and what all of that means and how you can stay up to date with services in Azure and how you can know when services are ready for production use. First we are going to cover public and private preview features, what that is, what that means, how you can identify those. Then we are going to look at understanding the term general availability and then, understanding how to monitor feature updates and product changes. But what are Azure previews? Microsoft has Azure feature previews, and these previews are for eval features, products, services in Azure are still in a beta or stage. This is when you can get early access to the services that Microsoft is still working on, and you can start to provide feedback. You can start to consume the service, try it out, see if it's going to fit for the use for what you need, and really just get that early access. These previews are also a way for Microsoft to get services to customers early on and start to get feedback so they can improve the products as they're still developing on those products and start to interact with the customers to make sure the use cases for the services in Azure are good to go. What is public and private preview? Well, Public is when any customer that's consuming Azure can go out and sign up for the preview of a service and start using that service and start getting that feedback to Microsoft and just start using and consuming that service. Keep in mind that with a public preview, you may not have SLAs around the service, and you may not have official support around the service. Also, you may see things changing rapidly in the UI or the way that service works, and you might see some bugs here and there. Private preview is like a public preview in that it's a way for Microsoft to get early access to you to a service, but this is usually by invite only, or you might have a form in the Azure portal or outside of Azure that you have to sign up for to get access to the preview of that service or product feature, etc. An example I can give here is that the Azure Kubernetes Service has been out for a while, but it's rapidly changing, and Microsoft is rapidly adding updates to that service. Some of the updates you've been able to get access to through private preview, like when Microsoft started working on the Azure policies for the Azure Kubernetes Service, you couldn't just go into the portal and sign up for that, and it would be turned on, and it would start working. You had to fill out a form and then get approval on the back end to start using the Azure Policy services for that Azure Kubernetes Service. So the public preview is available to anyone, where the private preview is limited and you have to sign up for it whether it's directly or whether you have a link to a form from Microsoft. Now let's talk about access to the public previews. Within the Azure portal, there's really two ways where you can get to the public previews. You can create a resource, or you can go to All Services. You will see a label on the service preview, and that's how you'll know that the service is in a preview state. Then you can go ahead and just click on the service, go ahead and create the service and start consuming it. Or if you go to All Services, you'll just see it and you'll know that it's in preview, and you might already be using it, or you could create it from there and start to consume it as well. The term general availability - what does this mean? Well, after Microsoft is done with the preview stage of a service or product feature, once they've had it out there in the market for a while with Azure customers, it's been successfully tested, it's finally released to Azure customers, generally it becomes a part of Azure's default product set. This is when it moves to becoming generally available or often referred to you as GA. And when a service goes into a GA state, it will basically have SLAs on it. It will officially be supported so you could call tech support and get help with that service. You won't have to work with a special group that's actually working on that service. And it will be ready for production use. But how do we monitor feature updates and product changes in Azure? There's three main areas to do this. The first one is the What's New page in the Azure portal. The second one is the official site for important updates, roadmap, and announcements to Azure products. This is great because it has the roadmap there as well. So you can go out there, and if you want to see what Microsoft is planning for the future, what's coming to Azure even before it hits a preview state, you can go to this updates site. Then the last one is the official announcements from the Azure blog. As Microsoft releases stuff to general availability or even if they announce something new in a preview or a feature add to a service in preview, they'll announce it on this Azure blog as well. This is just generally a good place to follow for good information around Azure, and you'll find other Azure blogs out there under the azure.microsoft.com/language of your company/blog. I highly recommend that you go out and you follow this blog, subscribe to it. because it has a lot of good updates. These are the three main areas where you can go and get updates around what's coming in Azure. Now, we're going to take you through some of the links so you can see preview services in Azure, and you can learn about upcoming previews and updates to Azure. We're in the Azure portal, and we're on All Services and I just wanted to show you that you can scroll through here and you can see the services that are in preview.

If we keep scrolling down, you can easily identify the services within Azure that are in preview and the ones that are not. So we can see Mesh applications there is in preview. We can also see SAP HANA on Azure is in preview and then Azure Spring Cloud.

It's really easy to go through here and identify the services that are in preview. We're on the What's New page in the Azure portal now, and you can see updates around all Azure services from this What's New page.

You can click on the link for any of these updates, and it's going to bring you out to the Microsoft update site for Azure. Also, while you're in here, you can change the timespan for these updates. You can see what's new in the last month, what's new in the last 6 hours. You also can change the tag here to scope it down to a specific service that you want to see what's new about this service.

So we could scope this down to Azure Monitor, click Apply here, and we'll see the latest update there for this service. A couple of other things to note here is you could go to the Azure blogs, so you could see updates right here, or you could see updates from the Azure blogs and link right out to the latest blogs. And you could go to Videos here and see updates from here as well. Now we're on the Azure updates site, and on this site, you'll see that you can search for the service that you care about.

You could type in Azure Monitor or any other service, like Network, and then it will give you the list of updates and preview features for that service. You also could just do the browse, and you could scope it down to get a narrowed list. When you scroll through here, you'll see links, and you can click on the link to get to the specific page for that update with more information around that update. It's very useful and good information to understand what things are in preview, what updates are coming, what's happening with Azure overall. Back on the homepage of the Azure updates site, you also can scope things down just to preview services by checking the box and go ahead and filter the results. Or you could see things that are in a GA status by clicking on Now Available and filter the results. Then you would scroll down and you would only see services that are in GA, or, in comparison, you could do the same thing for the preview. This is a good way to come and quickly see what's what from a GA and a preview status on the Azure updates page. The last site here is the Azure announcements blog.

Again, you can see that Microsoft is making full blogs about announcements. Here's one that's the general availability of Azure files on premises with Active Directory Domain Services authentication. If you click on that, it just links out to a full blog with a lot of information about the release of this service and links to getting started, who to contact. It links out to the Azure Storage forum. So it’s very useful and a great way to understand what Microsoft's done with services or doing with services. They'll often write blogs when they're announcing previews, and you'll have links out there to get in contact with Microsoft or go to forums or different special places to understand more about that service.

BOOK 4

MICROSOFT AZURE

AZ-900 EXAM PREPARATION GUIDE

HOW TO PREPARE, REGISTER

AND PASS YOUR EXAM

RICHIE MILLER

Chapter 1 How to Register for the AZ-900 Exam

In the following chapters, we're going to learn how to register for the Certification exam. We will first learn how to register for the Certification exam, learn about the different certification providers, as well as the different steps in the registration process. We will then look at the flow and really some tips and tricks whether you decide to take your exam at a testing center or take your exam at home or in your office. By the end, you will be able to register for the exam, as well as have an overall idea on how the examination process will work. Let's start by learning how to register for the Certification exam. First things first, in order to register, you need to have a Microsoft account. It's very important that if you have done a Microsoft certification before, make sure to use the same account all the time. If not, you'll have two different MCP, or Microsoft Certified Professional, IDs and your transcript will be split. To be honest, it's a bit of a mess to actually merge them afterwards. It's doable through support if it ever happens by accident, but let's hope we don't have to use the support. So try to always use the same Microsoft account every single time. If this is your first ever Microsoft certification, make sure to choose a Microsoft account that you want to use for the long term, and you will create your certification profile. As a tip, make sure to enter your name exactly as it appears on your government issued identification. For example, a lot of us have middle names or really names we don't use all the time, so make sure your certification profile matches exactly what you have on your ID. There are two exam providers for the exam. The first one is Pearson VUE, and the second one is Certiport, which is for students or instructors. Certiport is actually a Pearson VUE business, but it's still shown as two different options on the site. Even if you're a student, unless you have a very specific reason to go use Certiport, such as a class requirement, I would personally go with Pearson VUE since all of the other exams above Associate level, so Associate, Expert, Specialty, are only provided by Pearson VUE. This way, if you decide to pursue your certification journey further, you will already have the experience with the certification provider and it will be one less thing to worry about rather than switching providers between your Fundamental exam and then your Associate exams and higher. To start the registration, you need to go to the exam page, and you will have the schedule exam option with either Pearson VUE or Certiport. So, let's go through the process. First, it will ask you to sign with your Microsoft account, and if you have done certifications before, you will need to validate your certification profile.

If this is your first certification, you will need to fill it up, but it should only take you 2 to 3 minutes. You will see your email, as well as your MCP ID so you can make sure that you're logged in with the right account. If you have participated in various Microsoft promotions or conferences, you might have a discount on your exam, which will be presented to you on the Exam Discounts tab. After you validate your information and choose a discount if you have any, we need to choose where will we take the exam. The two choices are either at a test center or online at your home or office.

Depending on what option you choose, you will actually be presented right away with some extra interesting information and links to resources. Remember that even if you take the exam at home, it's still a proctored exam, so a proctor will be watching you on camera and hearing everything through your microphone, so whether you take it at a test center or at your home or office, still the same security standards apply. The next step will be to select the exam language.

As the is a very popular exam, it actually exists in multiple languages, but not all Microsoft certifications have as many options. If you have selected to do it in person, the next step will be to choose the location where you want to do the exam. You will be presented with choices based on the address in your certification profile, and you can also choose multiple test centers if you want to compare availability. Next, you will need to select the date, as well as the time you want to do your exam at. Make sure you check your calendar and block that time off. This way, nobody schedules meetings or anything during that time and I would block off extra time before and after because you really don't want anything, a meeting, a presentation, to stress you while you take the exam. Something to consider as you select your date and time is if you need an accommodation, both Pearson VUE and Certiport can provide appropriate arrangements for individuals that demonstrate a documented need. It can be things such as extra testing time, a separate testing room, or even breaks. Special accommodations can take up to 10 business days to be approved, so if you need to request a special accommodation, try to schedule your exam a bit more in advance. This way, you can go through the request process without being stressed for time. After you've selected everything, our next step is to review and confirm the exam in your cart. It will show us the exam name, language, the appointment location, as well as the price, and finally, we have to put our credit card number in and pay for the exam.

After this, your exam registration is complete. Before we finish off the registration part, I just want to share a quick reminder to verify the reschedule and cancellation policies in case you need it. Traditionally, it has been six business days before the exam date in order to change or cancel without paying a fee; however, make sure to double check it. This information will be available in the confirmation email, as well as on Microsoft Docs.

Chapter 2 How to Take your Exam at the Testing Center

Now that we know how to register, let's look at how it works when we take an exam at a testing center. First of all, try to arrive early. You have probably heard this about every important appointment in life, but planning to arrive early will make you so much less stressed in case there's traffic, the subway breaks down, or anything else. An exam is stressful enough, we don't want to add arriving late to the list of things that can stress us out before the exam. Very important: don't forget to bring the required piece of ID with you. According to my last exam confirmation, you need to bring one valid, unexpired, government issued ID with a signature and a photo and the name must match the name on the registration exactly. Always double check your confirmation email for the requirements. I always bring two pieces of ID with me just in case. Also, don't forget to go to the restroom before starting the exam. Microsoft exams do not have scheduled breaks, and while I'm sure that the proctors can find a way if needed, let's not add more stress to our exam. Let’s now look at the overall process and how it works. First, you will arrive at the testing center a bit early and at the reception, give them your name, as well as your appointment details. If they ask you for your exam sponsor or provider, just say Microsoft as Pearson VUE does exams for a lot of different organizations and sometimes they have a lot of different sheets for different exam providers and by knowing to look in the Microsoft pile, it can help them find your appointment a lot easier. They will ask for your ID, as well as to assign the exam rules, and they will probably mark the time on there. After that, the procedure can vary a bit by testing center but they generally take a picture of you and then they will compare it with the last picture of you taking an exam so they can make sure that you're the same person. Some other testing centers might use different forms of biometrics, but just so you know, there might be an extra form of identification depending on the testing center. After that, you will be provided a locker for all of your personal effects, and you have to put everything in there, and don't forget to turn off your cell phone. I generally like to fully turn it off and not just put it on vibration or silent mode to make sure it doesn't ring or annoy other people as I get a ton of notifications. Remember, you cannot bring anything with you in the exam room except something to take notes on, generally like an erasable whiteboard sheet that the testing center will provide to you. So you don't have to bring anything, the testing center will provide you everything that you are allowed to take in the examination room. After that, you will take the exam and receive your score at the end. Finally, as you leave, make sure you don't forget anything in the locker or your coat before going back home. Adrenaline is generally high after taking the exam and we often forget things. Something to remember is that one of the advantages of doing an exam in person is that you have plenty of people working at the testing center who you can ask for questions. They are nice people, and don't hesitate to tell them this is your first exam if it is so they can really help explain it to you step by step.

Chapter 3 How to Take your Exam at Home

Now that we know what it's like to take an exam at the testing center, what about taking an exam at home? First of all, a few days before the exam I highly encourage you to go perform a system test on the device you intend to take the exam with. This way, if anything is not working, you have a few days to find another machine. Before you actually start the exam, you need to clean up your desk space and room. You need to make absolutely sure there are no papers, electronics, or anything within arm's length of your device. Make sure that you tell your family or coworkers to not interrupt you during the exam. This is very important as if the proctor hears anything or sees anybody else in your camera, it's an automatic failure. Something I didn't realize the first time I took an exam at home, make sure you have your smartphone and a piece of ID nearby because you will need them during the process. When I first did an exam online, I had my phone turned off and in another room, and then during the process it asked me to use my phone to submit a piece of my ID. So it took me a bit of extra time to go get the phone, turn it on, so make sure you have them by you. Also, make sure to close all of the apps and unnecessary services on your machine. The testing system will provide the proctor with a list of everything running, so try to close everything in advance. It makes the process a lot smoother. The last thing is that if you're doing the exam on a laptop, make sure to plug in your laptop or at least have a full battery. To share with you a bit of what has worked for me personally, I prefer to do my exams on the dining table instead of my office. I have too many gadgets in my office and things everywhere, so doing the exam at the dining table means there is less things for me to clean up, and also I don't have to worry about things such as the whiteboard as there is a lot more space around me on the dining table. So I have less things for the proctor to check out before the exam. Also from a device perspective, I use an old laptop with a base Windows install. This way, there's less apps, notifications, services, and things to worry about during the exam. This is not possible for everyone, but I just wanted to share with you what has worked for me personally. Now that we have our preparations, you can actually begin online check in and system check for the exam 30 minutes before the exam start time. This gives you more time in case anything doesn't work with your system and allows you to go through the process more calmly knowing that you've got the time. If everything goes smoothly, you'll probably be able to start your exam even before the scheduled time. As soon as your system checks are ready, you'll be put in a queue and when a proctor is ready you will be able to start the exam. But where do we start the exam from? You need to go to the Microsoft Learning dashboard, and from there you will see your appointments and, if you have an online exam, you'll have a button that says Start online exam, right there. If we take a look at an overview of how it works, at a very view, you will first start the exam from the Learning dashboard, and then it will ask you to download the OnVUE proctored exam app in which you will first start by doing a system test. If everything is good, you will provide your phone number where you will get a text message with a link where you can submit your information. You'll be asked to take pictures of you, your ID, as well as your surroundings. After all of that is submitted, you will be connected with a proctor, and depending on the proctor you get, they might talk to you through the device speakers or they might write to you on the chat. Some of them are okay with only the pictures you provided, some of them might ask you to move your device camera around as they might want to see some more things in detail. After all of those checks by the proctor are good, you're ready to take the exam. Remember, the proctor will always see you and hear everything through your device camera and microphone, and you are not allowed to mute it, stop it, block the camera. If not, it's an automatic disqualification. Pearson VUE also has their own nice preparation video on how to take an exam at home and it's about 5 minutes long, so if you want to watch it, it's a nice introduction video if you are taking the exam for the first time. In summary, we have first covered how to register for the exam, the different exam providers, either Pearson VUE or Certiport, as well as how the registration process works. We have then looked at the process, as well as tips and tricks whether you decide to take your exam at a testing center or take your exam at home. Next, we will learn about the exam structure and question types.

Chapter 4 Azure AZ-900 Exam Structure

Now, we're going to talk about the exam structure and question types. We will first talk about the basics of the such as how many questions you should expect, how much time you have, and the overall structure of the exam. We will then talk about the different question types you can expect in the exam. Let's first cover the basics of the exam. Microsoft does not share the exact information about each exam, like other certification authorities might do. Also the number of questions for the same exam might even be different from person to person again on the same exam. Microsoft does provide guidelines for each exam category, which is what I will share, but don't worry if it's not exactly the same when you take the exam. With that in mind, for a exam such as the you should expect to have between 45 and 60 minutes to do the exam, but in your schedule, plan for around 80 minutes of time, which includes the time to set up, sign the agreement, comment the questions, so make sure you block 80 minutes on your calendar in total, but only for the questions or exam part, you'll have somewhere between 45 and 60 minutes. For the number of questions, you should expect anywhere from 40 to 60 questions for this exam. But isn't this almost saying like I only have 1 minute to answer each question, and the answer is yes, for fundamentals exams, you will have somewhere between a minute and a minute and a half for each question, if we look at it from a purely mathematical perspective; however, you should expect short questions. We will talk about the types of questions later more in detail, but really for fundamental exams, I like to describe them as fairly straightforward questions, whether if you know to answer, you should be able to answer it in no time. I do, however, understand that taking an exam is stressful, and adding a time constraint adds even more stress, so my advice is that Microsoft actually offers a feature allowing you to mark questions for review later. So if you read a question, and you're not sure about the answer, you can mark it for review later. This way, you quickly answer all of the questions that you know you've got the answer for, and at the end, you can use the time left to spend more time thinking at the questions you're not 100% sure about. To give you an idea of what it looks like, at the top of the screen, you will have two options, Review later or Comment later.

Review later is for yourself to tell yourself that at the end, come back and recheck this question, and the commenting is for after the exam, if you want to give feedback to Microsoft that maybe there's a typo in the question, or that it doesn't make sense, or maybe that is outdated. The commenting is not included in the exam time, but it's included in the 80 minutes that we have talked about earlier. After you get to the end of the exam, you will get a review screen like this one where you will see how many questions you have answered and how many you didn't.

You will also see all of the questions you marked for review and for comment, as well as the time remaining. You can then click on the tiles to see the ones you marked for review, and then now did you know how much time you've got left and how many you marked for review, you can really more calmly review them so you can feel 100% confident in your answers. Before we end this chapter, here is the general exam flow that you should be prepared for. After you check in for the exam and all of the prerequisites are done, the first thing you will have to do is read the instructions and accept the Microsoft nondisclosure agreement in regard to certifications. You can also read it before the exam so you have one listing to worry about reading on the exam day. You will also probably have some optional questions about your level of proficiency around the different exam sections. This is for Microsoft to better balance the exams in the future and has no impact in the level of difficulty of the questions you will actually get. After that, it's exam time where you will have all of the different questions for the exam, and after you're done with the exam, you'll have dedicated time to provide comments to Microsoft on questions. This is optional, and it's for you to give your feedback if you found questions were confusing, outdated, had typos, and so on. Again, this is optional. You don't have to do it if you don't want to, but if you see something that's wrong, please do it because it can help Microsoft fix any mistakes, and by providing comments, we make sure the certification experience is better for everyone.

Chapter 5 AZ-900 Exam Question Types

Now that we know the overall structure, let's talk about the types of questions you should expect in the exam. Microsoft will not specifically say which types of questions you will get on a specific exam, but they do share the types of questions for all exams. There are actually 10 types of questions you can expect on any Microsoft exam. They are called active screen, best answer, build list, case studies, drag and drop, hot area, multiple choice, repeated answer choices, short answers, and labs. Given the time constraints, you can really rule out some of them, such as short answer, labs, case studies, as those, in theory, should not show up on the Fundamentals exam. But Microsoft can never confirm the types of questions you can expect, so better be prepared. Let's take a look at what those types of questions actually look like. The one that you're probably the most familiar with is the best answer. You have a question, a list of answers, and you can only select one answer. A very similar one, but slightly different is the multiple choice where you have a question and multiple answers, and you need to select all of them that apply. Microsoft has been fairly good at giving a bit more information on those questions, such as select the three answers that apply and not simply all that apply, but that is not a guarantee. The next one, which is very popular, is the drag and drop. You have a question, you have possible answers on the left, and then you need to drag and drop them to the answer areas on the right. You can often expect questions that are scenario based and they would question you on what Azure products would fit that particular use case. Most times, you will actually have more answer options on the left than possibilities on the right. So you might have like say five possible answers, but only three drag and drop spots, so it doesn't mean that everything on the left will actually be used. Another very popular one is the build list type of question in which you are given a task, and then you need to put the actions from the left in the correct order on the right. This is a bit more complicated because not only do you need to know the appropriate choices because not all of the options on the left are needed, but also put them in the right order. Microsoft does offer some amazing short videos for each type of question and what I like about their videos is that they are filmed in the actual exam interface, so it can give you a good idea on what the actual interface looks like.

Once you go to the page, you'll see a bunch of short videos for each exam in the actual interface. It will either play in line, or you can go to YouTube to watch it in full screen. In summary we have first looked at the exam basics. You have between 45 and 60 minutes to do the exam, but you should plan for about 80 minutes, which includes the NDA, instructions, and comments at the end. You should also expect anywhere between 40 to 60 questions for this exam. To better manage time, if you're not sure about a question, use the Review later option. This way, you can answer all of the questions that you know first and at the end spend the time you have left on those that you are not sure about and maximize the time you have. We have also looked at plenty of question types, such as best answer, multiple choice, drag and drop, build list, and you can go watch short videos for all of the rest of them. Next, we will coever what happens after the exam.

Chapter 6 What Happens After the Exam

Now we will cover about what happens after the exam. There are actually two things that can happen after the exam. You pass the exam or you fail the exam, which can happen as well, and it's okay. I have failed a few Microsoft certifications as well, so we have to talk about that option as well. If you pass the exam, we will learn about the different digital assets that you get and the different steps that you can take afterwards. We will also cover what happens if you fail the exam, who you can see and what are required before you can reattempt this exam. As soon as you click the end exam button, you'll get your score right away. The max score you can get is 1000, but the passing score, or the minimum you need to pass, is 700. A pass is a pass, meaning that only you can see the score on the official transcript. You will not see the score, only if it's a pass. So whether you pass with 701 or 1000, a pass is a pass. If you did your exam at a testing center, you will get a printed version of the score report right away, which you'll not get if you do your exam at home, but you could always print your own score report if you wanted to. Something that is really interesting, the score report will show you how well you did for each exam objective. I personally find the Performance by exam section to be very important. Whether you pass or fail the exam, it will show you the areas that you did great or not so great in. It's important to remember that not each section of the exam will get an equal number of questions. The score report will cover things such as Understand Cloud Concepts, Understand Core Azure Services, Understand Security, Privacy, Compliance and Trust and Understand Azure pricing and Support.

But you won't be able to directly find out how many questions you got wrong.

Chapter 7 What if You Pass the Exam

Once you got your score report, let's take a look at what happens if you passed the exam. Once you pass the exam, you'll get three types of digital assets to help you promote your achievement. First of all, you will get a Credly badge, which you might know under the name Your Acclaim, but it has recently rebranded to Credly. You will also get a digital certificate, as well as the exam will show up on your official Microsoft transcript. But with all of those options, what's the goal or how should you use them? Well, the Credly badge is built for sharing on social media or professional sites such as LinkedIn, for example, to show your achievement. As for the digital certificate, you can print it and showcase it in your office, for example. A few years ago, Microsoft used to allow you to order printed copies, but that's not possible anymore. But with a good quality printer, you will get similar results. Lastly, the Microsoft official transcript is very useful when a potential employer or client needs an official document to validate your certifications. If we dive a bit deeper about the Credly badge, they actually made it very easy to click the Share button, and then you can both share it as an update on LinkedIn, as well as add it to your LinkedIn profile as an active certification. If you prefer to not use the Credly mechanism, you can also manually add the certification on your LinkedIn profile. I strongly encourage you to add the certifications to your LinkedIn profile or any other professional social network that you use in order to showcase your continuous learning. Lastly, let's talk about the

Microsoft certification official transcript. When you share the transcript, you have an ID, which you cannot change, but you can customize your password, as well as decide if you want to include your address details or not. After that, a prospective employer can go to the transcript validation tool, enter your ID and password, and then validate your certifications directly on the Microsoft site. This makes sure that no one tampered with the transcript or faked anything in the PDF. The information comes directly from Microsoft, so it's more official. Now that we have talked about all of the ways to share your achievement, don't forget to celebrate your accomplishment. It's important to celebrate when we succeed. However big or small you decide to do it, don't forget to celebrate what you have just achieved. Also, don't forget to take a break, sit down, and enjoy the moment. Whether it's for a day or a week, take some time after the certification to just enjoy. And after that, start looking into your next certification, whether it's another exams or so on or a more advanced certification such as an or The cloud is always evolving, and it's important to always keep learning more and staying up to date, and I personally find that certifications are a great way to motivate me to keep learning.

Chapter 8 What if You Fail the Exam

Now that we have covered what to do if you pass the exam, what happens if the exam score report comes out and it's a fail? First of all, don't put yourself down too much. It's never fun to fail an exam, but it happens. I have failed certification exams myself, and it's important to see them as a learning opportunity to get better rather than a failure. Also, it’s important to know that if you fail, it will not show up on your transcript. So you're the only one that has access to this information that you did not pass an exam. Remember the score report that we have talked about previously? This shows you how well you did for each exam section. That information is even more useful now, because it can help you understand what to focus on as you review the material again for the next time you attempt the exam. Talking about retaking the exam, let's talk a bit about Microsoft's exam retake policy. If you fail the exam for the first time, you need to wait at least 24 hours before being able to retake the exam. If you fail a second time, however, you will need to wait 14 days before taking the exam a third time, and there is also a delay between twice for the third, fourth, and fifth attempts as well. Microsoft does have a restriction where you cannot attempt an exam more than 5 times in a period, starting from your first attempt. I hope that it never happens to you but I still wanted to share the information so you know about it. In summary, we have covered what happens right after the exam, and we have talked about the score report and the information it provides. We have also learned what to do if you pass the exam and I think you share it with the world and celebrate. Next we have covered the different digital assets you'll receive, such as the Credly badge, the digital certificate, as well as the Microsoft Official Transcript. After that, don't forget to take a break and then start looking at what your next certification might be, so you always keep learning and keep your skills up to date. We have also covered what happens if you fail, which can happen to all of us, and don't forget, failed attempts do not show up on your transcript, so really only you know. Remember to use the score report to identify which areas you need to improve on. This way, when you go back, you will feel more confident. This is it - you are now ready to go schedule your exam. Good luck and I hope you are going to pass it for the first attempt.

BOOK 5

AWS CLOUD PRACTITIONER

CLOUD COMPUTING ESSENTIALS

RICHIE MILLER

Introduction

This book will help you prepare for the AWS Certified Cloud Practitioner exam (Exam Code CLF-C01). In fact, this book is designed to be a shop for you. It has everything that you need to get ready for the exam. This book will not just help you through sample questions, but it will also walk you through and teach you how to think about the exam questions to become an AWS Certified Cloud Practitioner. AWS says that cloud computing is the delivery of compute power, database storage, applications, and other IT resources through a cloud services platform via the internet with pricing. First, we're going to walk through the process of creating your own AWS account, and this can be extremely beneficial as you're going throughout this book to be able to follow along and actually experiment within AWS. So, we're going to do three things. First, we're going to create a new personal AWS account, we will then activate the new account, and then finally, we will be configuring a budget alert for the account so we can make sure we don't have any unexpected charges that end up growing on our account. I'm going to go ahead and go through the process of creating an AWS account. I'm on the AWS home page, and it'll probably look different depending on when you're looking at it, but you should see at the top right a button that says Create an AWS Account.

So I'm going to go ahead and click on this option. Next, we're going to need to put in two different pieces of information.

First, we need to enter in a root email address. Next, we're going to need to give this an AWS account name. Once this is in place, we can click Verify email address. And so here it's going to send something to your email address, and you'll need to enter in the verification code. Once I've entered in the code, I can click Verify. Next it's telling me my email address has been verified, and I need to create a password.

Once the password is in place, I'm going to hit Continue. From here, it's going to ask if I want this to be a business or a personal account.

So I'm going to assume that most of you in walking through this book just want to create a personal account that you can test as you're going through this book, so I'm going to click Personal. I'll enter in my full name, I'll enter in my phone number, and then I'll enter in my address. Once all of that information has been entered in, I need to agree to the terms of the AWS customer agreement, and then I can continue. Next it's going to be asking for my credit card information, and it's important to note here AWS is a service, and when you utilize this service, it may charge your credit card.

However, it's letting you know here that there is a free tier where there are many different services you can try out without having to pay anything for those services within given limits, so keep that in mind, and we will set up a billing alarm so you shouldn't have any big surprise charges that end up on your account. The next step is confirming your identity, and it's going to do this by sending either a text message or a voice call to a mobile number that you specify.

So I'm going to go ahead and enter in my mobile number. The next thing I'm going to need to do is I'm going to need to pass the security check. Here I'll enter in the characters that are on the screen, and then I'll hit Send SMS. Next I'm going to enter in the code that I received on my phone, and I'll hit Continue. The next step is to select a support plan.

Don't worry, later on we'll walk through each of these support plans in detail, but for now, we will just stick with the Basic support option and hit Complete sign up.

We have completed the creation of an AWS account. From here, I can click the option to go to the AWS Management Console. I'm going to specify here that I want to sign in as a root user, and I'm going to give the root user email address.

Then I'll hit Next, and I need to enter in the password. Now I am here within the AWS Console, and this is the way that I'll interact with AWS from my browser.

But the first thing that we need to do is to set up the billing alarm so there aren't going to be unexpected charges that end up on the account. So I'm going to click under the name of the account, and I'm going to go to the option here that says Billing Dashboard. Now that I'm here, I'm going to go here under the left pane and go down to the option that says Budgets.

From here, I'll specify that I want to create a budget, and I'm going to select the option here that says Use a template, and then I'll go to Monthly cost budget. I'll specify here that my budget amount is going to be $10.00. You could specify a lower amount if you would like. I'll leave the budget name that they specify here, and I'll specify the email addresses that I want it to notify once the notification threshold has been met. So you'll notice here that all AWS services are in scope in this budget. We will be notified when our spend reaches 85% of the total amount, which in this case would be $8.50 for one month.

Also, we'll be notified when our actual spend reaches 100%. So with that being said, we can say Create Budget. Now we have a budget in place with notifications to let us k if our costs are even approaching the budget that we've specified.

Now we should be able to take advantage of many of the free tier services without having to worry about cost, but you shouldn't have to worry that you would have costs growing without being notified.

Chapter 1 Out-dated Data Centers

Now we're going to talk through how organizations leverage traditional data centers, and so we're specifically going to be focusing on organizations that have not yet shifted to the cloud, and we're going to be starting with a fictional organization called Acme. And within Acme, there is a group that has decided that they should launch a new social network for professionals as a new line of business for the organization. Initially, they're focusing on the United States at launch, and if they're successful, they are looking to expand into Europe and Asia. They’re currently working to secure funding for the initial infrastructure because it's going to cost a good deal of money to build up the initial data center. So let's begin to play this out. So here at Acme, they were able to secure an initial round of funding, and since they're located in Chicago, they built a data center in Chicago. It took five months from when they started to when they were actually able to move in. And once they were able to load their servers in, they knew they needed several types of servers. They needed some web servers so they could serve their content out through their website, they needed some file servers so that they can store user uploads like profile pictures, and they also needed a database server so that they could store user information like messages and who their friends are. They had a great launch in the United States, and they feel like there is a lot of growth ahead, so they want to plan a move into Europe. They specifically want to plan to move into London. However, once this process starts, it's going to be another 6 to 7 months for them to be able to move into that new data center. Because of this, they need to go through that entire process and secure more funding, and they need to order some more servers. After seven months, they're finally able to move in some servers in their data center in London as well. But they don't want to just stop there. They also want to move into Asia, but something unexpected happens. They have tremendous growth in the United States, and they need to expand their data center in Chicago so they're able to go in and add some more servers. But once that demand happens, it takes them at least a month to order the servers, get them there, get them installed, and configured. So during that time, several users were experiencing performance issues with their social network. Once all of that's in place, they're finally able to shift their attention to the data center in Asia, which is going to be in Hong Kong, and it takes another six months to begin to get that in place and to order servers, and they then deploy their servers in Hong Kong. But something happens. Users that were interested in the platform in Hong Kong have already moved on to something else. And so they have all of these servers and their data center sitting there in Hong Kong being barely used, and they have put a significant investment in getting it up and running. Let's pause on Acme for a minute, and let's take a look at what we've learned. First of all, when we're looking at traditional data centers, it takes an initial upfront investment, and a pretty large one at that. Organizations have to plan to either build a new data center or rent space within an existing data center. Next, forecasting demand is difficult. Even the best organizations struggle at this, understanding how many users are going to latch onto a new solution, whether it's an external one, like a social network, or even an internal business application. Next, it's slow to deploy new data centers and servers, so not everything can be ordered overnight via Amazon Prime, and so there are times when it will be very difficult to get up and running if we need to respond to a change in demand quickly. Next, maintaining data servers is expensive. With Acme, for every new data center that they spun up, they're going to need to add dedicated resources, dedicated individuals to maintain that server infrastructure at each of those data centers, and that's a cost that's ongoing. And next, you own all of the security and compliance burden with each of those data centers. So nobody wants to get hacked, and nobody wants to be sued for being hacked, but if you're an organization owning those data centers, that entire responsibility rests on your shoulders to be sure that you're following all of the industry best practices. With this said, we're next going to look at some of the benefits of cloud computing. AWS calls out several key advantages. The first of them is you trade capital expense for variable expense. Let me break that down for you. What that means is we don't have this large capital investment initially to build out a data center, like we talked about with Acme. In this case, we get a variable expense, meaning that if we're using a server in the cloud, we simply pay for the amount of time that we're using it, and we can throw it away at any point. We're going to talk a lot more about the economics of the cloud later within this book. Next, we get to benefit from the massive economies of scale. So this really factors in in two different areas. One is just the cost of building out a data center and all of the equipment involved. AWS buys things on a very large scale, and so they're able to make sure that they're getting the best price for those. In addition, in the economy of scale, when we look at it from a maintenance perspective, Amazon has determined how to manage all of these fleets of servers effectively at scale, and so those cost savings get passed down to us as consumers. Next, you can stop guessing capacity. We talked about with Acme how difficult it can be to properly predict demand. Well, what if you just didn't have to? What if you could choose to have a system that could either grow or shrink based on demand? That's one of the paradigm changes you get with cloud computing. Next, you get to increase speed and agility. So when we talked about Acme in the previously, we mentioned that they were looking to find funding partners before they were able to go out and test their social network. Well, that's not very agile. Ideally, we want to be able to test out our hypothesis against real users and to do it quickly, and one of the things you can do with cloud computing is try out new ideas and have minimal cost to do so. Next is you can stop spending money maintaining data centers. So with Acme, they were having to staff individuals at each of the data centers, and at each of the data centers they would own that maintenance burden for as long as the data center was up and running, and so this allows you to shift away from that. And next, go global in minutes. So in the case of Acme, they were having to wait six or seven months to move to a new region within the world. However, what if they could simply send data over to Hong Kong or to London at any point without having to build out new data centers? They literally could do it within minutes instead of within months. With these advantages, I want to focus on a few key terms that you need to be familiar with. The first of them is the concept of elasticity, and this is what we talked about earlier when we were talking about demand, but it's the ability to acquire resources when you need them and then release them when you no longer need them. And in the cloud, you want to do this automatically. AWS has several services available to you to do just this, and we'll talk more about these later. The next concept I want to talk about is reliability. When we talked about Acme, one of the things that we didn't even cover within their data centers are things like failover. So what happens if one of your data centers goes down? Well, one of the advantages of AWS is that it has global infrastructure that is built with reliability in mind because we want our solution, whatever it is, to always be available to our users when they need it, and so we can take advantage of this by building a solution on AWS. And the next concept I want to talk about is agility. We want to have the ability to lower the cost of trying new ideas or business processes. So instead of having to go out, for example, let's take a new concept like blockchain, instead of having to go out ourselves, hire individuals that have their expertise, buy the hardware that we need, install it in a data center, and then start using it, in this case, we're simply able to tie into the resources that AWS has and use the services that they provide to us. Next, it reduces the time required to maintain infrastructure. So we begin to shift. Instead of simply spending time maintaining, we're able to spend time working on things that add business value. Next, it does reduce the risk for the organization around security and compliance. , don't let anyone tell you that you don't have to think about security when you move to the cloud, you certainly do, and we will be looking later at the AWS shared responsibility model so we understand what is our responsibility and what is AWS' responsibility when it comes to security and compliance. And next, it provides access to emerging technology, as we mentioned earlier with blockchain, without having to spend a ton of money to ramp up on that new technology

Chapter 2 Cloud Computing Types & Scenarios

So now that we've gone through and discussed benefits of using cloud computing, let's take a minute and talk about different types of cloud computing. And before we dive too deep into that, let's discuss a definition for cloud computing as a whole. AWS says that cloud computing is the delivery of compute power, database storage, applications, and other IT resources through a cloud services platform via the internet with pricing. And that pricing is critical, as we discussed previously. Let’s look at some different models of how we use cloud computing. So let's imagine for a minute that we have a spectrum, and on this spectrum on one end we have absolute control, but on the other end we have minimum maintenance where things just run without us even having to think about it. Let's imagine three different points on this spectrum. First of all, on the far left with maximum control, we have Infrastructure as a Service, or commonly abbreviated as IaaS. This is where we run servers in the cloud, virtual servers, that are very similar to how we would run servers in our own data center, meaning that we have full access to those servers. We can change the OS that's running on them, we can completely configure them to do exactly what we want; however, we also have to perform maintenance on those servers, keep them up and running to do exactly what we want them to do. But then if we go to the far right of the spectrum, we end up with something else, and that's Software as a Service, or SaaS, commonly abbreviated as SaaS. For Software as a Service, you might not realize it, but you're probably using multiple Software as a Service solutions every day, and that could be your email provider, or that could be your specific chat service that you use with other employees. It's just a piece of software that you're able to run with. You don't have to worry about configuring servers. It is just provided to you as a service. And in the middle of these two, we have something called Platform as a Service, or PaaS, and this means that we're given a service that is configured for us where we simply need to deploy our customizations onto it. And so if you've ever used maybe a WordPress host, like WP Engine, you can simply drop in your code, and you can be up and running. They configure the WordPress install in the server for you. There also is a Platform as a Service solution on AWS called Elastic Beanstalk. So with these different options, we can understand that there are different ways to use cloud computing. Next, let's talk about different cloud deployment models. The first model is public cloud, or sometimes people will just say cloud, and this is when you are deploying a solution onto a public cloud provider, like AWS. And there are others here as well, like Azure from Microsoft or the Google Cloud Platform. Next, we have or private cloud. You'll also sometimes hear people say just And this is when you have a platform in a private data center. So if you wanted to be able to be somewhat scalable, like you have with AWS, but you wanted to be able to do it within your own data center, there are solutions that you can deploy from companies like VMware, for example, to have a private cloud. And in some cases, you will have what we call hybrid, meaning you have an organization that is leveraging both a public cloud, but those public cloud applications are working in tandem with a private cloud within their own data centers. And so it's important to understand that for some, especially large organizations, they could be using both a public, private, or using them together in a hybrid model. Now, we're going to run through some scenarios that are based on what you learned so far. We'll be following this pattern for the remainder of the books in this path. So we'll present the scenarios here, and then we'll be running through the solutions. That way, we'll give you time to think about the answers. So let's look here at Scenario 1. We have Rob, and his company runs several production workloads within its data center. They use VMware to manage the infrastructure in their data center, but they want to use AWS, and they want to integrate it in with their data center for some of their new workloads, so using the two in tandem. In this case, which cloud deployment model would his company be following? Now, let's look at Scenario 2. So here we have Elenor, and her company is trying to decide whether to fund a new line of business, and in this case, her team is looking to monetize a new emerging technology. The new line of business will require some new infrastructure, so beyond what they currently have. So what benefit of cloud computing would be most relevant to her company? Now, let's look at our last scenario, and this is going to be Judit. And Judit is the CTO at an insurance company, and this is just a insurance company, and they are considering moving to the cloud instead of colocating servers, which means where you're renting space in another data center. They want to make sure that they have maximum control of the cloud servers, and they want to do that for security and compliance reasons. So in this case, what cloud computing model would they need to leverage? Well we'll go ahead and run through each of these scenarios and explain the answer, as well as why that's the right answer for that scenario. In summary we've covered quite a bit in this initial chapter here in this book, and I want to quickly review each of the things we've covered before we dive in and take a look at our scenarios. Then, we created an AWS account for personal use. Following that, we examined how organizations leverage traditional data centers, and we focused on a fictional company, Acme, and how they were looking to deploy a new professional social network. Following that, we explored the benefits of cloud computing, and then we reviewed the different cloud computing models. And finally, we understood the cloud computing deployment models. So with that said, let's take a look at our scenarios. So our first scenario was Rob, and his company was running their own data center where they were managing their virtual servers with VMware, and then they wanted to also work alongside AWS in the public cloud. So which cloud deployment model would his company be following? Well, the answer to that is hybrid cloud, because they were using both a private cloud in their own data center and a public cloud, being AWS. Next, let's take a look at Elenor, and they were looking to monetize a new emerging technology. So with this new line of business and the new infrastructure that would be required, what benefit of cloud computing would be most relevant to her company? Well, the answer here is the concept, transitioning from that large, upfront investment to a variable cost. This way, they can try out this new emerging technology, and they can try it out at a small scale and only pay for what they're using instead of having to make a large, upfront capital investment. And, we'll be talking through the economics of the cloud later within this book. Next, Judit is the CTO at an insurance company, and they were looking to move to the cloud, but they wanted to make sure they had maximum control of their cloud servers, so which cloud computing model would they need to leverage? Well, in this case, it's going to be Infrastructure as a Service, or IaaS, because they want to have maximum control, so just like they were running these servers in their own data center, and probably for security and compliance purposes given that they're an insurance company. So in this case, instead of choosing to do Platform as a Service or Software as a Service, they would fully take advantage of Infrastructure as a Service to make sure that they had that maximum level of control.

Chapter 3 AWS Regions and Availability Zones

Now, we're going to be talking about the AWS Global Infrastructure, and that just means we're going to be looking at how AWS deploys its infrastructure globally and then how they make those different bits of infrastructure available to us to build solutions on the platform. We're going to talk about four distinct areas of the AWS Global Infrastructure. First, we're going to be discussing the concept of Regions. Then, we'll be discussing Availability Zones. We'll take a look at Local Zones, and then we'll take a look at edge locations. So, over the book of this chapter, we're going to first be reviewing the elements of the AWS Global Infrastructure. We'll start off by understanding the use of AWS Regions. We'll then be understanding Availability Zones within AWS Regions. We'll also be taking a look at a newer concept, which is the use of Local Zones. We'll then be reviewing the purpose of edge locations, and we'll take a look at how you can utilize the AWS Global Infrastructure visualization. So next, I'm going to be introducing two very key concepts of the AWS Global Infrastructure. The first is AWS Regions, and the second is Availability Zones, and you'll see that these are very closely connected. First of all, let's take a look at AWS Regions. The first thing to k with a Region is that each of them are located within a specific geographic location. You might think, that means they have a data center there. Well, no. In this case, each geographic location has a cluster of data centers, so more than just one.

AWS currently has 30 Regions that have been launched around the globe. So here we can take a quick look at these, but I'm also going to show you in a bit where you can go to get an always updated map that shows you where the Regions are located.

But next, let's talk about another concept, Availability Zones. Within an Availability Zone, it contains one or more data centers. In addition, multiple Availability Zones are included with each AWS Region. And this is different from some other cloud providers where Availability Zones is an optional feature. Here, every AWS Region does include multiple Availability Zones. Also, it's important to note that these Availability Zones, they are located within the geographic area of the AWS Region. So when we have a Region like the one in London, all of the Availability Zones will be located within that Region. These Availability Zones have redundant power, networking, and connectivity. So the reason that these exist is to make sure that you generally wouldn't ever have a scenario where an entire Region would go down, and we'll talk more later about how these Availability Zones enable high availability for your applications. And there are currently 96 Availability Zones globally. Let's just take a look at some Regions within the United States. First of all, we have six different Regions that exist within the United States.

So the Availability Zones here are associated with the Region that you can see that they're closest to, and these Availability Zones provide a level of failover to ensure that you have high availability for the applications that you deploy. And let's talk about a few terms here. First, we have the term of availability, and this is the extent to which an application is fulfilling its intended business purpose, and applications that are highly available are built in a manner where a single failure won't lessen the application's ability to be fully operational. There’s another concept that you need to be aware of, and that is how we name AWS Regions and Availability Zones. So you might see something at some point like so let's talk about what this means. Let's break down each of the components. So first, you might guess that us in the beginning stands for United States, and you would be correct. Here, this is going to be the area. We then have a underneath that main area. So in this case, this would be on the eastern side of the US. We then have a number because we might have two different Regions or more in a specific area, and that is the case. We have both a which is in Northern Virginia, and a which is in Ohio. But we also have a letter at the end, and this is for the Availability Zone. So, for example, if you had two Availability Zones you were considering in Ohio, you would have and That’s the complete name, but if we leave off the letter, that gives us the Region name. So when we're talking about a Region, we would say as a Region. But if we want the entire Availability Zone name, it would include everything, from all the way to the end to 2a. I want to just show you here is a list of the currently launched AWS Regions.

There are some Regions in China and those that are used by the US government, and those are not on the list because those aren't available for everyone to use.

Chapter 4 AWS Global Infrastructure

I want you to be familiar with two additional concepts when it comes to AWS Global Infrastructure. The first is the concept of a Local Zone, and the second is a Wavelength Zone. So let's take a look at these two concepts. An AWS Local Zone places compute, storage, database, and other select AWS services closer to end users. Each AWS Local Zone location is an extension of an AWS Region. AWS Local Zones provide a secure connection between local workloads and those running in the AWS Region, allowing you to seamlessly connect to the full range of services to the same API and tool sets.

That might not make a lot of since yet. Let's take a look at a map and see if I can help drive this point home. So here you can see the six different AWS regions that I included on the map previously. In this case, we have two primary regions that are going to be associated with Local Zones, so here I'm going to fade the others out. And these two that we're looking at here is we have and we have I'm going to go ahead and show you all of the current Local Zones that exist within the United States. And you might notice that these actually are in key areas, such as Dallas and Houston and Miami and Chicago and Seattle and Los Angeles, as well as many, many more.

The way this works is that highly efficient connection that exists between those specific Local Zones, and the Region can be demonstrated this way. Here you can see that a vast majority of the Local Zones on the West Coast actually link up with as their parent region. And what we see over on the East Coast is that all of those link up with as its parent region. So, why would this matter? Well, if you're trying to build an application in Chicago, you might want to make sure you have some infrastructure running close to the end users that are going to be in Chicago, and you could accomplish this by utilizing a Local Zone, in this case, the one in Chicago, partnered with the parent zone, which is which is located in Northern Virginia. Next, I want to talk to you about Wavelength Zones. Wavelength Zones are AWS infrastructure deployments that embed AWS compute and storage services within communication service providers' 5G networks, so application traffic from 5G devices can reach application servers running in Wavelength Zones without leaving the telecommunications network. So if you're looking to create a highly efficient application that can run off of wireless providers on a 5G network, this is what Wavelength Zones were designed to do.

So next, we're going to discuss another element of AWS Global Infrastructure, and that is AWS edge locations. But to understand this, I also need to introduce you to another term. Points of Presence are elements of the AWS Global Infrastructure that exists outside of AWS Regions. These elements are located in or near populated areas, and specific AWS services use them to deliver content to end users as quickly as possible. Within the overall Points of Presence, there are two types of infrastructure edge locations, which we'll be focusing on here, and regional edge caches, which we won't be covering to prepare you for your Cloud Practitioner exam, but it is important to know that they exist. At the time of this book, there were over 410 Points of Presence globally within the AWS Global Infrastructure with over 400 edge locations and 13 regional edge caches. Let's talk about why that's significant. First of all, these edge locations are used as nodes of a global content delivery network, and there are specific services that leverage these edge locations, primarily Amazon CloudFront and Amazon Route 53, and you'll learn more about these services and what they do next. But there are over 400 different locations, as I previously mentioned. Next, it allows AWS to serve content from locations closest to users. So, for example, if you wanted to create a application and you wanted to be sure that it could be served as quickly as possible, and let's say you have an end user in Copenhagen, well, you could make sure that this is being served from the edge location in Copenhagen via Amazon CloudFront instead of making them go to wherever the next closest AWS Region would be. Now, I want to show you a resource that can help you actually plan how you utilize the AWS Global Infrastructure. We will be reviewing the method of accessing the AWS Global Infrastructure site. We'll also be reviewing Regions and Availability Zones within the site, as well as the edge locations. So I want to show you the AWS Global Infrastructure site.

If you scroll down from this main part of the page, first of all, you'll notice here are some key statistics about the global infrastructure, including the number of Regions and Availability Zones and Points of Presence. But if we scroll down, we're going to see a map.

There doesn't seem to be a whole lot going on with this map, and so you might think this isn't of huge value, but I believe that as you are creating infrastructure on AWS, this is going to be an extremely valuable tool for you to take advantage of. Let's say, for example, that you want to build a solution in Asia and Australia. Well, if I go in here to the map and I mouse over this area, you can see that all of Asia and Australia actually lights up. I'll point out here that first, at a high level, we can see the different Regions that exist. So for example, I can see that Hyderabad has a region that launched in 2022, and it has three Availability Zones.

I can also see another one in Mumbai, and this one has three Availability Zones and one Local Zone. And I can see that for all of the Regions. Some of the Regions aren't always available to you. For example, if I look here at the Region here in China, this particular one requires you to opt in, and not everyone can have access. It will also show you regions that are planned. We can see, for example, that a region is planned here in Thailand, as well as in Melbourne, and in Auckland. So, with that said, that information in and of itself is valuable. But if we click into an area, we can see a lot more information. Here, I can see a listing of all of the Regions, and then in parentheses, I can actually see the number of Availability Zones associated with that Region.

I also here can see all the edge locations that exist, as well as the regional edge caches that exist within this area as well. And at the top, I can get here just some key statistics. So we can see at a very high level there are 35 Availability Zones within 11 geographic regions, 34 edge network locations, and 5 regional edge cache locations. And you can do this for every geographic area that's included here on the map. We can see the same here for North America. We can see, for example, that there are 25 Availability Zones within 7 geographic regions, 44 edge network locations, and 2 regional edge cache locations.

So for example, if I look at Atlanta, Georgia, I might not have a Region that is super close to Atlanta, Georgia, but I can see that there are three different edge locations there ensuring that if I'm using a service like CloudFront, that my data is going to be extremely close to the end user. So this is how you can utilize the AWS Global Infrastructure visualization to help plan how you utilize the AWS infrastructure that exists globally. Next, we're going to cover some scenarios here, and these scenarios are going to help you see how well you have retained the information that has been covered in this chapter. As a reminder, we're going to go through the scenarios here, and I'm not going to go through the answers, so be sure that you take time to write down your answers before you move on to the next chapter. For our first scenario here, we have Jack, and Jack's company is looking to transition to AWS, and they're going to start with just a few workloads. However, it is a requirement to store backup data in multiple geographic areas. So, which element of AWS Global Infrastructure will best suit this need? Moving on, next we have Tim, and Tim's company serves content through their site to users around the globe, and they're looking to optimize performance to users around the world. They're looking to leverage a content delivery network, or CDN. So, which element of the AWS Global Infrastructure will be used in this case? Elise's company is transitioning one of their legacy applications to AWS, and this application requires uptime of at least 99.5%. They want to be sure that any issues at a single data center don't cause an outage. So, which element of the AWS Global Infrastructure supports this need? Before we move on to the answers to the scenarios that I presented previously, let's just take a quick look back at what we have been able to cover. First of all, we reviewed the elements of the AWS Global Infrastructure. As a part of that, we understood the use of AWS Regions and why having these different geographically based regions helps people create applications that can be highly available. And as a key part of that, we talked about the Availability Zones that exist within those AWS Regions and how they make sure that even if you have an Availability Zone that goes down, the entire Region still stays up and running. We also talked about the use of Local Zones and how they partner with parent zones to help bring content to end users. We also talked about the purpose of edge locations, especially as it ties into a content delivery network, or CDN. We also utilized the AWS Global Infrastructure visualization to help understand which infrastructure elements exist at what place in the world. So, with that out of the way, let's go ahead and visit our scenarios. So first up, Jack's company was looking to transition to AWS, and if you remember, we were trying to determine which element of the AWS Global Infrastructure will best suit the needs that they have. Well, in this case, the answer is AWS Region because there are Regions that are geographically based, and so even if they're deploying workloads within a single Region, they can actually save that backup data across multiple Regions. Maybe they're running in Northern Virginia with but they want to store that backup data in which is in Ohio, as well as which is in Oregon. The next scenario here we had Tim, and he was serving content through their site to users around the globe. So, to do this, which element of the AWS Global Infrastructure will be used? Well, in this case, it's going to be the edge location. And the key word here is content delivery network, or CDN, because here Amazon CloudFront, which is the content delivery network solution on AWS, utilizes edge locations to make sure that that content is as close to possible to those end users. And if you remember, we have over 400 of those locations that exist globally. For our final scenario, we had Elise, and if you remember, her company was transitioning a legacy application to AWS. So which element of the AWS Global Infrastructure supported her need? Well, in this case, the answer is the AWS Availability Zone, which will sometimes be referred to as an AZ. And the reason is, is because here, if we need to have a high uptime, that means we need our application to be highly available. So by taking advantage of multiple Availability Zones within a single Region, we can help make sure that the application won't go down if just one data center has a problem. The Availability Zones help make sure we can create applications that are indeed highly available.

Chapter 5 Cloud Economics

So in this chapter, we're going to be talking about the economics of the cloud, and we're going to cover several topics. First of all, we're going to be looking at the funding differences between traditional data centers and the cloud. And then we're going to look at the tools for the organization of our costs. And then we'll be utilizing AWS tools to actually make a case for moving our organization to the cloud. And then we'll be exploring AWS costs using the tools. To start off with, I want to talk about two terms that have to do with how we finance our projects. The first term is capitalized expenditure, or CapEx. So, let's say you're building your own data center. When you're building that data center, you have to make a large upfront investment, and that's going to cover buildings and servers and supporting equipment. This type of expense is there to attain a fixed asset, and we're going to call that a capitalized expenditure. The other type of expenditure we're going to talk about is an operating expenditure, or OpEx, and this has to do with the regular expenses of your business, and these are OpEx. So when you build a data center, your ongoing costs around connectivity and your utilities and your maintenance cost, those would be considered OpEx. You might be saying, what does this have to do with the cloud? Well, let's dive into that. Let's talk first about having your own data center. So if you have your own data center, let's say that you decided to launch an application within that data center, and let's say that here is your demand. It starts off pretty low because it's a new application, but then it continues to grow, and you can even see, for example, that between month 2 and month 3 it more than doubled. And so with this, we're going to look at our capacity. So our data center was built to handle an initial amount of capacity, and we're noticing here that there's a few problems. First of all, we have unused capacity. So when our data center launched and we then launched our application within the data center, we had to account for the amount of capacity that we believed our application needed. So in the beginning, before users were leveraging our application, there was a lot of unused capacity, and that's capacity that we're paying for whether it gets used or not. But then we have another even more critical issue, and that's that we have demand over capacity. And that means in these cases we have users that are getting denied access to our application because we don't have the data center power that we need to actually serve those users. So if you notice, between month 4 and month 5, we decided to make an investment and build out our data center to handle more capacity. Let’s talk about how you fund this type of initiative. So initially, when you're building out your own data center, you're first going to have a very, very large upfront capitalized expenditure, or CapEx, to build out your data center, to take care of that building and all of those servers that are going to be needed. Then, you're going to have an operational expenditure that's going to happen month to month. But if you notice, when we increased our capacity between month 4 and month 5, we had another large upfront capitalized expenditure, followed by an increased operational expenditure as we have to operate more servers and more equipment. This is how it works with your own data center, but let's talk about how this works in the cloud. So let's say that we have the exact same application with the exact same demand in the cloud. In this case, we're able to do something different with capacity. We're able to make sure that we have just enough capacity to meet the demand that our users have for our application. And another thing is different. Let's look at the costs. So here, with the costs in the cloud, so there is no capitalized expenditure because we're leveraging the cloud, and here our operational expenditure mirrors what we have in terms of the demand. So what does all of this mean? Well, let's compare and contrast our own data center and cloud infrastructure. So with our own data center, there is a large upfront cost. That's CapEx. Also, there's a potential for either underused capacity or unmet demand, and in many cases, these are going to be inevitable unless you can predict the future. Also, increasing capacity takes time and additional investment because you have to order servers, you have to build onto your data center, you have to add additional cooling and networking and a lot of other things to support your data center, and that doesn't just happen overnight. So you're going to have situations where if there is a need for increased demand, there will be time that will elapse before you can meet that. Then, monthly costs, in this case, will map to predicted infrastructure needs. This is why I mentioned earlier that you need to be able to predict the future if you're going to go down this path, because you're going to, in this case, have costs that align with what you think the demand will be. But let's talk about cloud infrastructure. So first of all, there's no upfront investment. So you don't have that large CapEx at the front end because you're not building out your own data center. Next, you pay as you go for infrastructure. And this concept is critical because it allows you to only pay for the demand that your users have. And it can scale to meet that demand, and that additional capacity can be provisioned immediately, which is a big difference to having to build onto your data center and add more servers. And another factor is that monthly cost will map directly to the user demand.

Chapter 6 How to Organize and Optimize AWS Costs

Now that you have an understanding about the difference between working within your own data centers and working in the cloud, let's take a look at how we can organize and optimize our AWS costs. So first, we're going to be looking at one of the most critical tools for managing your cost within AWS, and that is the AWS Cost Explorer. It provides you with a user interface for exploring your AWS costs, and you access this user interface through the console in your browser, and it will provide you with breakdowns on your cost, including by service. And it's important to note here that, again, AWS is a collection of services. So if you're utilizing a service like EC2 for your virtual machines and S3 for your storage, you would want to k how much am I spending on each service. In addition, we have a breakdown by cost tag, and we'll talk about cost tags later. One of the useful features of Cost Explorer is that it also provides predictions for the next three months of your costs based on your current usage. This can help you better understand what your monthly run rate is for the AWS services that you're leveraging. And here, it also will provide recommendations for how you can optimize your costs. , you might want to utilize the console in the beginning, but if you wanted to automate aspects of this cost analysis, you can also access the same data via an API. Let’s talk about another service that works hand in hand with Cost Explorer, and that would be AWS Budgets. Earlier, we set a budget alarm, so you already have had some level of exposure to AWS Budgets. But this utilizes data from the AWS Cost Explorer to plan and track your usage across AWS services, and it can track cost per service, service usage, Reserved Instance utilization and coverage, and Savings Plans utilization and coverage. I understand that we haven't yet talked about Reserved Instances or Savings Plans, but we will be talking about those throughout the book of this learning path. There are some tools that we can use to help plan our future costs. And first, we have the AWS Pricing Calculator, and we'll actually do a demo of this later within this chapter. But this gives some analysis of your potential costs by utilizing multiple AWS services for your workloads. So let me explain that. Let's say you were looking to launch a social network for your company, and you believe you're going to need five different web servers, you're going to need two different database servers, and you're going to need 500 GB of storage inside of a service like S3. Well, you can enter all of those factors into the Pricing Calculator and determine how much it would cost. You can even look at different ways to build it and then compare costs between the different approaches. So another approach we can take is when we're just looking to migrate existing workloads into the cloud, and there are tools you can use to actually make the business case for moving into the cloud. One of those tools would be Migration Hub, but it's not the only tool. But this gives you the ability to go through and make a business case for transitioning your workloads to the cloud by comparing costs from your own data center to what it would be in the cloud. We will examine the way you can utilize AWS tools to make the business case for moving to the cloud. So next, I want to highlight a couple of deprecated tools in case you see those listed in blog posts or other things you'll k what people are talking about. Originally, we had the TCO Calculator, which did enable an organization to determine what could be saved by leveraging cloud infrastructure. That's been replaced by some other tools that we'll talk about later. Also, before we had the AWS Pricing Calculator, we had the AWS Simple Monthly Calculator, and it did similar things to what the Pricing Calculator can do. It just couldn't do everything that the current Pricing Calculator can do. So if you see either of these, just know that these are deprecated tools. These, in essence, are previous generation tools that we no longer use. So let's talk next about AWS resource tags. Really, these are just metadata that gets assigned to a specific AWS resource within our account. We can give it a name and an optional value. So let's say, for example, with the use case I mentioned just a little bit ago, if we have a social network that we're launching and we have maybe five web servers, well, we could go in and just put Web Server as a tag on each of those, and then when we went in to look at our costs, we could say, okay, how much are we spending on everything that has the tag Web Server? But another common use case here is to include department, environment, or project. So, let's look at that for a minute. Let's say that you have some internal applications that HR is using. What you can do is you could set a name of Department, and then you could enter in an optional value of HR. Or we could break it down by environment. Many times when we build applications in the cloud, we'll have a testing environment and a production environment. So we could create a tag with the name Environment, and the value could be either Production or Testing. With this, you can utilize cost allocation reports to see your cost grouped by active tags. This is something you can enable within Cost Explorer. Also, we can use the tags within Cost Explorer when we're looking at our costs. There’s another way that we can actually group our costs together, and that is with AWS Organizations. And this allows organizations to manage multiple accounts under a single master account. So instead of doing what we mentioned earlier with Department and having HR, for example, as one department, we can actually create a completely separate AWS account under a master account that is for HR. And in some ways, that makes tracking and access control even easier because it's in a completely separate account. One of the features you get by using AWS Organizations is consolidated billing. So this means that even if you have 100 different AWS accounts using AWS Organizations under a single master account, you only get one bill. And we can see within this bill a breakdown of our costs by account. We also can go into Cost Explorer in the master account, and we can see the breakdown of costs within each account and then even breakdowns further beyond that. And this also can enable organizations to centralize things like logging and security standards across multiple accounts. Now, let's talk about how organizations can actually build a business case for moving into the cloud from their own data centers. So let's talk about what you need to do to build a business case. First of all, you need to analyze your current workloads. You need to understand the applications and processes that you have running so that you can begin the process of forecasting what those infrastructure needs could be in the cloud. We’ve already reviewed a tool that once we k what resources we need, we can use to determine the cost in the cloud, and that would be the AWS Pricing Calculator. But in this case, how can you streamline the process of analyzing those workloads and forecasting the future needs, because then you ultimately can create a TCO analysis, or total cost of ownership analysis, of both options? There are tools that help you in this process within AWS. First, we have Migration Hub, and this is a central location to gather information from multiple different AWS tools together to forecast your needed infrastructure, and you can use this information to begin to build out your business case. There's also a more option that goes even further in building out your business case, and that would be the Migration Evaluator. And this is an service to calculate your infrastructure needs and build a business case for the cloud. There’s a lot of discussion we could go into with each of these, but what I want you to understand at this level is to understand that these both are tools that can help organizations decide if a move to the cloud makes sense for them.

Chapter 7 How to Use the AWS Pricing Calculator

Now, we're actually going to walk through the process of using the AWS Pricing Calculator. And as a reminder, this is designed to estimate future workloads. we did have an older tool, the AWS Simple Monthly Calculator, but this was replaced by the newer AWS Pricing Calculator, and that's what we'll be using here. So let's talk about what we're going to do exactly. First, I'll show you how to access the AWS Pricing Calculator in your browser, and then we'll talk through the process of estimating costs for a workload on the cloud using the calculator. We'll even look at how we can save and share the results with other people. So here I have my browser pulled up to the AWS Pricing Calculator.

You don't need to be logged into an AWS account to use the AWS Pricing Calculator. So here I'm going to click the option to create an estimate. Here we can see a lot of different options. Let's just take this piece by piece.

First of all, we're going to say here that we want to work within the region, which is in Northern Virginia, and it lets me know here that there are 148 different AWS services that are available here. So here, if we want to use a service, we can simply go in and enter the name for the service. I'll add a caveat here. I know we haven't reviewed a lot of AWS services yet, so I'm going to enter in some information here, and you might not fully understand what those services are for just yet, but we'll be covering that in depth later. But from here, let's use our social network example. Let's say that I need to have some web servers for my social network, so I'm going to spin up some virtual machines in this case. So I'm going to search for EC2 as a service. So I'm going to click here where it says Configure.

From here, I can enter in a description for this part of the estimate, so we'll say Webservers. And then we can specify that we either want to do a quick estimate or an advanced estimate. For the purpose of this demo, we're going to stick with the quick estimate, but the more you learn about AWS, you may want to go through the advanced estimate process, and it will be able to give you a more accurate description of what your future costs would be. Here we're going to say that we're going to use Linux servers, and let's say I want to have at least 8 vCPUs and 32 GB of memory. And then it's going to give us a recommendation, which in this case is a t4g2xlarge. I realize that sounds like a foreign language, but this is a name given to an instance type for a virtual machine that we're spinning up. I'm going to say that I want to have five of these, and these are going to be 100% utilized. We have the option here to pick different pricing strategies. And again, I know we haven't been through this fully yet.

I'm just going to leave the options that they have selected here, and I'm also going to say that I want each of those virtual machines to have a 20 GB volume attached to them. So we can save and actually add this.

So now that I have this in place, a portion of my estimate has already been added in. You can see here that I can see at the bottom an overview of my monthly cost and my total 12 month cost already. But we also need to add in some database servers. So I'm going to go here, and I'm going to say rds. Tat stands for Relational Database Service. And I'm going to scroll down. I'll select the option for Postgres, so I'm going to hit Configure for this. And let's say that we're fine here with the size, a db.m1.large, but let's say I want to have three different nodes. And I can scroll down here and see that I also want these to be 100% utilized, and we want to use the Zone deployment option, and that's going to give us the ability to have high availability with this particular database cluster.

We're also going to say here that we want to use an RDS proxy, and I'm going to say that 30 GB of storage for each node is going to make sense. We also could go in and add in backup storage. Let's say that we want to have an additional 40 GB of backup storage, and we can just leave the rest of the values at their current defaults. So I can hit Save and add that to the service. So you can see that the bottom has been updated to include an updated monthly cost and our total cost. I'm going to go ahead and click the option to view the summary.

This is going to give me a complete breakdown, and we can see here that we have Amazon EC2, we can see we have Amazon

RDS, and in all likelihood we would have several more things as well. We would need to calculate storage, and we would need to look at the data that's going in and out of AWS, but for this gives us a good baseline to work from. We could go in, and we could add an additional service. We could go in and we could group different costs together. But for we have an estimate, and what we can do from here is we can actually export it. We can either export it to a CSV or PDF file, but another great feature, and one that you'll find yourself using often if you're collaborating with other people to build applications, is the ability to share these costs. So I'm going to click on the option here to share, and it first is going to tell me that this is going to be public data. It's going to be stored on public AWS servers, and it's going to give a link that doesn't require you to log in because, as you don't have to log in at all to use the Pricing Calculator. So you'll have to agree to this before you're able to actually create a URL where you can access this estimate. If you say, I don't want my information to be public, well, you can simply export your estimate as a PDF or as a CSV file and then send it to whomever you'd like. But with this approach, you end up with a URL that someone else can utilize to go in and look at your estimate and then continue customizations based on that estimate.

Chapter 8 How to Review Costs with the Cost Explorer

So next, we're actually going to get in and utilize the Cost Explorer within the AWS Console, and we'll see how we can review our costs within our account utilizing this tool. So here in this demo, here's what we're going to be doing. First, I will show you how to actually access the AWS Cost Explorer within an AWS account. We'll then be reviewing charges by service for an AWS account, we'll be utilizing the predefined reports included with Cost Explorer, and then ultimately, we will be downloading the data from the AWS Cost Explorer.

I'm here in the AWS Console, and I've actually logged into a different account than the one I created earlier within this book, because since we're looking at AWS costs, I wanted to give you an account that actually had some real AWS charges on it. So to get to Cost Explorer, I'm going to go under my name, and I'm going to select Billing Dashboard. Here, within the left pane, I'll select Cost explorer, and then I'll need to click this button here to actually launch Cost Explorer. So we're inside of AWS Cost Management, and at a high level you can see here on the dashboard I have a current month. And you can see here on the dashboard I have my current monthly cost, as well as my forecasted month and costs. And in some cases, this might be all the information you need. But when you're dealing with many different AWS services, you might want to dive into it a little bit more deeply, and that's where Cost Explorer comes in. So here I can see my daily unblended costs, and I'm going to click the button to view in Cost Explorer.

So the first thing I'm going to do is I'm going to close the side panel here so we have a little bit more room to look at Cost Explorer. This is grouping my costs together on a daily basis, so I can see my daily costs across all AWS services. This is another thing that can be valuable, but a lot of times we're going to want to k what AWS service actually generated those charges. So you can choose to go in and group your cost by several different factors, but one of the things you'll often use is to group by service. And I can see that I'm generating these charges from services like EC2 and App Runner, as well as several others. I could choose to go through and look at more detailed information in the table below, but maybe I don't really want to k by service. Maybe I'm utilizing an AWS organization, which I am in this case, and I want to see each of the different accounts and how they're generating charges within consolidated billing.

So I can actually click out of Service, and instead, I can say group by Linked Account. And in this case, I can see the different accounts and which ones are actually generating charges.

This enables me to see a complete breakdown by the different accounts that I have. I can choose to group by many different things, including by region. I can do it by usage type. I can even do it by tags, which we talked about in a previously. And I can even choose to go through and change the date range and even change how they are grouped. But instead of going in and actually just generating these reports on the fly, I might want to take a look at some reports that AWS has already created for me. So I'm going to go back to AWS Cost Management, and from here I can click the option that says Reports in the left pane. I can see here that there are reports that are already generated, Monthly costs by service, Monthly costs by linked account. Let's just go ahead and take a look at Monthly costs by linked account. I can see here that there is a report that's already created for me so that I can see this breakdown by linked account month to month. So instead of simply having to go in and do this every time, I can utilize this report. I can even save my own reports and access them at a later time.

If I were to scroll down below the graph and see the table view, I can actually click the Download CSV button, and I can download the actual data that is being used to generate the graph. And in that way, I can do my own analysis on the data inside of separate tools. I can even send this information to someone else who doesn't have access to my billing dashboard if I wanted them to explore our costs within AWS.

Chapter 9 How to Apply Cloud Economics

So we're going to apply these principles of cloud economics by looking at some scenarios, and you can judge how well you've absorbed the material that has been covered within this chapter. So grab those guided notes, grab a pencil, and be ready to jot your answers down, and I'll be reviewing the answers for these scenarios next. So here is our first scenario, and here we're going to be talking about Oliver. And Oliver's company has multiple departments that work within AWS, and in this case, Finance is asking for a clean separation of AWS costs between departments. , currently, all resources are included within a single AWS account. So what approach would meet this need for future costs with minimal effort? Think about the answer for that one. And, let's look at scenario 2. So for scenario 2, Sophia's company is considering a transition to the cloud. They currently have two physical data centers that they own and maintain. Stakeholders are questioning whether this approach will save money. So which approach should Sophia take to make a case for the cloud?? Here, we have scenario 3. And in this case, we have Don, and he is a web developer at his company, and given some recent downtime, he is looking at moving there site to the cloud, but Finance is asking for an estimate of costs for this transition to AWS. So what approach should Don take to get this data to his finance team? So, next, we'll walk through the answers to these scenarios. Before we dive in and explore the answers to the scenarios, let's just take a step back, and let's take a look at all the things that we've been able to cover here within this chapter. So first of all, we understood funding differences between traditional data centers and the cloud. We also utilized AWS tools for cost organization. We then examined the AWS tools that you could use in making a case for moving to the cloud, and we actually explored real AWS costs using the tools. So, with all of that information behind us, let's take a look at the scenarios and their answers. So first, we have Oliver, and if you remember, he was looking to provide data to finance that was cleanly separated between departments, and he needed to do this with minimal effort. Well, what would he do in this case? In this case, he would create and leverage a resource tag for each department. And you might say, well, wouldn't they be more cleanly separated by using AWS Organizations? And that's a great point. But moving resources between different accounts would be a bit of additional effort required here, and he was looking to do something here with minimal effort. So in this case, the tag approach makes sense. And next, we have Sophia, and Sophia's company was considering a transition to the cloud, but she needed to make a case for the cloud, so what would she do here? Well, in this case, she could look at utilizing AWS Migration Hub or Migration Evaluator to build her business case for moving into the cloud. Last, we have Don, and if you remember, he needed to provide Finance with an estimate of cost for transitioning an existing workloads into the cloud. So how should he do this? Well, in this case, he can utilize the AWS Pricing Calculator and then share results with his finance team.

Chapter 10 How to Support AWS Infrastructure

So next, we're going to talk about how you support your AWS infrastructure once you transition some of your workloads to the cloud. So over the book of this chapter, here's what we're going to be covering. First of all, we'll be understanding the tools that are provided by AWS to support your workloads in the cloud, and we'll be reviewing the AWS Support plan tiers. Then, we'll be reviewing AWS Trusted Advisor recommendations and exploring the AWS Personal Health Dashboard. So next, let's talk about the tools that AWS provides to support your infrastructure. And anytime we're talking about support, we're really talking at a high level about AWS Support, which is the service that enables you to file support requests with AWS resources. However, within AWS Support, there are two additional services that are also made available to you. The first is the AWS Personal Health Dashboard, and the second is AWS Trusted Advisor. So let's take a look at each of these three services. So first of all, we're going to talk about AWS Support. So with AWS Support, it enables you to get support from AWS resources for your workloads that are running in the cloud, and it is provided in different tiers based on your need and the scope of your need. it includes tools to provide both automated answers and recommendations, and we'll look at some of those recommendations later within this chapter. The next service is the AWS Personal Health Dashboard, and it is going to provide both alerts and remediation guidance when AWS is experiencing events that may impact you. For example, if there is a partial outage of a service within a region, this is where you can gain insight on that and understand how this could affect your infrastructure. the next service is the AWS Trusted Advisor, and this is a powerful automated tool that enables you to check your AWS usage against best practices. So in many cases, if you are regularly implementing the recommendations that Trusted Advisor provides, you may even eliminate the need to file a support request. It can be accessed from the AWS Console, and different checks within Trusted Advisor are provided based on the AWS Support plan tier that you have, but all AWS customers, even ones without a paid support plan, get access to seven core checks within Trusted Advisor. the Trusted Advisor checks are really divided into five different categories. The first is cost optimization, the second is performance, the third is security, then we have fault tolerance, and finally, service limits. And so if you have a support plan that supports all of the checks, you can gain guidance in each of these areas for your workloads that are running on AWS. So now that we've discussed the different services that AWS provides to support your infrastructure in the cloud, we're going to break down the specific AWS Support plan tiers that are included. First, we need to understand the differences between the different Support plan tiers, and it really falls into four key areas. The first is communication method, the second is response time, the third is cost, and the fourth is the type of guidance offered. So let's look first at the support plan, AWS Basic Support. AWS Basic Support is provided for all AWS customers, and you have access to Trusted Advisor, to the seven core checks within Trusted Advisor. You also get 24x7 access to customer service, documentation, forums, and white papers; however, you'll need to note here that you don't have access to AWS support engineers for your technical implementation questions. You do, however, get access to AWS Personal Health Dashboard, and with this plan there is no monthly cost. However, if we move up a level, we get to AWS Developer Support, and this is targeted at an individual developer that's looking to get support for running their workloads in AWS. It does include all the features of Basic Support, but you also gain business hours email access to support engineers. However, we'll talk in a minute about the response times between the different Support plans. Developer Support is limited to one primary contact, so that individual needs to be the one to actually file the support requests. It starts at $29 per month, but it is tied to a percentage of AWS usage. So if you have a lot of AWS resources that are currently deployed, you may end up paying more than $29 per month. Next, we have AWS Business Support, and this includes all of the features of Developer Support, but it also adds in a full set of Trusted Advisor checks. It goes quite a bit beyond the seven core checks that are just provided with Basic and Developer Support. You hear have 24x7 phone, email, and chat access to support engineers. If you want to be able to pick up a phone and actually call an engineer, this becomes the first tier where you are able to do that, but you also have access here to email and chat. Next, we have unlimited contacts. So this is not just limited to potentially your root user. It allows you to let all of your resources have access to the Support Center and file support requests. it also provides software support. So if you're in a scenario where you want help in deploying a piece of software onto AWS, this is the tier where you can start to get that access. it starts at $100 per month, but just as the previous plan, it is tied to your AWS usage. So with more AWS usage, you would end up paying more than just $100 per month. The last support plan is AWS Enterprise Support, and it includes all of the features of Business Support. It also includes a designated technical account manager, so a single individual that can become your point of contact for the account. It also includes a concierge support team. This starts at $15,000 per month, but it is also tied to your AWS usage. Next, let's review one of the most important aspects of comparing these different support plans, and that's the support response times. So we have five different categories that are provided by AWS in terms of their response time, General guidance, when we simply have questions that we need to have answered; System impaired, where something is not working as it should; Production system impaired, when we actually have a production system that is not performing at its desired capacity; Production system down, meaning that we actually have a production system that is completely and finally, system down, meaning that this is a core system for our organization, and it is completely down. So in terms of the Developer Support plan, we're able to provide support response times across two of these categories. So within 24 business hours, we should be able to receive a response for General guidance and 12 business hours for System impaired. With the Business Support plan, the first thing you'll notice is we're dropping the business hours designation. We're actually talking here about just hours. So within 24 hours, we should have

General guidance questions answered. Within 12 hours, we should have a System impaired response. Within 4 hours, for impaired, and 1 hour for Production system down. But when we transition to Enterprise, we're going to see this expand all the way down to our fifth category. So within General guidance, again, 24 hours; with System impaired, 12 hours; Production system impaired, 4 hours; Production system down is 1 hour, but for system down, here we're able to see that there is a response time provided if you have the Enterprise Support plan.

Chapter 11 AWS Support Tools

Now that we've reviewed AWS Support, we're actually going to dive into the AWS Console and review some of the services that are provided as a part of AWS Support. We are first going to be accessing AWS Trusted Advisor in the console, and then we'll be reviewing the AWS Trusted Advisor recommendations. Following this, we'll be accessing the AWS Personal Health Dashboard and then reviewing the information provided in the Personal Health Dashboard. I want to point out a few things. With the widget configuration that we have, we're going to see little bits of information from the tools that we're looking for just here on the home page of the console. For example, we can see our personalized health view here under age AWS Health, and we can go directly to there from this link.

And we also can see Trusted Advisor, and we can click on it to go directly to Trusted Advisor from here. But I want to show you another way that you can navigate through the console. You can simply type in the search bar above.

So first, I'm going to search for Trusted Advisor, so I'll type in trusted, and I can click here on this service.

So here I can see that there are some recommendations already. But before we get to that, I want to point out here on the left pane you can see that we have the five categories that we mentioned earlier, Cost optimization, Performance, Security, Fault tolerance, and Service limits. Because here we're simply using a Basic Support plan, we are only going to see recommendations that are included under Security and Service limits. But if we were to upgrade the plan, then we would get more checks.

However, we'll be able to see what some of those checks would be when we're examining Trusted Advisor here. So first of all, you can see that we do have a recommended action, that we need to set up authentication on the root account. And so we can actually see how we go through this process by simply following the steps here within Trusted Advisor. We can also see here there are just some areas that would require some investigation, maybe not immediate action, and we can see here that we're discouraging the use of the root user access, and so they want to see that we have at least one IAM user, just for example.

However, you can see here at the bottom if we did upgrade our AWS Support plan, we could get access to all Trusted Advisor checks. Let me go through here, for example, and click on Security. Here you can see the things that it is checking for our account, things that are included in those core checks. But if I scroll down here, we'll see that we can see the items that it isn't currently checking because of the Support plan that we have. So you can begin to understand the full scope of the capabilities that can be provided by Trusted Advisor if you did happen to have a bigger support plan than the one we currently have configured on this account. So I'm going to go ahead and scroll back to the top, and from here you notice we also have the ability to actually download those checks. So if we don't simply want to use them here in the console, but we want to save them for later, we can get that information here directly from the console. I'm going to move away from Trusted Advisor, and I'm going to go back here to the home page of the console. I could just click on AWS Health Dashboard here under my Recently visited services, but again, I want to help you navigate through the console, so here we're going to type in personal health, and we can see that the AWS Health Dashboard shows up here at the top of the list.

And you can see here that for this particular account, there have been no recent issues, and that makes sense, to be honest, because I haven't configured any resources yet within this account.

However, we can see some of the issues that have happened recently under the Event log. For example, I can see here that there was an operational issue with EC2 in Northern Virginia, which is the region.

So I'm going to go ahead and click on this issue, and here we can gain a summary of when the issue was reported and when it ended. We can see the summary of what actually was happening here. So for example, we can see that some EBS volumes, which stands for Elastic Block Store volumes, experienced degraded IO performance within the region. The issue has been resolved and the service is operating normally. So this is very beneficial information if you're utilizing the service and you want to k why something isn't working exactly as it should have been working. And so you get all of this information here within the Health Dashboard. And there's a lot of different ways you can get access to this information. This can even show up on the home page of the console when you log in under the widget for your AWS health. So hopefully here you've been able to see how we can leverage multiple tools that are included within the AWS Console to support our use of the platform

Chapter 12 Infrastructure Support Scenarios

When organizations make the shift into the cloud, they quickly find that they're going to need some help because in most cases they really want to answer the question, are we doing it right? And sometimes that's more than can be handled in just a support request with an AWS Support plan. And so there are three resources that can be very helpful for organizations that are in this position, and to be honest, these are also resources you should be familiar with for the exam. So the first resource for organizations that are looking to answer that question about whether or not they're doing it right is the AWS Quick Starts. Each Quick Start provides deployment instructions for a common technology platform into AWS. This is a great place for organizations that are using a standard platform and just want to know if they're deploying it in the right way. But in many cases, organizations are also going to want to bring in subject matter experts to come in and help with their migration to AWS. And if you want to find trusted third parties that have been vetted by AWS through their partner program, you can take advantage of the AWS Partner Network Consulting Partners. These people don't work for AWS, but they do work for a consulting company that has been vetted. And next, if you want to work directly with AWS resources, you can choose to take advantage of the AWS Professional Services team. So here, these are three resources that can help organizations move forward with their migration into the cloud. So next, we're going to be taking the knowledge that we've gained in this chapter in terms of AWS Support, and we're going to be applying it to some scenarios. And our first scenario here has to do with Kate, and Kate's company is in the process of moving multiple workloads into AWS. One of these workloads is a application, one that they want to be sure they maximize the uptime of. Her CTO has given her some specific guidance in saying that they need to be able to call support 24 hours a day. With all of these criteria here, the question that he is asking her is, what is the most support plan that meets this criteria? Next, let's look at Danny, and his company is evaluating AWS for some future workloads. One of these future workloads that they're planning would be supporting multiple offices globally, so it would be in use 24 hours a day, and the company wants to be able to call, text, or email support if any issue occurs. Because this is going to be so for their company globally, they want to be able to get a response within 15 minutes if there is a complete outage of this infrastructure. So for Danny and his company, what is the most support plan that meets the criteria? We're going to be talking about Don. And we've talked about Don before. He is a web developer at a company. But in this case, we're going to be talking about Don's personal account that he is using for a personal project on AWS. Because in his day job he works in AWS, he doesn't expect he's going to need technical guidance from AWS; however, he does want access to AWS Trusted Advisor and the core checks that are provided. So for Don, what is the most support plan that meets his criteria? So we're going to take a minute and review what we have covered within this chapter. And in addition to that, we're going to go through and talk about the solutions to the scenarios that we introduced before. We started off by understanding the tools that are provided by AWS to support workloads in the cloud, and then we reviewed the specific AWS Support plant ears that they provide. We then reviewed AWS Trusted Advisor recommendations, and we looked at the different categories of recommendations that are provided. And we also looked at the difference between the Trusted Advisor checks that are included within the Basic and Developer Support versus the Business and Enterprise Support tiers. And then we explored the AWS Personal Health Dashboard. Let's dive into our scenarios. Scenario 1 had Kate needing to recommend a support plan for her company, and we were specifically looking at the most support plan for her criteria. And the answer to this one is Business Support. But let's talk about why. So first of all, her CTO said that they needed to be able to call support. So first off, we k that if we're calling support, we have either a Business or an Enterprise Support plan because the Developer plan only includes email access, and the Basic Support plan does not include access to Support Engineers for technical implementation questions. So because of this, we k we are immediately dealing with one of those two support plans. It says that she needs to be able to call support 24 hours a day. Well, in this case, that works for both Business and for Enterprise. So, when we're looking at this, if we're looking at the most support plan, we k that Business Support would be more than Enterprise Support, and in this case, she doesn't have any requirements that would only be met by Enterprise Support. So next, let's look at Danny. So Danny's company, again, is evaluating AWS for future workloads.

So for him, what should he recommend? Enterprise Support. Let's talk about why. First of all, if we look at it, it says the company needs to be able to call, text, or email support if an issue occurs. Just as before, we k that this means that we are looking at either the Business or the Enterprise Support plan. However, the key here is the company wants a response from an issue within 15 minutes if there is an outage. That means that the Enterprise Support plan is the only option because that's going to give us that increment if we have a system down. Next, let's talk about Don, and Don has an AWS account for a personal project, which is separate from what he's doing for work, and he doesn't think he's going to need technical guidance, but he does want access to the AWS Trusted Advisor core checks. So what would his support plan be? Well, in this case, we're going to say Basic Support, so the general support that is provided for anyone that has an AWS account. This includes access to the seven core checks that are provided by AWS Trusted Advisor. It doesn't provide technical implementation assistance. He can't email a support engineer. However, in this case, he doesn't believe he's going to need it, so he can choose to leverage Basic Support until he has a need to use one of the paid support options.

Chapter 13 How to Prepare for the Exam

Let's now talk about what comes next and how you get ready for the exam. The first thing you'll need to do is you'll need to make sure that you don't sign up for your certification exam until you've completed all our books in this learning path. Later on in this path, we'll walk you through the entire process of getting signed up for the exam and how to get ready for taking the exam. That being said, what you can do is you can go through and study the guided notes that you've taken throughout this book. That will give you the material that you need to get ready for the exam, and then you're going to be ready to move on to the next book, Understanding AWS Core Services. And please don't skip the last book in this path. This is truly a differentiator where it will walk you through how to get ready for the exam and how to think about the questions that you'll find on the exam, and it will walk you through two complete sets of questions to help you evaluate whether or not you're ready to take the exam. So, I'm excited for you to go ahead and take your next step on your journey to becoming an AWS Certified Cloud Practitioner.

BOOK 6

AWS CLOUD COMPUTING

INTRODUCTION TO CORE SERVICES

RICHIE MILLER

Introduction

This is the second book designed to get you ready for the exam. So hopefully by this point, you've already had a chance to go through the Fundamental Concepts book. We're going to be diving deep and understanding many of the AWS services that are covered on the exam, but this is a lot of information to remember and so that we have a good understanding of AWS as a whole, we're going to be looking at how we leverage the services that are included within the platform. And when we're talking about interacting with those services, we're really looking at three different approaches. The first of these is the AWS Console, and we used this when we set up our initial AWS account in a previous book within this path. In addition to that, we also have the AWS CLI, or Command Line Interface, and this is what allows you to access those same AWS services, but just from the command line on your machine. If you want to program access to your AWS resources, this is when you would look to leverage the AWS SDK, which is supported across a variety of languages. Let’s take a look at each of these in turn. First of all, we have the AWS Management Console, and, we have used the browser version of this earlier in this path. But there also is a mobile application that goes along with this. It provides access to most all of the 150+ AWS services, and in this case, all major browsers and mobile operating systems are supported. You'll probably primarily use the interface, but it wouldn't hurt to download the mobile version and explore that on your own. So here's an example of the browser version of the console. We can see here that we're logged in, and we can see that I'm using the Northern Virginia region.

And you can see a list of some recently visited services that I have had here on this version of the console. We'll dive into the console later on within this chapter. Next, we have the AWS Command Line Interface, or CLI, and this is a tool that allows you to manage your services from the command line, and this works on Windows, Mac, and Linux. And most everything you could do in the browser with the console, you can do with the CLI. So as an example, if we wanted to list the current users for our specific AWS account, we could run the command aws iam For those of you that maybe have some experience with the platform, you're noting that this is just limited to IAM users and not the root user, which is totally true. And we'll cover more of that in a later book within this path. The next thing we have is the AWS Software Developer Kit, or SDK. And this is what allows us to use a programming language to actually script the process of how we interact with AWS. This is great because it can allow us to automate many things within the platform. And as we look at the different languages that are supported, we do have quite a range here, and this includes Java, .NET, Node.js, JavaScript in the browser, PHP, Python, as well as Ruby, Go, and C++. For most developers, you're working in one of these languages, and that is supported with an AWS SDK. Let's talk about when you would use one of these approaches versus another. So, first of all, the console is a great method for testing out AWS services. So if you want to leverage a service for the first time, if you want to spin up a virtual server and maybe you haven't used EC2 before, or maybe you're looking at testing out a new service that AWS just announced, the console is a great way to do this. However, if you have repeated tasks, you might want to look at using either the CLI or the SDK. If we require a person to log into their browser and go and perform tasks, that can be very cumbersome. But if we're going to be doing these tasks continually, this becomes a great case for automation, and both the CLI and SDK can enable that automation. And next, the SDK enables automation of AWS tasks within custom applications. So if you have a custom application that you're running on AWS and you want it to automate some way of interacting with an AWS service, the SDK becomes a great choice for that. It's important to note here, most all services and actions can be performed in any of these three. There are some exceptions to that rule, and we'll cover a few of those within this book. But at a high level, just k that for most services, you'll be able to use either the console, or the CLI, or the SDK to perform whatever tasks you need when using these services.

Chapter 1 How to Use the AWS Console

Now, we're going to look at how we leverage the AWS Console, and we're specifically looking at the browser version of this and not the mobile app version of the console. We're first going to be accessing the AWS Console in the browser, and then we'll be talking about the login differences between root and IAM users. We'll be introducing this concept. Then, we'll look at how we select a specific region, and then we will be reviewing the list of services that are supported within the AWS Console. It's important to note that the root user is a special kind of user. It was the user that was created when you initially created the AWS account, and it has special permissions that no other account has, including things like selecting a support plan and deleting the account. In this case, if you're here, you need to be sure that you're using the root user to log in. There is another type of user that can access the AWS Console, and that is an IAM user, and we'll talk more about IAM later. However, in this case, I'm going to be able to log in, I'm going to take my password, I'll sign in, and then here I have it currently set up for authentication, and that means I'm going to have to enter in a special code in addition to my password to be able to log in. So I'll enter in the code, and I'm logged in to the AWS Console.

At a high level, there's a few things I want to point out. One of them is here, you can clearly see my user information, and if I pull down on this, I'll be able to go in and look at the information for this account, including the account, the organization, which we've talked some about, as well as the billing dashboard.

I can see all of my invoices, and I can go look at my security credentials if I need to change those. In addition to this, we also have a next to that for selecting the AWS region. So here, I've currently selected N. Virginia, which is but we can see that there are many different regions, and we've already talked about the AWS global infrastructure and the different regions that are supported.

But in this case, if I were to switch over and select any of the resources that I launched here within the console would be launched within the Ohio region as opposed to the N. Virginia region. For most all services, this matters. There are a few services that don't, and we'll talk about those shortly. , the next thing I want to do is I want to look at the Services From here, you can see a list of all AWS services, and they're currently grouped according to category. For example, you can see the Compute category, which has EC2 and Lightsail, and ECR, and ECS, and EKS, and several more, and we'll talk more about those in an upcoming chapter. But this gives us a high level way of looking at the services based on their category.

However, we can also choose to go in and search for a service. For example, if I wanted to look at a service like Lambda, I could simply type that in, and you can see a list here of services that include Lambda. In this case, we would be looking here just for Lambda. And if we click on it, we would go into the console for this specific service.

Next, we can also go back to the home page and there also is a search box that is included here within the home page of the console. And so we could go in here and we could type in a service like rds, which is Relational Database Service. In this case, we also can look at recent services that we have launched here within the console, which include Lambda, and Trusted Adviser, and Cost Explorer. And so, if you're working within a small handful of services, this can be a very quick way to simply click on these and actually launch into those specific services. I'm going to search for a service called Route 53, and I'll click on Route 53.

One of the things that I want to point out here as we go into this service, and this is AWS' DNS service, is that this is one of the few services where the region doesn't matter. You'll notice here that it says global as opposed to region. And it says here that Route 53 does not require region selection.

So you can note when you're working with services whether or not they are global services or whether or not they are services. So I'll navigate back to the home page of the console. And the last thing that I want to point out is we can go here within our user and go to Security Credentials.

So here, we can go in as the root user, and we can update our password. You can turn on authentication, which we'll do later. You also can configure access keys, and access keys will be where we pick up our next demo when we're looking at leveraging the SDK and the CLI.

Chapter 2 How to Use the AWS CLI

So next, we're going to be talking about using the AWS CLI. I need to make a note here. You don't need to k how to install and configure the CLI for the Certified Cloud Practitioner exam. You do need to k what it is and why you would use it. However, it's a valuable tool if you'll be working within AWS, so we're going to do a quick review here. We will first be accessing AWS access keys. We'll then be reviewing installation instructions for the CLI. We'll then be configuring the CLI for a specific user. And finally, we will be interacting with AWS utilizing the CLI. So, I'm here at your security credentials, and this is where we left off when we were demoing the console. Just as a reminder, you can get here by going under your username and then going into My Security Credentials. I mentioned that access keys were critical. This is basically the way that we control authentication and authorization when we are using the CLI and the SDK. I'm going to select Access keys, and AWS is going to present me with a warning, and rightly so. They let us know that root users have unrestricted access to the entire AWS account.

They recommend using an IAM user with limited permissions if we're going to be creating access keys, and that is great advice. , as a note, we will be creating access keys here. However, I am going to be deleting these immediately following this demo. You should not create root user access keys and use them on a basis. That is not a good practice. However, for simply demoing the CLI, in this case, that should be fine. So I'm going to select Create New Access Key, and it lets us know here that it has created our access key.

And it lets us k here that this is our one chance to see this information, and we need to download our key file. It will not show it to us again. If I click here on Show Access Key, we can see here, both our access and secret access key, and this is the information that we will need to be able to access our AWS resources from the CLI. Before we use these keys, I'm going to navigate over to the installation instructions. So here within the AWS documentation, there are instructions for installing the CLI. There are currently two versions of the CLI, version 1 and version 2. Version 2 is a newer version that is just currently in preview, so for, we'll look at version 1.

Depending on your platform, you can install the CLI in different ways. For example, Windows has an installer that actually comes with Python integrated in. Python is what the CLI is actually built on and what is required. So in Windows, there's a single installer that will handle it for you. For many users that are on Linux or Mac, Python is already included on your machine, and you can simply use tools like pip to install the CLI. Next, I'm going to navigate over to the terminal so we can actually use the access keys that we've created. So from here, I can first check to see if the AWS CLI has been properly installed on my machine. I can simply type aws and I can see that I'm using the AWS CLI version 1.16.180, so I do have it properly installed on my machine.

The next step is going to be to configure it with the access keys that I just created. The first step will be to enter in the access key. Next, I'm going to be entering in the secret access key. Once we have that in place, we can enter in the default region name, and in this case I'm going to choose Finally, we can choose the default output format. In this case, I'm just going to choose JSON because that's a format that I'll work with. Once I have all of that in place, I'm going to hit Enter. And we have configured the AWS CLI to work with those credentials in a profile called PS test. So, I should be able to utilize the AWS CLI to get information about the AWS resources that I'm leveraging. In this case, I'm going to run a command that's going to list the S3 buckets that I have created within this account. And here you can see it has returned to me a list of all of the different S3 buckets that I have created within this account.

So, to quickly review what we've done. We have gone in and created access keys for our user with a caveat there that we don't generally want to create access keys for the root user, but for simple testing purposes, as long as you delete them after the fact, that should be fine. Then, we went through the process of looking at how you would go about installing the AWS CLI. And we noted that there are different instructions for Windows, Mac, and Linux users. We then configured the AWS CLI to use the credentials that we had created. And finally, we verified that we could access the AWS resources that we're leveraging, utilizing those credentials from the CLI. So next, we're going to help you gauge how well you understand the material that has been presented in this chapter by looking at three different scenarios. We're going to take time and review each of these scenarios here. Be sure, in your guided outline, to write down the notes about what you think the answer is for each scenario. Next, we will be walking through the answers to each scenario. So here's our first one. We're going to start off with Rob, and his company runs several production workloads in AWS. They have a new web application that manages digital assets for their marketing team. And what they want to do is they want to automatically create a user account in Amazon Cognito when the user signs up through their custom application. In this case, Amazon Cognito, we haven't covered yet, but it is basically a user directory for our custom applications. And what they want is they want this to seamlessly integrate into the application that they have, and the application that they have is a web application that actually has some microservices on the back end that are written in Node.js. In this case, which interaction method would Rob's company use for this? Next, we're going to look at Elenor, and her company is considering moving to AWS, and they want to leverage a service called Amazon Relational Database Service, and she just wants to test out a single database on the service. She hasn't ever used it before, so she just wants to get in and understand how it works. So in this case, what interaction method would Elenor use for this use case? Next we're going to look at Judit, and her company is a startup, and they created a social network for entrepreneurs that has both a web and a mobile app. , she has a set of tasks that she needs to run each day to help generate some reports on usage, so in her case, what interaction method would she use for this use case? Next, we'll run through and look at the answers. Before we dive in and take a look at the different scenarios that were presented previously, let's take a minute and look back at what we have covered in this initial chapter in this book. First of all, we reviewed the ways you interact with AWS services. We talked about leveraging the console, the CLI, and the SDKs. We then looked specifically at the AWS Console and its use. We even went through a demo to highlight key areas of the console. , with this as well, we talked about the fact that the console is primarily a experience, but there also is a mobile application that you can leverage. We then introduced the AWS Command Line Interface, or CLI, and its use. We also did a quick demo to showcase how you would configure the CLI with your credentials. Then we introduced the AWS Software Developer Kits, and we talked through the different programming languages that are supported with the SDKs. Let's take a look at our scenario. Here in this first scenario, we talked about Rob, and he wanted to seamlessly integrate Amazon Cognito into his custom application. Well, in this case, the solution is going to be the SDK, or Software Developer Kit. The reason is because Rob wants to bake this into his custom application. So, because he wants to bake it into his custom application, if you remember, we talked about some services that were using Node.js, in this case, he can choose to leverage the SDK for Node.js to interact with Amazon Cognito within his web application. So that's scenario one. Let's look next at scenario two. And we were talking about Elenor, and she wanted to leverage Amazon Relational Database Service, or RDS. What should she use? Well, in this case, using the AWS Console makes sense. Elenor is not looking to automate this. She's not looking to create this for production. She just wants to get in and use the service and understand how it works. So this seems to be a great fit for using the console. Next, let's look at Judit, and when we were talking about Judit's specific scenario, she had tasks that she performs on a daily basis that she needs to utilize something to automate. And so what interaction method would she use? Well, in this case, we're going to say the CLI. However, you could also say the SDK, in this case. She could build it out either way. But what she doesn't need to do is go in and perform the same tasks in the console day in and day out. There's a more efficient way to do it using one of the two different approaches for automation, either the CLI, where she could write a custom script to do these things for her, or bake it into a custom application utilizing one of the supported programming languages with the SDK.

Chapter 3 Amazon Compute Services

Now, we're going to be talking through compute services on AWS, and we're going to be focused on the compute services that will be covered in the Certified Cloud Practitioner exam. At a high level a compute service is just a service that enables you to leverage virtual machines for your workloads. And this could be a lot of different things, running a database, having a web server, or data processing. We're going to primarily be talking about three different compute services on AWS. First, we're going to be looking at Amazon EC2, which is a foundational core service on AWS, and this is what enables us to run virtual servers on AWS. Then, we'll be looking at Elastic Beanstalk, which is a platform, so this is platform as a service for scaling and deploying web apps and services. And then, we're going to be looking at AWS Lambda, which enables compute, but without having to manage any servers. First of all, we're going to be introducing Amazon EC2 capabilities, and then alongside that, we'll be exploring the different pricing approaches for EC2 instances because there are a lot of options to consider here. Then, we'll be introducing the capabilities of AWS Elastic Beanstalk and then reviewing use cases for Elastic Beanstalk. And finally we'll be introducing AWS Lambda. So, next we're going to talk about Amazon EC2. And Amazon EC2 is one of the fundamental core services of AWS. And AWS defines it as a Web service that provides resizable compute capacity in the cloud, and it's designed to make computing easier for developers. So, before we get too deep into EC2, let's look at some sample use cases where we could choose to use EC2. Let's say, for example, that we want to host a web application in the cloud. We could choose to an EC2 instance, and then on that instance, we could install a Web server, and then on that web server, we could go in and put the files for our web application. So, in that case, EC2 would totally meet that need. But we also could do things like batch processing of data. So if your organization produces maybe a 1,000,000 rows of point of sale data each day and you want to pull that in and do some preprocessing before you analyze it, you could use an EC2 server for that. You could also use it to just be an API server, where you take web services and launch them in the cloud and let other applications access them. Or you could do something like, even having a desktop in the cloud. So if you want to launch a windows instance on Amazon EC2 and then be able to connect to that through remote desktop, all of those things are possible utilizing Amazon EC2. Next, we're going to talk about the core concepts that we need to k before we're able to launch an EC2 instance in the cloud. First, we need to understand instance types, and then we need to understand root device type. Next, we're going to look at the Amazon Machine Image, or AMI, and then we're going to look at the different purchase options for EC2 instances. But let's start off by looking at instance types. So, an EC2 instance type defines the processor memory and storage that are available to any servers that are launched with that instance type, and you can't change this without downtime. So, if you've launched an EC2 instance, which is your virtual server, and it has a certain amount of memory, based on the instance type that you chose when you launched it, you can't simply go to that server and say, I want to put a new instance type in. So, we do want to make sure that we make good choices on instance types when we launch them. We have instance types across several different categories, including general purpose. So, for most workloads that you're going to put in the cloud, general purpose will probably work fine for you, but we also have compute, memory and storage, optimize instance types. And again, that's three separate categories there.

So, for example, if we wanted to launch an database on our EC2 instance, we might choose a memory optimize instance type. But we also have accelerated computing, and these are for specialized use cases, for example, machine learning, because in this case, with those instance types, you could get access to a GPU, for example, which could be very important for the different machine learning work that you're doing. , pricing is based on the instance type. So if you choose something that has more resources available to it, and it's a specialized type, it will cost more than just using a general purpose instance. But some instance types also have unique capabilities, so some families of instance types have access to things like specialized storage, or GPUs, as we mentioned earlier. So next, we're going to look at some sample EC2 instance type pricing. , I need to make a couple of caveats here. One is, is that EC2 pricing changes over time, so these numbers might not be accurate in the future, but in addition to that, the prices can also change from region to region. So let's just use these prices I'm showing here, even though I pulled them from let's just use them to help us understand relatively what different instance types cost. So I'm going to start off here with a couple of different general purpose instance types. You can see here that we have a t3.medium and an m5.large. Well, we can see that the m5.large has twice as many vCPUs and four times as much memory, and in this case, it's just a little over twice as much per hour, in this case, coming out to be just a little bit under 10 cents per hour. Here is an example of a compute optimized instance type, a c5d.24xlarge. We can see here that we have a dramatic jump in all three categories, vCPUs, memory, and pricing, with this coming out to be about $4.60 per hour. But then we have a p3.16xlarge. , this is a very specialized instance type that comes from the accelerated computing category because this gives us access to some of the industry leading GPUs for things like our machine learning workflows. And in this case, this one comes out to be about $25 per hour, so, even though it looks here, like it has less vCPUs, these factors into some of those special capabilities that are included within the P3 instance type family. And then, we have a storage optimized instance type, an i3.16xlarge, which again has 64 vCPUs and a lot of memory, and it comes out to be about $5 per hour, but it has some special access to storage capabilities that the other instance types don't have. Next, let's move away from instance types, and let's take a look at the root device type. There really are two different root device types that you need to k when working with EC2. The first is what we call the instant store, which is ephemeral storage, and it is actually physically attached to the host that the virtual server is running on. So that's one type. And the next type is Elastic Block Store, or EBS, which is persistent storage that exists separately from the host that the virtual server is running on. , when EC2 initially launched, we just had instance store, but that we have EBS, I'll go ahead and tell you that for most work you're going to do on EC2, you want to use Elastic Block Store, which is EBS, unless you have a specific reason not to. It provides a lot of capabilities that will help you, but one of the key differences is the difference between the words ephemeral and persistent. So with an instant store, if you your EC2 server and you actually completely shut it down, then the data on that instant store will go away. However, EBS data will be persistent, and we can even go in and take snapshots of it and copy it and launch new EC2 instances with EBS volume. There's a lot that we can do, and we'll cover that more, later within this book, when we talk about the different storage services that are available on AWS. But just know here, that instant store is ephemeral, meaning that if you shut down the server, that data will go away, and EBS data is persistent, meaning that if you shut down the server, that data is still there. Let's look next at Amazon Machine Images, or AMIs. , I should probably also mention here, that there is fierce debate amongst the AWS community on whether or not this is pronounced or Amy, but in this case, you'll hear me refer to it as an But just know if you hear somebody else refer to it as an Amy, that's what they're referring to. , an AMI is a template for an EC2 instance that includes configuration, the operating system, and the data that actually would go on that specific instance. And AWS provides many different AMIs that you can leverage, and when we go through, actually using EC2, later within this chapter, you'll see that there are several provided by AWS that are relatively easy to a new instance from. But AMIs can also be shared across accounts. So if your organization has a specific version of, let's say, Ubuntu Linux that you want to modify in a certain way for security purposes and you want to have it be just exactly the specific way and you want it to have one extra drive that's attached where you store other pieces of data, those are all examples of configurations that you can make within an Amazon Machine Image. And, you can create your own custom AMIs, and there also is a marketplace for commercial AMIs. So if you want to use AWS Marketplace, you can go and explore the different AMIs that are provided from commercial vendors.

Chapter 4 Amazon EC2 Purchase Types

Now we're going to talk through the different purchase types that you get when you're working with EC2. So first of all, we have And if you just launch an instance by default, this is what you're going to get. We then have the option for Reserved Instances. Then we have Savings Plans, which was a newer option. Then we have Spot Instances and Dedicated Host Instances. So let's understand each of these options and when you would want to use them. So, first of all, we have Reserved Instances, and what this does is this provides discounts over that model, which is going to be your default model when you can commit to a specific period of time, which is going to be one or three years. In addition, it also provides a capacity reservation for the specific instance type that you specify. So you can make sure that when you go to launch that that that's going to be there for the entire period of time that you sign up for. Within Reserved Instances, there are multiple types, and let's quickly review these. , as a note here, you're not going to need to dive super deep into each of these for the exam, but I do think it's important to understand the different approaches that you can take. So first we have Standard, and this gives you the highest discount, and it works well for steady workloads. But, if you have a Standard Reserved

Instance, what that means is is that if you want to get a bigger instance type, you can't make that change, you're locked in for that period of either one or three years. And that's why we have a Convertible Reserved Instance. And with this, this does allow us to convert some of the attributes if it's going to be of equal value to what we already have. This is also great for steady workloads. , in some situations you're going to have predictable, but not steady workloads, and if that's the case, you can create a Scheduled Reserved Instance. So this is able to have a time window attached to it, and so if you know, for example, that during weekdays for a specific period of time you're going to have a lot of usage and you want to get a Reserved Instance for that, the Scheduled type would work well. Let's look at just the Standard Reserved Instance cost model. So, with this you have some choices. You can specify that you want to pay this all up front. So you can say if it's one or three years, you want to pay all of it, and this is going to give you the highest level of savings, but it's also going to require the biggest upfront investment for that instance. You can then do a partial up front, and this is going to enable you to pay part of that one or period up front, and then you'll have a reduced monthly cost. But even if you don't choose either of those, if you take a no upfront approach, you're not going to have to pay anything up front, but you'll still get a reduced monthly cost over the option. So here's the way to look at it. You need to be able to evaluate a situation and determine which of these would make the most sense. Next we have the option that is the Savings Plans option. This is very similar in concept to Reserved Instances, but one of the key differences is that it's not just limited to EC2, it supports compute with EC2, Fargate, and AWS Lambda. But unlike Reserved Instances, it does not reserve capacity. That being said, there's still a way you can reserve capacity with savings, but it's going to be in addition to what you do with Savings Plans, and it can provide savings of up to 72%. And just as with Reserved Instances, it comes in one or terms. However, if you want to get the biggest price reduction, you can look at Spot Instances, and this really enables you to leverage the excess EC2 compute capacity that might exist within an availability zone. So let's quickly look at some notes here on Spot Instances. First of all, it can provide up to 90% in discounts over pricing. With this, there is a market price for instance types per availability zone, and this is called the Spot price. So think of this almost like a stock market for excess compute capacity. , when you request instances, if your bid is higher than the Spot price, then you'll be able to launch your instances. If the Spot price grows to exceed your bid, then your instance is going to be terminated. It's not instant, you do get 2 minutes prior to termination. However, what this means is this will only work for workloads that can start and stop without affecting what you're trying to do. So if that's your use case, you can get great savings by using Spot Instances. Next, we have the Dedicated Host option, and this gives you a dedicated physical server in the data center. This is going to be your most expensive option, but there's two use cases that you need to know that would require this approach. The first is going to be if you have a licensing model and you want to be sure you're abiding by the terms of that license. In that case, you'll need to use this option. Also for some compliance requirements, it requires that you utilize a dedicated host, and if that's the case you'll also need to use this pricing model. So let's quickly review the different options. First of all, if you have an instance that is consistent and always needed, you should purchase a Standard or Convertible Reserved Instance. Next, if you have batch processing where you can start and stop without affecting the overall job, this is where you should look to leverage Spot Instances because this is going to give you the maximum savings. So next, if you have an inconsistent need for instances and you can't stop it without affecting the overall job, this is where you want to look at just the Standard Instances. A few other notes here - if you have specific licensing or if you have a compliance requirement for a dedicated server, you should use Dedicated Host. And also here if you're leveraging Lambda and/or Fargate alongside EC2 and want to achieve discounts for a one or period, this is where you can choose a Savings Plan. But if you have predictable, but not steady workloads in EC2, you should purchase a Scheduled Reserved Instance. Let’s look at some examples specific to Reserved Instances.

So first, let's say that we have a t3.medium. So we can see here it's a little over $.04 per hour. If we look to purchase an EC2 Reserved Instance, if we do all upfront for one year, we pay $213. If we look at the effective hourly rate, that's going to be about $.02.4. That's going to save us about $150. And if we look at the same thing for three years, we're going to have the ability to save over $680. So here, our savings are only going to increase as we move to bigger instance types. So here, if we look at a compute optimized instance like a c5d.24xlarge, the rate is going to be $4.60, almost $.61 cents per hour. If we look here at just a partial upfront for only a year, what we're going to do is we're going to pay $12,000 plus a little bit more up front, and then we're going to pay about $1000 monthly. What this does is this lowers the effective hourly rate to $2.76 a little bit more per hour. That gives us about 40% savings. We can look also here at an i3.16xlarge. We can see here that the rate is almost $5 and we're able to achieve 52% savings. So even though the cost is high here, if you need this specific instance type, this approach is going to give you a lot of savings if you can commit to the one or period. If we look here at spot instances, this is where we're going to see even more savings.

So let's take a look at our t3.medium that we mentioned earlier. So is going to be about $.04 per hour, and with Spot pricing we can see this get down to maybe a little bit over $.01 per hour. As a reminder, this is a market price, so this is going to change. This is when I was actually looking at the pricing when I was creating, here's what you could get this for. You can see here, this is going to give us about 70% savings. We can also look at our c5d.24xlarge. Here we're going to see 80% savings. And we also can see 70% if we're looking at our i3.16xlarge. So you can see here that there are huge benefits to exploring the different pricing models that you can leverage with Amazon EC2.

Chapter 5 How to Launch EC2 Instances

Now that we've covered the core concepts that you need to launch an EC2 instance, we're actually going to dive into the console and launch an instance ourselves. I do want to here, you need to understand EC2 for the exam. You don't necessarily need to k how to launch an instance yourself, although this will be valuable in understanding the concepts that have already been presented within this chapter.

So here's what we're going to do. We're, first, going to launch a new EC2 instance based on an AMI provided by AWS. And then, we're going to be exploring the launch wizard in the AWS Console. We'll then be configuring an EC2 instance to be used as a web server. And finally, we will cover how you terminate an EC2 instance once you're done.

So, I'm here within the AWS Console. I've already logged in as my user and here, from the search box, I'm going to search for EC2. I'm going to click on EC2, and from here I can see my dashboard for EC2 which will list all of my running instances as well as many of the other types of information that are contained within this dashboard. , the next thing I'm going to do is I'm going to scroll down, I'm going to go to the option to launch an instance, and from here we're going to click on Launch Instance. We get a chance here to choose an AMI.

There are several that are provided by AWS. There also are some that are provided by the community, and then there are some that are provided in the AWS marketplace, which those could charge an extra fee for using those AMIs. We also can see here that there's an option for my AMIs because, we can actually create our own AMIs based off of our own customizations. In this case, I'm going to choose the Amazon Linux 2 AMI. So I'm going to select this first one. And we get an option to go in and select our instance type. We've talked a lot about instance types, and if we were to scroll down, this list is pretty long, and we can go all the way from general purpose into commute storage, memory optimized, and accelerated computing.

For now, though, since we're just interested in testing this out, we're going to use a t2.micro instance type, and part of the reason that we're going to use that is because that is free until you're eligible. So if this is a new AWS account for you, you should be able to run this within the free tier and not incur charges in doing so. However, I do want to quickly remind you that if you leave these instances up and running there will be charges associated with them. Make sure that you've set up the billing alarm that we guided you through in the previous book. So next, I'm going to hit the Next option to Configure Instance Details.

From here, I'm telling it to only launch one instance. That's all that we need. , the next step is we have an option here for our purchasing option. So I do have the option here to make this a Spot instance or to at least request a Spot instance. However, I'm not going to do that here, and we're going to choose to simply launch an instance. We haven't yet talked about VPCs, and so I'm going to leave these options here as the default values for both the network in the subnet, but I do want to switch Public IP to be Enable. And we can look here that we have some other values for things that we haven't covered yet, but I'm going to navigate down to the bottom, and I'm going to go to the section for Advanced Details. Here within this section we have something called User data, and this is basically just commands that the server will run when it starts.

And so I'm going to paste in a value here that will basically have it install a web server and then start that web server. And if all of this works correctly, when we're done we should be able to go and see the test page for that web server up and running. So let's go ahead and hit Next and add storage. So here, we do have an EBS volume that we're going to be leveraging, and we could choose to add additional storage if we wanted to onto this server. we don't need to, and so because of that, I'm not going to change anything on this screen, but just k that you can. And we'll talk more about EBS later, but we even have the option here to go in and change the type of volume that we're using, in this case, we're just going to keep it as a General Purpose SSD, which is kn as a gp2.

Next, we could go through here and add tags, so I could go in and add a tag here. We could say here that this is just a test server, and so if we used that purpose tag across everything, we could note how much of our charges are associated with, for example, test versus production.

We could change a lot of different things on that, but in this case we'll keep that tag in place, and next, we'll go in and configure the security group. This is something else that we haven't covered in depth yet, but we will be covering more later on within this book.

But I need to do two things here. First of all, currently any IP address, so any user, could go in an attempt to log into the server to manage it, and we don't want that to be the case. And fortunately, AWS provides an easy way to limit access. At this point, I'm just going to say my IP, so only my IP address, can actually go in to manage the server. But I need to add one more rule. We want to set up a web server. The type of communication that a web server has is called HTTP, so I'm going to select HTTP, here on the left. If we leave this at the default value where it says 0.0.0.0/0, in this case, what that's saying is is that anyone can access the web information from this server, and that's what we want.

Now that we have that in place, we're going to hit Review and Launch, and then we will look at the summary page here. Everything looks good, and we'll just hit Launch. , it's going to ask us here to create a key pair, and if you haven't done that yet, you will need to do it. In this case, this is what will allow you to actually sign in to this server, utilizing this key pair. We're not going to be signing into this server to administer it. But you still will need to create a key pair, then we'll need to acknowledge that we have access to that key pair, and then we can finalize the launch process.

From here, it's telling us that our instance is launching, so congratulations, you have launched an instance in EC2, so you've launched a web server in the cloud. It’s going to take in a minute to get started, but what we can do is we can click on this little value here.

This is the identifier for our EC2 instance, and it will let us k right that it is currently pending, so it is going through the launching process. So we'll give this just a minute to complete. So we can see that our instance is currently set to be running, and so we should be able to access it.

So, what I'm going to do is I'm going to look at the section down here that gives the information for this instance, and there will be a section called Public DNS. I'm going to click on this to copy it to the clipboard, and then I'm going to go up here and I'm going to open up a new tab in the browser and I'm going to paste that value in. I'm going to hit Enter and you can see that we do indeed see our test page.

This means that you have properly configured a web server to be up and running in the cloud, so this is what we want to see. However, we just don't want to leave an instance running if we don't need it. So I'm going to go in here and I'm going to be sure that my instance is selected, and then from here we're going to go under Actions, and then we're going to go to Instance State and we're going to go to Terminate.

Here, this is going to let us know that the EBS volume will be deleted when we delete the instance, and this is actually what we want. So we can hit Yes, Terminate, and by doing so, we have shut down our instance and it'll take it a little bit to shut down and be fully terminated. But let's walk through what we've done here. We've been able to go in through the console, launch a new EC2 instance from an AWS provided AMI. We have then been able to go in and configure it as an instance to be a web server, and then we've launched it, we've seen that it worked, and we have terminated our instance. So congratulations on firing up and spinning down your first server on EC2.

Chapter 6 AWS Elastic Beanstalk

Up to this point, we've been focusing on Amazon EC2, but we're going to look at a different compute option that is available on the platform, and that is AWS Elastic Beanstalk. So at a high level, Elastic Beanstalk automates the process of deploying and scaling your workloads on EC2, but the difference is instead of dealing with those servers directly, which we would call Infrastructure as a Service, this is more of a Platform as a Service type approach. Because of that, it does support a specific set of technologies. So unlike EC2 where you can really do anything you want, as long as you can get it up and running on a server, you can do that on EC2. Here, Elastic Beanstalk works within a set of technologies. It does leverage existing AWS services, and you only pay for the other services that you leverage, and so this brings up kind of a new category of services for us on AWS, and these are services that really just make it easier to use other AWS services. So in this case, we're still theoretically running all of our compute on EC2, but the process of managing those servers and handling things like provisioning and load balancing, scaling, and monitoring are all handled automatically through the work of Elastic Beanstalk by connecting other services into the overall platform. You might be asking, well, what platforms does it support, and that's a great question. So here, we're going to be looking at a lot of the usual suspects here, like Java and NET. We also have PHP and Node.js, as well as Python, Ruby, Go, and then interestingly enough, it also supports Docker. So the good thing here is even if you weren't using one of these other platforms, but you still wanted to use Elastic Beanstalk, if you could configure a Docker container with whatever framework you wanted to leverage, you could still find a way to support it on Elastic Beanstalk, but with just a little bit of extra work. You might be saying, well, I get that, and I get that all of these different technologies are supported, but why would I choose Elastic Beanstalk? What are the features that it has that would be an advantage over using EC2 directly? Well, first of all, one of the great things about Elastic Beanstalk is it does have integrated monitoring included, and again, it's using other services that we'll talk about later to pull that monitoring information in. here, deployment is critical. You might just think well I'm just copying some files to a server, how hard can it be? But in reality, in production, deployments aren't that easy, especially when we're talking about load balanced environments and you're dealing with multiple servers and trying to figure out how to appropriately deploy things onto servers and make that available to end users, and Elastic Beanstalk manages all of that for us. In addition, it also handles scaling. So if we want to be sure that our web application that's used within our business as an internal application, if we k that there's going to be a huge demand upcoming, we can trust that Elastic Beanstalk is going to be able to handle that scaling appropriately, and we can add some configuration to that so that we can best handle upcoming demand. But it also allows for EC2 customization. So here is the one area where it kind of starts to maybe stretch beyond just a little bit, the Platform as a Service approach, it does give you the ability to add some customizations to those servers that it's actually going to be running on. So why would you want to use Elastic Beanstalk? So, if you want to deploy an application with minimal knowledge of other services, so let's say you know Elastic Beanstalk, but you're not familiar with really administering EC2 servers or auto scaling groups or getting metrics from CloudWatch, setting scaling rules, these are things that you can simply rely on Elastic Beanstalk to do for you. In addition, if you want to reduce the overall maintenance needed for the application, this can be a great choice. So if you can fit into the specific use case that Elastic Beanstalk has, you can avoid having to deal with many of the other administrative tasks that are simply taken care of by the platform. In addition, this makes sense if you're not looking to completely customize the environment that you're in. And if you say, well, I want to use this specific AMI and it has to have this specific package and I don't want it to upgrade to this other version of the package, you can get really specific about those things. If that's you, you probably want to look at using EC2 directly. But if you say, you know what, I really don't need to customize it, I can take what it gives me by default, then Elastic Beanstalk can be a great choice and one that will save you time, especially over the life of the application. Now that we've had a chance to understand the purpose and features within Elastic Beanstalk, we're going to launch a sample application on the platform. We are first going to be accessing the sample Elastic Beanstalk applications from the Elastic Beanstalk documentation. We will then be launching one of those sample applications on Elastic Beanstalk. And then finally, we will be deleting a deployed

Elastic Beanstalk application. So I'm here on the Tutorials and samples page within the Elastic Beanstalk documentation.

And I'm going to scroll down to the bottom half of the page where we actually have sample applications that have been included. I'm going to click on this Node.js application to download it. In this case, this is going to be the sample application that I choose. If you wanted to try out one of the others, you certainly could as well.

But when we go to launch it within Elastic Beanstalk, you'll need to be sure that you select the correct platform. So I'm going to navigate over to the AWS console, and I'm already logged into the console. And from here, I'm going to either click on Elastic Beanstalk within Recently visited services, or if you don't have it within recently visited services, you can simply type in Elastic and then search for Elastic Beanstalk.

From here, it's going to tell us Welcome to Elastic Beanstalk, and we're going to click on the Get started button. Next, we're going to create a web application, and I'm going to call this one And then from here, we're going to go down and choose the platform. Because I chose Node.js, this is the platform that I'm going to select here.

We do have the option here to just do a sample application without having to leave the Elastic Beanstalk dashboard here in the console; however, I'm going to choose to go to the Upload your code option. Part of the reason I'm doing this is because I want you to have experience uploading your own code into Elastic Beanstalk. I'm going to hit Upload, and I'm going to select the local file. And here I'm going to go in and select the Node.js sample application, and then I'm going to hit Upload. From here, we'll get a chance to go in and configure more options. At a high level, we don't need to change any of the settings that are included here; however, I wanted to show this to you so that you can see some of the different capabilities that you can configure within Elastic Beanstalk.

At this point, we'll choose to leave all of these values in place, but you can see there's quite a bit of configuration that is available to you. We'll hit Create app. And we can see that Elastic Beanstalk is going through the process of guiding our initial deployment and we're able to use this console here to view the progress as it deploys it out for the first time.

And we can see that our application has launched successfully. It's currently letting us k that the health of this application is okay. And if we look, there is a URL that we can click on to get access to our web application.

And if we go look at this, we can see that we have indeed launched our first web application here on Elastic Beanstalk. And it actually allows you to kind of scroll through and look at the different documentation based on the links that are included.

If we go back and look at the dashboard, as a reminder there are several different things that are included within the platform, including keeping things like our logs, so we can see our logs, we actually can go in and request those at any point in time, we can look at the health, it'll go through and let us know the instances that are running and the percentage of requests, we can see here that we have had some good 200 requests, which is what we would want. We can go in and see monitoring so we can actually see how our web application is performing.

This has just started up, so we're not going to have any good data to look at the moment. We can even go in and configure alarms based on, for example, if our application becomes unhealthy if it's not responding. So these are all the things that are provided by Elastic Beanstalk just as a part of the platform as a whole, and it doesn't require you to know how to go in and configure all of the backing services that are feeding into Elastic Beanstalk. But we're going to take our last step, and that's we're going to delete our application. So I'm going to go here under this application, so I'm going to go here under Actions and I'm going to choose to Terminate environment.

And here to terminate it, we're going to need to enter in the name of our application, and then we're going to hit Terminate, and this will destroy the web application that we've just deployed, which is what we want. So, through this process, we have been able to take a sample application provided by Elastic Beanstalk, launch it within the platform, see it running, see some of the capabilities that are provided by Elastic Beanstalk, and then ultimately terminating our custom application.

Chapter 7 AWS Lambda, VPC and Direct Connect

We have already talked through two of the three compute services that we said we would be covering in this chapter, and that would be Amazon EC2 and AWS Elastic Beanstalk. But, now, we're going to be introducing the third, which is AWS Lambda. Lambda is different in that it lets you run code without provisioning or managing servers, and you only pay for the compute time that you consume, and so you can run code for pretty much any type of application or service, all with zero administration. So at a high level, AWS Lambda does enable you to run code without provisioning infrastructure, and, as we said, it is only charged for usage based on execution time. the real variable when it comes to pricing with Lambda is the amount of memory that you make available to it, and you can choose to have anywhere from 128 to 3008 MB allocated for your functions that are running on AWS Lambda. The great thing is, is that it integrates with many AWS services just out of the box. For example, the work with things like S3 and DynamoDB is just automatic once you configure it. And because of that, it can enable what we would call work flows. So, you could say something like when I upload a file, I want you to execute this function. This is the primary service in terms of compute for the serverless architecture approach, and we'll be talking more about serverless later within this path. , let's look at some advantages for Lambda. First of all, there are reduced maintenance requirements, so you don't have to worry about those underlying servers and keeping them up to date. In this case, AWS owns that for you. Also, it enables fault tolerance without you having to build it in. So this concept of running across multiple availability zones and making sure that no single point of failure can take down your application, well that's just built in to Lambda. Also, its scales based on demand, so irrespective of the number of users you have that are using your lambda function, you can k that it will scale. And your pricing is based on direct usage, whereas with EC2 if you have a server up and running and let's say it supports 1000 people and you only have 10 people using it, you're having to pay for that whole server, but with Lambda it truly maps to usage because it's only going to charge you based on the usage that you have. Now we're going to walk through three different scenarios that will help you gauge how well you have absorbed the information that we have presented within this chapter. So first of all, we're going to be looking at Kate, and her company is in the process of moving multiple workloads into AWS. And one of those workloads is an application that will be leveraged for at least five more years. This is just a core business application that her company will be leveraging for the foreseeable future. However, they're looking to be as cost efficient as possible for this EC2 usage. So for them, what EC2 purchase option should be chosen for the application? Next we have Danny, and he's looking to deploy his PHP web application to a virtual server, but he doesn't have experience managing EC2 instances on AWS. He does need the ability to scale, though, because he does think this application is going to be pretty popular. So for Danny, what is the best compute option for him based on this criteria? And our third scenario; Sophia's company is transitioning to the cloud for its data processing workloads. These workloads happen daily and can start or stop without a problem, so they're configured to handle that. The workload will be leveraged for at least one year, so for Sophia, what EC2 purchase option would be the most cost efficient choice? Compute services are at the heart of any cloud platform, and we have introduced three different compute approaches on AWS in this chapter. So here's what we've gone through. First, we introduced Amazon EC2 and its capabilities. We also explored different pricing approaches for EC2 instances. Then, we introduced the capabilities of AWS Elastic Beanstalk, as well as reviewing use cases for Elastic Beanstalk. And then finally, we introduced AWS Lambda. Let’s go back and take a look at the three scenarios that we presented before. So first we had Kate, and she was looking to move a workload over that was going to last for at least five more years, and her organization was looking to be as cost efficient as possible. So what purchase option should she choose? Well in this case, we would look at an all upfront reserved for three years. In this case, we can maximize the cost savings by having a reserved instance and paying for it all up front. Next, let's look at Danny. For Danny, he was looking to deploy his web application, but he doesn't have experience working with EC2 directly, so what should he do? So in this case, Elastic Beanstalk would be the best choice for Danny. Elastic Beanstalk supports PHP, and it also handles things like scaling out of the box, and it doesn't require manual configuration for that. So Danny could get pretty far by leveraging that platform. Let’s look at the third scenario. Here, we're talking about Sophia's company, and they're transitioning to the cloud for its data processing workloads. And the question is, what EC2 purchase option would be the most cost efficient choice? While you're looking at this here, you might be looking at the line that says this workload will be leveraged for at least one year and think, ah, exactly, this is about reserved instances, but it's not. The solution here is spot instances. Why? Well, because this workload can start and stop without a problem. So any time you see anything about starting and stopping a workload without a problem, and it's going to talk about the most cost efficient choice, the solution is always going to be spot instances, because spot instances are by far the most cost efficient way to leverage EC2; however, it can't work with an application that can't start and stop on demand. So in this case, spot instances would be the best choice for Sophia's company. Now, we're going to be talking through the content and network delivery services on AWS. And when we're talking about these services, we're really primarily talking about six different services, and that includes Amazon Route 53, Amazon VPC, or Virtual Private Cloud, AWS Direct Connect, Amazon API Gateway, Amazon CloudFront, and Elastic Load Balancing. First of all, we're going to be introducing the concept of virtual private clouds on AWS, and then we'll be understanding the purpose of AWS Direct Connect, when you would use it and why it's important. Then, we're going to be examining DNS utilizing Amazon Route 53. We'll then be reviewing Amazon CloudFront, which we talked about briefly when we were talking about the AWS global infrastructure. And then we'll be reviewing API Gateway. And finally, we'll be introducing Elastic Load Balancing and some different scaling approaches on AWS. First, we're going to be talking about Amazon VPC and Direct Connect. When we're talking about a virtual private cloud, or VPC, really what we're talking about is a logically isolated section of the AWS cloud, and this is where you can launch your EC2 servers in that area and k that it is just your slice of the cloud. And this is a virtual network that you define and you can configure. When we're talking about VPC, there are a couple of things to note here. First of all, it does enable you to have a virtual network in AWS. You don't have to be a networking guru to be able to pass your Certified Cloud Practitioner exam, but you do need to understand what a virtual private cloud is. But for those of you that want to dive a bit deeper, let me tell you that within a virtual private cloud, you can support both IPv6 and IPv4 addresses, and those are just different standards that we have for the addresses that our computers have on the networks that they're on. And you can configure an IP address range, subnets, route tables, and network gateways. Again, you don't have to know all of those details for the exam, but if this is an area of interest, you should know that you can configure all of these things within your VPCs. In addition, it does support both private and public subnets, so if you want to have areas that can't communicate or can't be reached from the internet, you can do that. If you want to have areas that are accessible to the internet, for example if you have a web server that you want to be available, you can do that here as well. If you do have private subnets, you can use network address translation, or NAT, for those private subnets, and there are multiple ways that you can actually establish a connection to your datacenter, and we'll talk about one of those shortly. And, you can connect VPCs to each other. So you can have what we call peering connections or use transit gateway to be able to connect multiple VPCs. You also can have private connections to many AWS services. Let me explain why that's important. If you want to be able to make sure that your sensitive application doesn't have to send traffic through the internet, that can just stay within your VPC, there's a way to do that, even when you're using specific AWS services. Next, we have AWS Direct Connect. This is a service that makes it easy for you to establish a dedicated network connection from your datacenter to AWS. So let's say, for example, you have a business application that uses application data that's stored within your datacenter, but the application itself is running on AWS. Well, it would be ideal to have a connection between your datacenter and AWS directly, as opposed to having to send it through the internet. And that's what AWS Direct Connect provides for you.

Chapter 8 Amazon Route 53 & Elastic Load Balancing

Now we're going to talk about Amazon Route 53, and this is Amazon's DNS service. And we'll talk more about what DNS means shortly. But if you remember, we have already mentioned this service. This is one of the two services that leverage AWS's edge locations within their global infrastructure. But at a high level, Amazon Route 53 is a domain name service, as we mentioned, and it's also a global service, which we also had mentioned previously. That means that it is not a regional service, that means any changes that you make are applied globally. Next, it's highly available, which means this service is going to have minimal, if any, downtime. In addition to it being highly available, it enables you to create highly available services, and we'll actually give an example of that shortly. Next, it enables global resource routing. So you could actually send people to a specific server based on what country they're coming in from. Or, in addition, you also could say I want to send them to the server that responds the fastest. What is DNS? DNS is really just a process by which we can map domain names, like pluralsight.com or amazon.com, to the specific addresses that are needed for identifying the computer services and the devices that are going to serve that content. So this is the connective tissue between those domain names and those IP addresses. When we look here at the console utilizing Route 53, as we've mentioned before, it is a global service and it does not require region selection. That means that any changes that you make here within the Route 53 console are applied globally. But you do need to know that DNS changes are not instantaneous.

It does need to what we call propagate those changes throughout the network of DNS servers around the world. So it might take a couple of hours for some of your changes to be realized for everyone. So let's give an example of how Route 53 can enable you to have a highly available application. So in this case, we have an ecommerce site, and it is currently hosted in US East 1, which is in Northern Virginia. And we have users from all over the globe that actually leverage this site. In this case, we have a customer that's coming in from South America, and when they access the site, normally it's going to route them to US East 1 and everything works great; however, what happens if our server goes down in US East 1?

Let's say we deploy a bad configuration, and that actually goes down. Well, what do we do? Well, we can configure Route 53 to have a failover so that if it can't reach its primary server, it can then route users to a new server. In this case, we can route them to EU West 1, which is in London. And in this manner, the user doesn't have to k that anything has changed, but we're routing them to a server that can fulfill their requests because we have that failover in place. And that's just one of several ways that we can utilize Route 53 to create highly available applications. Next, we're going to talk about a service called Elastic Load Balancing.

And this might be a term that you're familiar with because we did introduce earlier the concept of elasticity. And just as a reminder, it is the ability for infrastructure that's supporting an application to grow and contract based on how much it is used at a point in time. Well, one aspect of that is how we actually route users to the correct infrastructure. And that's where Elastic Load Balancing comes in. So what Elastic Load Balancing does, in essence, is that it can distribute traffic across multiple targets. So at a really high level, if we have users that are coming to our web application and we have two servers, it can choose to route the users between those two different servers based on the load of each of those servers. So by default, it will integrate with EC2, ECS, which we haven't yet talked about, but this is AWS's container service, so for running Docker containers, and Lambda. So it supports one or more availability zones within a region. So you could say, for example, we want to have our customer website running across three different availability zones, and we want to have servers that exist in each availability zone. And then we can leverage Elastic Load Balancing to distribute users to the right servers in one of those three availability zones. There are three types of load balancers. We have Application Load Balancers, or ALBs, Network Load Balancers, which you'll hear called NLBs, and we have Classic Load Balancers, and in some cases, those will just be referred to as either classic or ELBs. Next, let's talk about the process of scaling, specifically with Amazon EC2. And we will be diving down into this quite a bit deeper in a later book within this path, but at a high level, we have a couple of choices when we need to scale our servers on EC2. So the first here is what we call vertical scaling. This is when you scale up. So note here that vertical scaling and scale up are talking about the same approach. We scale up our instance type to a larger instance type with additional resources. So let's say we were using a t3 medium instance type, and let's say that we just noticed that our users are hitting that server pretty hard, and it's not responding as quick as it needs to, and it's even dropping the connection for some of our users. Well that's not a good thing, we need to address that. And sure, we could go in and say that we want to get a larger server, we could do something like an m4 xlarge, but if we do that, we're going to have to shut our server down, and then we can add those additional resources to it by changing the instance type, spin it back up, and we're able to meet our customer's need. However, that's usually not the best way. The best approach is what we would call horizontal scaling, or scale out. And again, remember here that horizontal scaling and scale out are talking about the same approach. This is where we leverage Elastic Load Balancing and we add additional instances to handle the demand of the application. So in that case, maybe we have a t3 medium server, and maybe then we just add two more t3 medium servers, and then we rely on horizontal scaling with the Elastic Load Balancer to actually handle the process of routing our users to the correct server. Later, we're going to tie this together with EC2 and talk about how you can leverage auto scaling groups alongside Elastic Load Balancing to make this work.

Now, we're going to talk about two additional AWS services, and those are going to be Amazon CloudFront and API Gateway. We’ve mentioned CloudFront before. This is a service that leverages the edge locations within the AWS global infrastructure. And it is a content delivery network, which means that there are servers around the world that you can send your content to. And why would you do that? Well, because it enables your users to get content from the server that's closest to them, which will increase performance. It also supports both static and dynamic content. Many people think about content delivery networks only serving static content, like images and videos or even specific text, but in this case, you can also configure it to support dynamic content. And it also utilizes the AWS edge locations that we've mentioned, which we also noted were the most prevalent form of the AWS global infrastructure. It also includes several advanced security features, and those include things like AWS Shield, which handles distributed denial of service attacks and their web application firewall. Within a later book in this path, we'll be diving more into security and understanding how that factors in to your use of CloudFront. But just as a reminder, there are edge locations that exist globally, so this gives you the power to take your content and distribute it globally within just a matter of minutes. So next, let's talk about another service, Amazon API Gateway. API Gateway is a fully managed API management service. This means that you can create APIs, which are just web services that then other applications can call, and you can make those available. And you can actually distribute those through CloudFront. It directly integrates with multiple AWS services, including several of the services that we have learned about so far within this book. It also gives you concepts like monitoring and metrics on your API calls so that you can understand how your APIs are being used and also debug them if they're not working properly. You can also integrate this in with both VPC and private applications, so it doesn't just have to be for public API calls. So next, we're going to discuss a service called the AWS Global Accelerator. And according to AWS, this is a networking service that sends your user's traffic through Amazon Web Services' global network infrastructure, improving your internet user performance by up to 60%. Let’s talk for a minute about how that's possible. So, first of all, Global Accelerator and one of the ways it's different from a solution like CloudFront is that it does utilize IP addresses, and in this case it still uses edge locations like CloudFront, but it's actually using it from IP resolution instead of DNS. So, once the user reaches the edge locations, instead of routing traffic with the user request through the public internet, once they reach those edge locations, no matter where you're resolving it to, it's going to route that traffic through the AWS network and not the public internet. And it can route requests to many different AWS resources, and that includes things like a network load balancer, or NLB, an application load balancer, or ALB, or even just EC2 instances, or maybe you've just assigned an elastic IP address to some type of AWS infrastructure, you could use that as well. Let’s talk about the different performance improvements that are possible with Global Accelerator. First of all, the distance between that user request and the initial endpoint is going to be minimized because it's using edge locations. This is also true when you're using something like CloudFront. Traffic is going to be optimized by using the AWS network instead of the public internet with that resolution. And what this does is it does result in improvement in that first byte latency, the jitter, and the throughput. So overall, the request is going to be more efficient, and it does also provides superior fault tolerance by not relying on DNS resolution. And let me speak to this for just a minute. So with most solutions, what can happen is that we use a hostname, and that hostname, the IP address, might get cached. So if you're using something like Route 53 for failover, what happens is that if the client remembers an old IP address and it needs to fail over to a new region, for example, in some cases that switchover might not be seamless. But here, because of Global Accelerator, we are using resolution so it is able to make that transition seamlessly. So let's talk about when you would consider AWS Global Accelerator, especially over a solution like CloudFront, because they are similar. First of all, if you're using a protocol, things like UDP, MQTT or VOIP, so you might be using UDP in a gaming context or in a video/audio type context, with MQTT, you might be leveraging this for devices. And then VOIP if you're doing, again, internet telephony. If we look at another use case, and that's going to be if you have a situation that requires a static IP and not a resolution, then in this case you're going to need to use Global Accelerator. Also, if you need instant failover with the highest level of high availability, then in this case you would certainly want to take a look at the AWS Global Accelerator. Now, we're going to walk you through some scenarios so that you can see how you have mastered the content that has been presented within this chapter. So first of all, we're going to look at Jack, and her company maintains two different corporate datacenters, and they want their datacenters to work directly alongside AWS for specific workloads. So in this case, they probably have some data that's stored within their own datacenter, but they want to have some public applications on AWS, and they need to have those public applications to be able to communicate with their datacenter. And so she's wondering if there's any way to have a persistent connection to AWS because she's worried about just sending everything out on the public internet that that could potentially be slow, and ideally, she'd like some of this communication to happen kind of behind the firewall. So, what service from AWS would you recommend that her company implement to meet these criteria? Next, we're going to talk about Tim, and his company is looking to serve content through their site to users all around the globe. And they're looking to optimize their performance to users around the world. They want to leverage a content delivery network. Previously, we asked what element of the AWS global infrastructure Tim would be leveraging, but in this case, we're going to get even more specific and we're going to ask the question, which service would enable optimized performance globally for this company's content? We've got one more scenario. Here, we're going to be looking at Elise, and her company has an internal application that runs on an EC2 server. Currently, there is some downtime because demand is greater than capacity for the server. So, for example, the server could probably handle about 150 users without fail, but in this case we have up to 200 simultaneous users. So this is a problem. So here, Elise is trying to decide if she should use bigger servers, so in essence changing the instance type, or more servers, so more servers of the same kind that they already have. So, which scaling approach, based on the ones that we've discussed, would you recommend for Elise's company? And then what services should they use alongside that approach? So we'll cover these answers next. Let's quickly give an overview of what we've learned, and then we'll dive into the scenarios that we discussed in the previously. So first of all, we introduced virtual private clouds on AWS. We also understood the purpose of AWS Direct Connect, which gives you that direct connection between your datacenters, your office locations, and AWS. We examined DNS with Amazon Route 53. We understood what DNS does and how Amazon Route 53 can help us build highly available applications. And then we reviewed Amazon CloudFront, as well as API Gateway. And then finally, we introduced Elastic Load Balancing and the different scaling approaches that you can take when you're leveraging it. Let’s look back at our scenario with Jack. So Jack's company has two different corporate datacenters, and they're looking to have their datacenters work closely with AWS. So, what service from AWS would you recommend for her company? Well in this case, AWS Direct Connect is a great fit. They can build direct connections between their datacenters and AWS, which means that that traffic does not have to go over the public internet. That also means that a lot of this communication it can happen behind the firewall for them. Next, we're going to talk about Tim, and his company serves content through their site to users all around the globe, and they really are looking to optimize performance. And they want to leverage a CDN, or content delivery network, so what service would optimize that performance globally? Well, in this case, this would be Amazon CloudFront. That is the content delivery network that is present on AWS. So any time you hear a question on the exam talking about a content delivery network, think about Amazon CloudFront and how that could help the users that are leveraging that service. So next we have Elise, and her company is running that internal application on an EC2 server and they're having the downtime. So which scaling approach would you recommend? Well, in this case, we would recommend horizontal scaling, or scaling out, using Elastic Load Balancing. This is preferable over using just bigger servers because it can handle whatever future load you can throw at it. However, if you get bigger servers, it's possible you're just going to have to take those servers down again and make them even bigger. We went to limit downtime, and we're going to do that by taking a horizontal scaling approach. And by using Elastic Load Balancing, we can make sure that we are routing users to the best server for them to use.

Chapter 9 File Storage Services & Hosting Amazon S3

Now, we're going to be talking through file storage services on AWS, and that means that we're really going to be looking primarily at six different services. And these are going to cover both file storage, but as well as data transfer. And so first of all, we'll be looking at Amazon S3, and this is one of the core AWS services. And then, we'll also be looking at Amazon S3 Glacier. We'll then be looking at Amazon Elastic Block Store, or EBS, Amazon Elastic File System, or EFS, as well as AWS Sball and AWS Smobile. So that's what we're going to cover. But let's talk about how we're going to cover that within this chapter. So first of all, we will be reviewing the overall storage services that are available on AWS to help you understand the ecosystem. But then, we will be doing a deep dive on Amazon S3 and its capabilities. Because it is a core service, it will be important to understand many aspects of this service. Then, we'll actually be implementing a static web site using Amazon S3. Then we'll be exploring the different archive capabilities that are present with both Glacier and Glacier Deep Archive. Next, we'll be looking at EC2 storage, and we'll be looking at that storage with both the Elastic Block Store and the Elastic File System. And then finally, we will be examining data transfer into AWS. First, we're going to be talking about

Amazon S3, and Amazon S3 is a core AWS service. It is the first service that I ever leveraged on AWS, and chances are if you've leveraged the platform at all, you have used Amazon S3 at some point. , Amazon S3 at a high level lets you store files as objects inside of buckets. Buckets are the unit of organization within S3. You will create a bucket, it will have a set of settings, and then any file that you drop in can have those settings applied to it. It provides different storage classes for different use cases, and we'll dive through those storage classes in just a bit. It allows you to store data automatically across multiple availability zones, which gives you durability and resiliency for your data. Another great feature of S3 is that it enables you to have URL access for your files. So if you want to be able to send a link to someone else to access a file within S3, assuming the permissions are correct, you can actually do that. And it also offers configurable rules for data lifecycle so if you want something to expire after a period of time or go to a different storage class. It also can serve as a static web host, and we'll actually be implementing that later within this chapter. So next, let's talk about the storage classes for S3. As a note here, we will cover the archival storage classes later on. So first of all, we have S3 Standard, and by default, if you upload a file to S3, it will have the S3 Standard storage class. Then, we also have S3 and this is a special storage class that will allow you to move your data to a correct storage class based on usage. And we'll talk more about shortly. Then, we have S3 Access. So, if you have data that is not frequently accessed, you can get a lower cost by utilizing this particular storage class. it still has the standard resiliency because it leverages multiple availability zones, but we also have S3 One Access, and this is for data that is not frequently accessed, but it is only stored within one availability zone. So you get a much lower cost point than you do with the other options; however, you do have less resiliency. So let's talk about , is the only way that you can automatically move data between different storage classes based on access. And within this overall storage class, it has two different classes associated with it. The first is frequent and the second is infrequent, so it will move your data back and forth between those two different classes. It gives you pretty much the same performance as what you get with S3 Standard, but it can provide a cost savings if you have some data that needs to be moved between those different storage classes. Next, let's talk about the S3 lifecycle policies. This is an approach that allows you to transition, based on your own custom criteria, the objects in your bucket. Transitions in this case can enable objects to move to another storage class based on time. It’s important to note here, you can't move something back and forth based on usage, that's only available with but you can do it based on time. And you also can delete objects with expiration based on age. So if you want a certain file to only last for 30 days, for example, you could configure that also using lifecycle policies. Policies can also factor in the concept of versions for an object within the bucket. So S3 does support versioning of data, and you could say something like, for example, delete a version of a file that's not the current version after seven days. So those are different things that you can configure utilizing S3 lifecycle policies. One additional concept here for S3, and that is the concept of S3 Transfer Acceleration. So this is a feature that enables you to upload data into your bucket much faster, and it does it by utilizing the AWS edge locations as a part of Amazon CloudFront. So if you need to upload your data in a fast and efficient way into your S3 buckets, you can consider utilizing S3 Transfer Acceleration. Next, we're going to walk through the process of hosting a static website on Amazon S3. And so over the book of this demo, here's what we're going to be doing. First, we're going to be creating a new S3 bucket from within the AWS console, and then we'll be uploading objects to that S3 bucket. After that, we'll be accessing objects from the S3 bucket using a URL. And finally, we will be configuring a bucket for website hosting. So I've logged into the console and I'm going to navigate to S3. I'm going to choose to search for it, and then we'll select S3 to go to the S3 console. And from here, the first thing I'm going to do is I'm going to create a bucket, so we'll hit Create bucket.

The first thing we need to do is give the bucket a name. You'll need to choose your own unique name. S3 buckets do need to have unique names, even across accounts. So now that I've entered in a bucket name and I have the region selected, I can hit Next.

From here, we have additional options that we can configure for our bucket, including versioning, but I'm going to choose to not set any of these values. We can hit Next.

From here, we have to specifically enable public access. By default, AWS is trying to block all public access. This is because organizations that put sensitive data into S3, AWS wants to be sure they don't accidentally enable public access to one or more files. So by default, they're telling everyone that the buckets need to block public access.

But in this case, because we do want to allow public access into our bucket, we're going to choose to deselect this. It’s important to note, this doesn't change the permissions for the data you put into your bucket yet, it just gives you the option to make items public. So I'll go ahead and hit here that I acknowledge this, and then we'll hit Next.

Once we see the information here for our bucket and everything looks correct, we can hit Create. So now that we've created our bucket, we can scroll to it and we can click on our bucket. From within here, we can go to upload files, so I'll hit Upload. We need to select the files, and I'll go in and select the files that I want to upload. We'll go ahead and select both of them, and we need to go through the wizard for uploading files. We can go ahead and hit Next here. The first step is to set permissions, and at this point we're not going to change any of the permissions, so we can go ahead and hit Next. We have the opportunity to go in and set the properties for those uploaded files.

You'll see first, we can go in and set the storage class, and we can see here the different storage classes that we have discussed for S3. We'll choose to leave it at S3 Standard for. But, once we go down to Encryption, I'm going to select Amazon S3

This will help us make sure that the data is encrypted at rest that we're uploading into the S3 bucket. We can hit Next. Once we look at the data here, if everything looks correct, we can hit

Upload. We have two objects that have been uploaded into our bucket. We can see here when we click on it, it presents us with an object URL. Everything within S3 you can access via a URL. So if I open up a new tab and I try to go to that URL, we'll see here that we get an AccessDenied message. That's because we haven't actually changed the permissions on our object yet.

So I'm going to go back to the console, we'll choose to scroll down here and go to Permissions. From within permissions, we're going to scroll down to where it says Public access. We'll then click on Everyone and we will say Read object. So this will give everyone read access to this particular object from within our bucket. I'll hit Save.

Once this is in place, I should be able to go back to the tab we were on previously and reload. When I do, we should be able to see the website that was included within the files that were uploaded into the bucket. We’ll navigate back to the Management Console tab. From within here, we're going to go back to our bucket. From within the bucket, we're going to go under Properties. We're going to enable the option for static website hosting, so I'll click on Static website hosting.

From here, we're going to say that we do want to use this bucket to host a website. The first thing we need to do is we need to enter the name for the index document for our site, which in this case will be index.html. We also can choose to go in and enter a custom error document.

This document will be shown every time the bucket needs to throw an error, for example a 404 error, which means they're trying to access a file that isn't included within the bucket. In that case, it would serve this error document, but we're not going to enter an error document for. We also have the option to go in and use the redirect rules. So, for example, if we wanted to redirect someone from index.htm to index.html, we could define those rules here, but we also aren't going to add any of those. We can hit Save. Once we do, we can see here that Bucket hosting is enabled. So if we click back on Static website hosting, we can see here that it gives us a unique URL. So we're going to click on this link to launch our site, but if you remember, we haven't yet set any permissions for our index.html file. So when we click on this, we should see, indeed, we do have a Forbidden message.

So let's navigate back into the console, let's go back to our bucket, and let's go to our index.html file. From within here, we should be able to go down to Permissions, we can go in for everyone, and we can choose to give everyone read access to the object, and then hit Save.

Once we have that in place, we should be able to go back to the tab where we had loaded in the specific URL that was given for static website hosting, and reload this.

And when we do, we can see that we have indeed loaded our index.html file and we're pulling in both the index.html file that we uploaded within the bucket. You've been able to create a bucket, upload files into that bucket, configure access for the objects within the bucket, and then ultimately enable static website hosting for the contents of that bucket.

Chapter 10 Glacier Deep Archive & Elastic Block Store

Next, we're going to be talking through Glacier and Glacier Deep Archive. So at a high level, Amazon S3 Glacier is designed for archiving of data within S3 as a separate storage class. Let’s talk about what that means. Let's say, for example, that your company needs to hold onto payment information from your customers for one year or for three years. In those cases, you need to have somewhere to store it, and you need to be able to produce that data for legal or compliance reasons, but you're not generally going to go in and access that on a regular basis. That is a great use case for archiving data within S3 using Amazon S3 Glacier. The way that this works is that it offers configurable retrieval times. So with this data, you won't be able to retrieve it right away, but you can choose to either retrieve it quickly or retrieve it less quickly, and you'll pay different based on the choice that you make. You can send files directly, or you can actually utilize the lifecycle rules in S3 to transition data into S3 Glacier. It does provide two different storage classes, being S3 Glacier and S3 Glacier Deep Archive. Let's quickly compare these two different options. So first of all, S3 Glacier. It's designed for archival data, and it has a minimum storage duration change. So, we wouldn't just store data here for a week or for a month. In this case, it needs to be at least 90 days, and it can be retrieved in either minutes or hours. This is where you pay based on the decision that you make, you would have a higher cost to retrieve in minutes over hours. You do pay a retrieval fee per gigabyte retrieved. So in this case, in addition to the storage cost, you do pay to retrieve it. , it will be about five times less expensive than using the S3 Standard storage class, so there is a compelling reason to utilize this if you do have data that you k you won't need to access at all except for rare circumstances. Then we have S3 Glacier Deep Archive, and this is also designed for archival data. But in this case, it has a minimum storage duration change, so you're going to want data here for basically about half a year at minimum. And in this case, it can be retrieved in hours, so we don't have the option to get it back in minutes like we do with S3 Glacier. Here, you also pay a retrieval fee per gigabyte that you retrieve. But in this case, it is over 23 times less expensive than the S3 Standard storage class. So in this case, the AWS Management Console can be used to set up S3 Glacier; however, when we're looking at uploading and retrieving data, this is where you're going to be using the programmatic approaches, so either utilizing the CLI or using the SDKs. This is one of those situations that I alluded to earlier where there are certain things you can't do within the console that you can do within the other interaction methods.

So next, we're going to talk about Elastic Block Store, or EBS. But before we get too deep into EBS, let's take a look at some different approaches that we have for file storage within Amazon EC2. The first is Amazon EBS, which is persistent block storage for use within Amazon EC2, and that's what we're going to be covering. But the next option is Amazon EFS, or Elastic File System, which is a network file system that is designed for workloads, and we'll be covering next. So at a high level, EBS is block storage that's designed to be connected to a single EC2 instance, and it can scale to support petabytes of data and also support multiple volume types based on what you need. So let's quickly give a overview of EBS. So first of all, it does enable redundancy within an availability zone. This way, you can make sure that your data is durable. It also allows users to take snapshots of its data. So if you have data that you want to have on a drive attached to your EC2 instances, but you do want to periodically take backups of this data, EBS can be a compelling choice. It does offer encryption of its volumes, but it doesn't necessarily encrypt things by default. You do need to make sure that is a step that you take if you're leveraging EBS. It also provides multiple volume types, including General Purpose SSDs, Provisioned IOPS SSDs, Throughput Optimized hard disk drives, and Cold hard disk drives. And we're going to review each of those volume types at a deeper level. So first of all, your General Purpose SSD is a cost effective type that is designed for general workloads. But if you have a more intense use case, you might want to take a look at Provisioned IOPS SSDs. So this is a situation where you're looking at volume, but you really need a low latency. We also have Throughput Optimized hard disk drives, and this is designed for frequently accessed data. But, if we have situations where we have less frequently accessed data, we might want to consider the Cold hard disk drive volume type.

So next, we're going to talk about Elastic File System, or EFS. Previously, we talked about Elastic Block Store, or EBS, as one approach for attaching storage to an EC2 instance. And here, this is going to be the other option. So here, for EFS, first of all, it's important to note that it is a fully managed NFS file system, and it is designed specifically for Linux workloads. , similar to EBS, it supports up to a petabyte scale. So we're talking here about petabytes of data. It also can store data across multiple availability zones, so you get some of that durability by default. It provides two different storage classes, the first being Standard and the second being

And it can provide configurable lifecycle data rules so you can transition between those storage classes. Let’s quickly take a look at an example of how we could leverage Elastic File System, or EFS. So here you can see an example where we have two different EC2 instances, one in availability zone A and one in availability zone B. And in this case, Elastic File System is working across both of those availability zones and has a mount point within each availability zone. So unlike EBS where we're targeting attaching a volume to a single EC2 instance, Elastic File System has an ability to be a network file system that you can attach to multiple instances at the same time. If you're running Windows workloads on AWS, one option you need to be familiar with is Amazon FSx for Windows File Server. And it is a fully managed native Windows file system, as opposed to being a Linux file system like we saw with EFS. And it includes native Windows features, and this includes things like SMB support, Active Directory integration, and support for Windows NTFS. And it also utilizes SSD drives for low latency.

Chapter 11 Data Transfer with AWS Sball

Next, we're going to talk about how you actually get large amounts of data onto the AWS cloud, even if you don't want to put it over the public internet. And one of the ways that we can do that is by leveraging the data transfer services that are provided with AWS. The first is AWS Sball, and this is a service where you want to physically migrate petabytes of data onto the cloud. But if that's not enough, there's even another service called AWS Smobile, and this is a service if you physically want to migrate exabytes of data onto the cloud. So let's compare these two options. First of all, we have AWS Sball, and it is designed for data transfer, and as we mentioned, it supports petabyte scale in terms of data. In this case, it is going to be a physical device that is delivered by AWS to your office location, and you can actually connect this to your network and then upload your data from within your network. When you return it, it's going to be returned by a local carrier back to AWS, and then when they get the device, they're going to load your data into S3. And so that's how AWS Sball works. , AWS Smobile is a little bit more intense. First of all, it is also designed for data transfer, but in this case it supports exabyte scale. It is a ruggedized shipping container, and it is going to be delivered to your location. A shipping container like one that's pulled by a truck. This is rather large, and this is for some more extreme type scenarios. AWS will be there to set up a connection from your network to that shipping container, and then it actually loads your data onto the Smobile, and then AWS will work to be sure your data is then loaded into S3 when that shipping container is received back at AWS and they can do multiple trips to get, again, exabytes of data. Now, we're going to talk through some example scenarios to see how well you've absorbed the material that we have presented within this chapter. And then, we're going to be walking through the answers to these scenarios. So, here is our first scenario. We have Elaine, and Elaine has launched a site that offers daily tutorials for developers. And she uses S3 to store the assets that are needed per tutorial. These assets are very popular within the first week that the tutorial is launched, but then, you know what, after this initial week, these assets are rarely accessed. So, how could Elaine reduce her S3 costs while maintaining durability? Next we have Rob, and Rob works for a social networking company, and they are moving to AWS. As a part of this transition, they have about 2 PB of content that they need to migrate into the cloud. He's trying to determine if there is a faster approach than uploading over the internet, because that is just going to take too long. So in this case, would there be another approach you would recommend for Rob's company? Finally, we have Bianca, and Bianca works for a company that produces a messaging app. And she's looking for a shared file system that'll connect to eight different Linux EC2 instances. The file system would need to support 1 PB of data, so what approach would you recommend for Bianca? We have looked at several different services over the book of this chapter that have to do with both file storage and data transfer. So, let's quickly review what we've covered before we dive in and take a look at our scenarios. So first of all, we were able to review the different storage services that are available on AWS. As a part of that, we examined Amazon S3 and its capabilities. We also implemented a static website on Amazon S3. We then explored the archive capabilities within Glacier and Glacier Deep Archive. We reviewed EC2 storage, leveraging both EBS and EFS. And then, we also examined the data transfer services that are available on AWS. So, let's review our scenarios. So first of all, we had Elaine, and she was trying to determine how she could reduce her S3 costs while maintaining durability. Well, what's the answer for her? Well in this case, it would be to use S3 lifecycle rules with S3 Access. So let's break that down. First of all, she could use S3 lifecycle rules because in this case, the assets are only popular within a week of when they are released, so she could define a policy that would change storage classes for the data after seven days of when they are placed inside of the S3 bucket. And then we could choose to leverage the S3 Access storage class because that data is not frequently accessed, but we would want to stick to this storage class and not one zone Infrequent Access because we do want to maintain the durability of the data. Next, let's look at Rob. And so what approach would you recommend for him to get 2 PB of content into the cloud? Well in this case, it's going to be AWS Sball. If we look here, we're looking at 2 PB of data. If we were looking at exabytes of data, AWS Sball wouldn't work. We would need to look at AWS Smobile. But because we're still dealing in petabytes, this would work just fine. So next we have Bianca, and Bianca was looking for a shared file system that would work between eight different Linux EC2 instances and would need to support up to 1 PB of data. So what approach would you recommend? Well in this case, it would be Amazon Elastic File System, or EFS. But note that there are a few conditions that need to be true before we can choose EFS. First of all, it needs to be for Linux instances, and in this case, it is. Also, it needs to be in petabytes of data or less, and not in exabytes of data. So it seems like both of these criteria are met, and EFS would be a great candidate for Bianca.

Now, we're going to be talking through the database services and utilities on AWS. At a high level, the services we're going to look at are, first, Amazon RDS, and then Amazon Aurora, Amazon DynamoDB, Amazon Redshift, Amazon ElastiCache, and the AWS Database Migration Service. It's important to note here that we're talking about several different types of services. So if we go back to our cloud computing models that we talked about previously, we talked about Infrastructure as a Service, Platform as a Service, and Software as a Service. With Infrastructure as a Service, this is the approach that gives us maximum control. But another option gives us minimum maintenance. So if we look at this, if we want to take the Infrastructure as a Service approach to databases on AWS, we would just take an EC2 server and we would deploy our database onto it. So, if you are a DevOps engineer, for example, and you want to have complete control over the OS that your database is running on, then you could utilize EC2 directly. However, maybe you want to have control over your database, but you don't need control over the underlying infrastructure. In that case, you could look to leverage Relational Database Service, or RDS. But then we'll also be talking about some different Software as a Service approaches that are available on AWS, like, for example, DynamoDB, ElastiCache, and Redshift. So, we're going to be talking through these services. First of all, we'll be reviewing the cloud computing models for databases on AWS, and then we'll be introducing the Relational Database Service, or RDS. Then, we'll be examining the capabilities of Amazon Aurora, which is one of the options within RDS. And then we'll be introducing the DynamoDB service. We'll then be reviewing the ElastiCache service, and then finally, examining how we do data warehousing on AWS. Another core service on AWS is the Amazon Relational Database Service, or RDS. And this takes a Platform as a Service approach to running databases that we can leverage within our applications that are running on the platform. So at a high level within RDS, first of all, it's important to note that it is a fully managed service for relational databases, and it does handle core things like provisioning and patching, backup and recovery of your database. These are all things that if you had to do it yourself, it would take a fair amount of time to implement this. And then to handle it at scale, it would take a fair amount of automation to integrate it in, and you get this by default when leveraging the service. One of the great things about RDS is that it does support deployments across multiple availability zones, and it also supports read replicas for some of the platforms. That means you can actually scale out your databases. This can help you scale more efficiently within the applications that are leveraging this database. It’s important to note that when you launch an RDS instance, it does launch into a VPC, or virtual private cloud. We talked previously about EBS and different volume types, and here within RDS, we have two different volume types that are available to us. First of all, we have General Purpose SSDs, and then we have Provisioned IOPS SSD drives. Let’s talk about the different platforms that are supported within Amazon RDS. First of all, we have MySQL. In addition, we have Postgres, MariaDB, Oracle, SQL Server, and Amazon Aurora. Amazon Aurora is an option that was actually created in the beginning to work on RDS. It is a MySQL and Postgres compatible relational database that was built from the beginning for the cloud. And it combines performance and availability of traditional enterprise databases with the simplicity and cost effectiveness of open source databases. You might be wondering if you have a database that's running within your own datacenter and you want to figure out how to get your data into a service like RDS, how would you do that? Well, AWS helps you solve that problem too. So we have the Amazon Database Migration Service, or DMS. And what this enables you to do is to move your data into AWS from your existing databases. And it supports both a and a continual migration of your data. And the great thing is, is it does support out of the box, many both commercial and open source databases, and in this case you only pay for the compute that you leverage in the migration process.

Chapter 12 Amazon DynamoDB, Elasticache and Redshift

Now, we're going to be diving in and taking a look at Amazon DynamoDB, which is a very exciting service on AWS. So first of all, it's important to note here what it is. It is a fully managed, NoSQL database service. So fully managed in this case, this is a Software as a Service option. Not only do you not manage the underlying infrastructure, but you also don't even manage the database layer, you simply use the database. In this case, it is not a relational database, it is a NoSQL database approach, which gives you some flexibility in terms of your schema, but also provides some other limitations depending on how you build your applications. And it provides both key value, and it also is a document database. And one of the great things about it is that it does enable extremely low latency at virtually any scale. You might be wondering how they were able to achieve that. Well, if you think about it, they have some great challenges to work with, including supporting amazon.com. And so DynamoDB was built with that kind of scale in mind. And so they were able to build a database that could perform extremely efficiently at a very, very high scale. And it does support automated scaling based on configuration, so you can choose to have its scale based on the predicted need that you think you're going to have, but it can also scale automatically based on your usage. And there's also several other additional features, like the Accelerator, or DAX, which gives you an cache that you can use alongside the database. We can handle more than 10 trillion requests per day and can support peaks of more than 20 million requests per second. And so this gives you just the scale that you can't get with many other databases, and it's provided here in a way where you don't have to manage any of the underlying infrastructure. So let's talk through some of the use cases on when you might leverage DynamoDB. Here we have scale without excessive maintenance. This is critical. If you have ever tried to scale a database in the cloud, it can be quite challenging. So if you have run up to an issue where you've had difficulty scaling, DynamoDB could be a great choice. Also, if you're embracing a serverless architecture, DynamoDB fits into that nicely because, again, you're not having to manage any of the underlying infrastructure or even the database layer. Also, if you have implementations where low latency is key, you want to be able to get responses quickly from the database, DynamoDB is a great use case. But also, if you have data models without blob storage, so big bits of binary data within your database schema, if you have that, it's not going to work well with Dynamo, but if you don't, this could be a great fit. So next, we have two other managed services that you can leverage on AWS, and those are Amazon ElastiCache and Amazon Redshift. So ElastiCache is a fully managed, datastore, and it actually has two different engines that are supported, both Memcached and Redis, which are really the two most popular options you could choose within datastores. It provides extremely low latency in terms of response times, and you can include things like scaling and read replicas to meet your application demand. And again, you can do this without having to be able to manage that underlying infrastructure. It handles most common use cases, including things like database layer caching. So if you're working with another database, even something like DynamoDB, and you want to add a cache layer in between your application and the database, ElastiCache could help fill that need, and even session storage. So if you're working with a web application that is doing sessions storage, you can utilize ElastiCache with something like Redis to be able to get quick responses on session data. Next, you need to be familiar with Amazon Redshift. It is Amazon's scalable data warehouse service. So if you're looking to do data warehousing for your user behavior data or for your analytics data, Amazon Redshift becomes a great choice. It supports petabyte scale, and in addition it also leverages disks and column storage. So this gives you the ability to beat out many traditional data warehousing solutions because, again, this was designed with the cloud in mind. It does give you the ability to fully encrypt the contents of your data warehouse, and it does give you a level of isolation within your own virtual private cloud. There are some great features that have been added to Amazon Redshift, including Redshift Spectrum, and this gives you the ability to query exabytes of data directly within Amazon S3. Now, we're going to run through a few scenarios to see how well you've done at absorbing the material we have presented within this chapter. And so the first scenario that we're going to look at has to do with Judit, and she is an IT executive in a financial services company. They're transitioning their data warehouse to AWS to take advantage of some of the great analysis features that are there, including some machine learning features they want to take advantage of in the future. Their data warehouse would need to support up to 2 PB of data, so for Judit and her company, which approach would you recommend? And so next, we're going to be talking about Sam, and Sam is a DevOps engineer at a tech company, and he needs to launch a MySQL database for a new web application they're going to start building next month. In this case, because of some of their security needs, they need to have direct access to the virtual server that MySQL is running on. So, what approach would you recommend for Sam's company? Next we have Dan, and he is the CTO at a gaming company, and they are trying to determine how to store user analytics. They need really low latency and the ability to scale to handle up to a million players for this new game that they're getting ready to launch that they know is going to be a hit. He also wants to minimize the amount of time it takes to maintain the database, because in the past, they've worked with some very complicated databases to be able to get this level of performance. But in this case, he wants to look at AWS first for a solution. So which AWS approach would you recommend for Dan? Next, we're going to be walking through the answers to this scenario and take a look back at all the things we've covered within this chapter. We have covered a lot in relation to databases and database utilities on AWS. So let's quickly take a look back and review what we've covered. First, we reviewed the cloud computing models for databases on AWS. We talked about how the concept of Infrastructure as a Service, Platform as a Service, and Software as a Service play out through the different database services that are available. We also introduced the Relational Database Service, or RDS. And we talked about Amazon Aurora, which is one of the specialty made engines for RDS that was built by Amazon for the cloud. We then introduced the DynamoDB service. We talked about how it is a completely managed NoSQL database, and we talked about some of the unique capabilities that it provides. Then, we reviewed the ElastiCache service, as well as looking at data warehousing on AWS with Redshift. Let's go back and take a look at the three scenarios that we introduced previously. First of all, we had Judit, and she is an IT executive at a financial services company and she has a need for a data warehouse in AWS. So, what approach would you recommend? Well in this case, the answer is going to be Amazon Redshift. This is the data warehousing solution from AWS. So if somebody asks you, what do you recommend for data warehousing on AWS, Redshift is going to be the best answer. And in addition, we need to look here and see that it is dealing with petabytes of data. If we were talking about something like exabytes of data, Redshift Spectrum might be an ideal option to consider. Next we have Fiona, and Fiona was looking to launch a MySQL database for a new web application. So what approach would you recommend for Fiona's company? Well, let me be honest with you. This is a bit of a tricky question. You might look at this and say, well, I know that MySQL is supported within RDS, so Fiona should choose RDS, but here's the problem. Fiona says that he needs to have direct access to the virtual server that MySQL is running on. So if she wants to manage that server directly, RDS can't be the solution for her. So in this case, we would be recommending EC2, and we would recommend this because he is wanting an Infrastructure as a Service option, which is going to be taking MySQL and installing it onto an EC2 server. If Fiona didn't need to manage the underlying infrastructure, we would certainly push her towards RDS because many aspects of the platform would be managed for him. Next, let's talk about Dan, and Dan needed a low latency database that could scale to handle up to a million users of the new game they're building. So, which AWS approach would you recommend for Dan? Well in this case, DynamoDB is going to be a great choice. It can absolutely handle this level of scale, and it can also handle the low latency requirement, so many gaming companies have adopted DynamoDB for requirements just like this.

Chapter 13 AWS Messaging Services

Now, we're going to look at another category of services called app integration services. When we're talking about app integration services, we're really talking about three distinct services on AWS. First of all, we're talking about Amazon SNS, or Simple Notification Service, which is a managed pub/sub messaging service. Then, we have Amazon SQS, which is a managed message queue service. Both of these services together would also be known as the messaging services on AWS. But in addition to those two, we are also going to look at AWS Step Functions, and that is a serverless workflow management service on AWS. So here's what we're going to be covering as we go throughout this chapter. First of all, we'll be introducing Amazon SNS, Simple Notification Service, and introducing Simple Queue Service, or SQS. And then, we're going to be exploring architectures that leverage SNS and SQS. Then after that, we're going to be examining AWS Step Functions, and we'll even have a chance to review sample AWS Step Function usage. So next, we're going to talk about the AWS messaging services, and the first one we're going to take a look at is Amazon SNS, or Simple Notification Service. This is a fully managed pub/sub messaging service. And what this enables you to do is it gives you the ability to create decoupled applications, and we'll talk more shortly about what that means specifically. But the way it works is that it organizes information according to topics, so you can choose to listen to one topic and not listen to another. And that's what we mean by pub/sub, it is publish and subscribe. So you can publish messages about, let's say, new orders, and then you could choose to subscribe to new orders and maybe order refunds. It integrates with multiple AWS services out of the box. And in addition to just providing messaging, meaning one part of your application can talk to another part of your application, you also can integrate this in and have end user notifications. So through this, you could send out an SMS notification or an email notification or a push notification to your mobile app. let's see some of this at work. So here we have a sample SNS architecture. Let's say that we're a Software as a Service company and we have a new user sign up, so we have a new user on our platform. We could choose to connect that process to an SNS topic. And let's say the first thing that we want to do is we want to integrate them into our CRM tool. Well in that case, we could choose to connect that to a Lambda function that would then talk to our CRM and put their data in there. Let's say that we also want to have a queue for some of our sales people to follow up with them directly. Well, we could use an SQS queue for that, and we'll talk more next about how you leverage SQS queues. But let's say we also want to send out an email so that the regional sales director in that area knows that you have a new customer that's come into the platform. We could also do that through SNS. But SNS works a little bit like some of the social networks that send out messages that then disappear. In this case, if we aren't listening for a message when it's published, we are not going to get it.

These messages aren't kept around, they're short lived. However, there's another approach we can look at, and that is leveraging Simple Queue Service, or SQS. So this is a fully managed message queue service, and it also enables you to build decoupled applications. But it also enables you to build fault tolerant applications, and we'll look next at how that works. It supports a decent amount of data, you can put about 256 Kb into a message, and it allows messages to be stored for up to 14 days. So while we're thinking about SNS being more like some of those new social media apps that enable you to just send messages and then they disappear, SQS is a little bit more like a mailbox. Those messages will stay there until you actually go and get them out of the mailbox. There are two different types of queues that are included with SQS. A standard queue does not guarantee the order of the items that you're going to be pulling off of the queue, but if you use a first in/first out queue, it will guarantee that you're actually processing those messages in order. Let’s begin to see what happens when you put together SNS and SQS into an architecture. First of all, we're going to have a user order, and this is going to come into our ecommerce company. And the first thing we're going to do is we're going to send that through an SNS topic. We're going to do something here called fanout. This is one of the benefits of using an SNS topic is that we can actually send messages out to multiple places. The first place we're going to send it is we're going to send it to a fulfillment queue. So this is going to connect to a server that we have in our warehouse, and this is going to be our order fulfillment service. And so our order fulfillment service knows to go and get a message off of the queue and then send that out so that we k what we need to put in the mail to send to our customers. But we also have another queue, and this is where the fanout happens. So coming from that SNS topic, we're sending out to multiple queues. The next queue is our analytics queue, and this is where we want to be able to store the information about the products that we're sending out to customers. And we want to eventually be able to pull this into our BI tool so we can see trends for our products. So to drop this into our data warehouse, we have a Lambda function on AWS that's going to drop that information into our data warehouse tool. But something happens. We send out an update to our analytics integration service and it goes away, it doesn't work. Here's the great thing. While that's not working, we're still loading up items into the queue. We haven't lost any information. And the moment that that service gets back up and running, it will be able to go and grab messages off the queue and put it into our data warehouse. This is what we mean by fault tolerant, is that we can have an aspect of our system go down, and it can still work according to its purpose. But let's say we also have something else happen in the same day, that server that's sitting in our warehouse just dies, the hardware itself goes kaput. Well, in this case, we're also still able to get orders in through our fulfillment queue. And the moment we get both our analytics ingestion service and our order fulfillment service back up and running, we will continue with our normal business processes without any lost data.

Now, we're going to talk about AWS Step Functions. And here are a few things that you need to know about AWS Step Functions. First of all, it enables orchestration of workflows through a fully managed service, and it does this in a way that supports serverless architectures. So this is one of the services that really enables this serverless mindset where you're trying to limit the amount of actual infrastructure that you have to own and maintain. It also can support very complex workflows, including error handling, and we'll see in a minute how you define those workflows within the service. You’re charged per state transition, along with any of the other AWS services that you leverage. In this case, that might sound strange to you, but here's what that means. If you have four steps in your workflow, you're going to be charged for each transition from one step to another. And then if that's running on a service like Lambda, you'll be charged the normal rates for the Lambda services that you're using. Workflows are defined using something called Amazon States Language. And you don't need to know how to write States Language at all, but I want to give you an example of how some of this would be defined. Let's say here that we wanted to get user signups and we wanted to do some things simultaneously.

After we get the signup, we want to insert the data into our CRM, and then we also want to send them a welcome email. And then when we get done with that welcome email, we want to schedule a call with one of our sales people. Once all of that's done, we want to wait a week, and then we want to send another follow up email to see how they're doing. This is an example of what you can accomplish utilizing a Step Function on AWS. And you see a very small portion of how that's defined on the left. This is using a format called JSON to define the Amazon States Language, and this is what allows us to define the different steps within the workflow. It does have a lot of integrations in to other AWS services. So we have, for example, compute services like integrating with Lambda, for example. It also can integrate with database services like DynamoDB, which we'll talk more about later within this book. It also can integrate in with the messaging services, meaning SQS and SNS that we have just talked about. Also, data processing services, and even machine learning services. So this is a powerful tool that you can use to build out very complex workflows and have Amazon manage the state for you of the different steps within that workflow. Now, we're going to walk through some scenarios to see how well you've absorbed what we have covered within this chapter. So our first scenario has to do with Ronald, and she started a nonprofit that assigns volunteers to open opportunities to serve. And recently, their database server went down, so when users went to sign up, it wouldn't let them do anything, so they actually lost some people that could have volunteered for some opportunities. While they have improved their database server, they're still not able to guarantee that there won't be downtime in the future. He really wants to explore an AWS service that could prevent these lost user signups. So in this case, what service would you recommend to Ronald? Next, we're going to talk about Jessica, and she created a list of onboarding steps for new customers for their app. , these steps have a lot of integrations with things like their CRM, they want to send out an email to users, they want to integrate this in with analytics. , Jessica's a bit worried that it's going to take her development team a long time if they have to build all of this from scratch.

So is there an AWS service that can help with this approach? Let's look at our third scenario, and this one has to do with Rob. And his company is an ecommerce company that's building a custom ecommerce platform, so they're not using one of the ones off the shelf. They're still adding some new functionality into their platform, and he wants aspects of the platform to listen for some key events, like orders and refunds. But, they don't yet k all the elements that need to respond to the events because they're still building it out. So, is there a service that would allow current and future parts of the platform to listen for events like these? Next, we're going to walk through the answers. We have covered three different app integration services. So before we go look at our scenarios, let's quickly review what we've covered. So first of all, we introduced Amazon Simple Notification Service, or SNS, and we talked about how it enabled you to either publish messages to a topic or subscribe to get messages from that topic. We then looked at Amazon Simple Queue Service, or SQS, and we talked about how in many ways it worked like a mailbox. You can put messages in and it will keep them there for up to 14 days. And this will help us build fault tolerant applications. We also explored architectures that leverage both SNS and SQS. And after that, we examined AWS Step Functions, and we reviewed how you could take a very complex workflow and define it within a Step Function. So, let's take a look at our scenarios. So our first scenario was Ronald, and his was losing volunteer signups because their database server went down. So what is Ronald really trying to solve for here? Well, he's trying to solve for being fault tolerant because right they're not fault tolerant, they're losing data with user signups. So what service would you recommend? Well, that's going to be Simple Queue Service, or SQS. This way they can take user signups even if their database server is down. And when their server comes back up, it can simply go pull messages off of the queue. Next, let's look at Jessica, and she had the complex list of onboarding steps. So is there a service that she could use to model these steps? Well, in this case, that would be AWS Step Functions because she could go in and take all the steps that she has in her onboarding process and integrate them into a workflow where every time there is a new customer, you could send them through that workflow with Step Functions. Last, let's look at Rob. And his ecommerce company was building that custom platform, but they don't k all the different components and what messages they're going to need to listen to, so they want to get a service that would allow for both current and future parts of the platform to listen for those key events. Well in this case, that would be Simple Notification Service, or SNS. In this way, certain parts of the platform could listen for order events and other parts could listen for refund events.

Now, we're going to be talking about management and governance services. And when we're talking about this category, we're primarily talking about six different services. First of all, we have AWS CloudTrail, and then AWS CloudFormation, Amazon CloudWatch, AWS Config, AWS Systems Manager, and AWS Control Tower. Here's what we're going to cover. First, we're just going to be reviewing the ecosystem of services that are provided for management. While we've looked at a lot of services that we can launch, this is where we transition to looking at what do we do once we launch resources within the cloud. Then, we're going to be examining how to create an audit trail by leveraging AWS CloudTrail. And then, we'll be exploring how you track infrastructure with both CloudWatch and Config. Then we'll be looking at a concept called infrastructure automation, or infrastructure as code, with CloudFormation. And then we'll be looking at operational insights utilizing Systems Manager. And then finally, we'll be reviewing AWS Organizations, which is a concept we have already explored, but we'll be looking at how that works with Control Tower. Now, we're going to talk about a service called AWS CloudTrail. And according to AWS, CloudTrail is a service where you can log, continuously monitor, and retain account activity related to actions across your AWS infrastructure. One of the great things about this is no matter which interaction method you're using, whether that's the console or whether you're using the CLI or whether you're working within the SDKs, CloudTrail is going to log all of those actions. So when we look at CloudTrail, first of all you need to k that it includes this information into an S3 bucket, or you can even utilize CloudWatch, which we'll talk about later. And it logs events in the regions in which they occur. One of the good things about CloudTrail is that it meets many different compliance requirements for infrastructure auditing. So in terms of even just general best practices, you want to know who is initiating actions that change your infrastructure. But for some of you, you might be in environments where that's not just a like to have, that is a requirement based on different compliance standards, and for many of those standards, CloudTrail will meet that. As a best practice, it should be enabled on every AWS account. if you have your own personal development account and you're the only one that has access to it and you don't want to deal with the cost of storing this information in S3, you could choose to forgo it. But for any other situation besides that, you want to be sure that this is turned on. And you can consolidate this. So if you are using AWS Organizations, you could utilize the organizational trail to get information across all of your accounts. , let's talk about some general use cases for using CloudTrail. First of all, and we've alluded to this already, it might be a compliance requirement. If you want to be able to track information specifically for compliance purposes, CloudTrail can meet that. It could be that you're using this for forensic analysis. So if you have already had a data breach and you want to go back and see what actions were taken against infrastructure, you could utilize it for that. Also, you could use it for operational analysis. So if you're looking to determine who potentially changed infrastructure that caused a crash or caused an outage, you could go in and analyze the users that perform that and then work with them to make sure that that action isn't taken again. You also can do it generally just for troubleshooting. So you could go back and look at when a specific bad configuration was injected into the system and use that to fix any issues that are happening within your infrastructure. So next, we're going to look at two services, Amazon CloudWatch and AWS Config, that help you manage your infrastructure. But let's quickly take a step back and look at all of the services that enable you to manage your infrastructure. And the first of these that we have is Amazon CloudWatch, and this provides metrics, logs, and alarms for your infrastructure. Then, we have AWS Config, and this continually evaluates your infrastructure against a predefined set of rules. We’ll be covering both of these. And the next option is AWS Systems Manager, and it provides operational data and automation across your infrastructure, and we'll be covering that next. These are not the only services that help you manage infrastructure on AWS, but these are the ones that you will need to k for your Certified Cloud Practitioner exam. Let’s first look at Amazon CloudWatch. So it is a monitoring and management service, and there are several different facets to CloudWatch. So it collects logs, so for example if you have log messages coming from your custom applications, this can actually collect those logs. It also gets metrics. So you could have metrics being the number of users that are visiting one of your load balancers, for example. Or you can even get events that are coming from most AWS services, and it truly is a citizen in AWS, which means that most services integrate with CloudWatch by default. But one of the powerful features is that it enables alarms based on metrics. So most all of the clients that I work with, we go in and set alarms so that if something is down or not performing as it should, CloudWatch lets you k as opposed to you having to go and find that out yourself or have your customers tell you. So next, it provides visualization capabilities for metrics. So if you want to have a chart that shows a metric over time, that's part of what is included with CloudWatch, and because of that, you can create custom dashboards based on the metrics that it collects. So here's a sample dashboard where you can see some of this in use. You can see here that it's tracking things like CPU utilization on some EC2 servers and network traffic and EBS activity for Elastic Block Store.

And so this is just an example of some of the types of data that you can put into a custom dashboard. But next, let's talk about another service, and that's AWS Config. So, AWS Config continually monitors and records your resource configuration, and it uses this so that you can evaluate that against the desired configuration. So let's look at how that works. So here within AWS Config, first of all is it does provide a level of configuration history for your infrastructure, and it's also going to take a look at a set of rules. These rules, there are many that are included within AWS Config. In addition, you can also customize the rules that are there, or you can even create completely new ones. Next, it includes conformance packs for compliance standards, including things like PCI DSS. that is a standard that if you're going to be accepting payment information, you need to be able to meet this criteria, and to do that, you can follow the rules that are already in place within AWS Config to be sure that the infrastructure that you have is compliant. , Config can also work within an AWS organization to look at all of your regions, as well as all of the separate accounts that you have. And one of the great things about it is that it also provides remediation steps. So if your infrastructure doesn't meet some of these standards, it will give you insight on how you can fix that. So next, we're going to talk about the third service that provides resources for managing infrastructure within AWS, and that is going to be AWS Systems Manager. , Systems Manager provides this unified user interface so you can look at operational data from multiple AWS services. And for many of those services, you can even automate some operational tasks across those resources. So let's look at AWS Systems Manager at a little bit of a deeper level. First of all, k that it provides multiple tools that makes your life easier when you're managing infrastructure on AWS, and we will not be reviewing all of them, but we will touch on a few of them. First of all, it can enable you to automate tasks for common maintenance actions. Let's say that you have two applications in your AWS account that you're supporting. Let's say you have 10 servers that support one application and 10 servers that support another, and you want to update those EC2 instances for your first application with a new version of a library. You can actually write that action one time and then simultaneously send it out to all the servers that need to receive it, and then you could choose to have it update on those servers, but not have it affect the other servers at all. That's just a example of how you can use this automation capability. It also gives you a secure way to access your servers using just your AWS credentials. So if you don't want to have to deal with separate keys or separate passwords, you can utilize this approach for accessing those servers. Another thing is that you can store commonly used parameters securely for operational use. So let's say that you have a specific database password and you don't want to store that with each of your applications, but you want your applications to securely access that password when they launch. You could utilize Systems Manager for that. Next, we're going to talk about AWS CloudFormation. Let's say for a minute that we have a custom application, and this custom application requires two S3 buckets, five EC2 servers, two SQS queues, and three Lambda functions. Well, we could go in and set up all of that in the console, but that creates a problem. There are a lot of manual steps that need to happen to make that work. And what happens if we miss one of them or we incorrectly apply settings to a resource? Well, CloudFormation exists to solve this problem, and let's talk about how it works. First of all, it is a managed service for provisioning infrastructure based on templates. So, instead of going in and manually clicking in the console, or instead of writing a custom CLI script, or instead of going in and using the SDK to write your own custom logic for creating your infrastructure, here you're going to use a service that manages that for you. There is no additional charge for CloudFormation. So with CloudFormation, you only pay for the resources that you launch, but the service itself does not have a charge. You can write these templates in two formats, one is YAML and another is JSON, and those are both common formats, and many different tools can support the writing of YAML or JSON templates. This enables an approach that we call infrastructure as code. This means that we're able to write a template that every resource on our team can use to launch infrastructure. And this means that we're taking the manual processes out of the picture. It also handles the management of dependencies between resources. So if we require one resource to be in place before we launch another one, it will go through and manage those dependencies. One of the powerful features that you get with CloudFormation, and this is going to be similar to what we've seen in services like AWS Config, but this is specific here to CloudFormation, is that it provides drift detection. So if we launch our application that has all of those resources I mentioned earlier, but somebody goes in and changes the S3 permissions to be globally available, then in that case, we can notice that change and we could choose to take action. Here's a very quick and limited example of what we can do.

This is a template, and with this information included in the template, if we ran it through CloudFormation, it would create an S3 bucket for us, and it would have the name , this is a very simple example, but you can expand to have very, very large templates. You can even have kind of templates within templates that enable you to launch infrastructure. So if you have a set of infrastructure, our custom application, you could launch that within one account and then turn around and launch it within another account. You could have one environment for production and another environment for testing and make sure that the two of those were completely identical because they're both based off the same template. And that's what you get with CloudFormation.

So next we're going to discuss another approach for managing your infrastructure on AWS, and that is going to be AWS OpsWorks. OpsWorks is a configuration management service, and it provides managed instances of both Chef and Puppet, so if you're already using either Chef or Puppet, OpsWorks might be something that you want to take a look at. If you're not familiar with Chef and Puppet, don't worry about diving into those to prepare for the exam. Next, your configuration is going to be defined as code for the servers that you're going to deploy it onto, and Chef and Puppet are going to manage the lifecycle of those configuration changes with your servers. One of the things to note about OpsWorks is that it can work in a hybrid cloud architecture, so you can utilize it if you want to manage your deployments both in the cloud, as well as within your own data center. OpsWorks itself is really made up of three separate subservices. First, we have AWS OpsWorks for Chef Automate. And so here this is your configuration management service that utilizes Chef Automate. Then we have AWS OpsWorks for Puppet Enterprise. So if you want to have a fully managed service that utilizes Puppet Enterprise, this would be for you. And next we have AWS OpsWorks Stacks, and this is what enables you to go in and define your application in layers, and then you can actually manage that using Chef recipes with Chef Solo. So, depending on what you would be looking to do, you would choose the specific subservice that matches the stack that you're working with. Next, we're going to revisit a topic that we have already discussed called AWS Organizations. But we're bringing it back because we want to talk about another management service, and that is Control Tower. So quickly, revisiting AWS Organizations. So it allows companies to manage multiple accounts under a single master account, and it provides organizations with the ability to leverage a feature called consolidated billing where it'll roll up all those child accounts under the parent account for the billing. And, it enables organizations to centralize logging and security standards across accounts. But really up to this point, we've talked about what you can do with AWS Organizations, but we haven't necessarily talked about what you should do with it. And so, AWS collected a lot of the best practices for this setup under a service called Control Tower. And so Control Tower is a service that creates this environment based on recommended best practices in areas like operational efficiency, security, and governance. So let's look at Control Tower at a little bit of a deeper level. So first of all, it does things like centralizing users across all AWS accounts, and this allows us to minimize effort of creating users across multiple accounts. And, it provides a way to create new AWS accounts based on templates. So maybe you want to be sure that every AWS account that is created within your finance department has specific settings included with it. It also includes this feature called guardrails, and this helps make sure that there are specific protections for accounts underneath your master account. So let's say, for example, you want to be sure that CloudTrail doesn't get turned off in any of the child accounts. Well, that is an example guardrail that can be included to make sure that they stay in compliance with what you want each of your child accounts to be doing. And it also includes a dashboard so that you can gain operational insights from a single view across all of your accounts. So next, we're going to talk through three different scenarios that will help you gauge how well you're doing at absorbing the information that we have presented within this chapter. So here in our first scenario, we're going to be looking at Elise, and he is an operations engineer at a financial services company. And he recently, by chance, discovered that someone had disabled a key security setting on a server. he's really concerned that events like this are going to go undetected, there's no way to k about them, and he's afraid they're going to have a data breach and then it's going to become a huge deal. So for Elise and his company, which service would allow the organization to continuously track configuration of infrastructure? Next, we're going to be talking about Phillip, and he is the lead architect at a SaaS company, a Software as a Service company, and they're going to be launching a new application that includes several components. It has some EC2 instances, some Lambda instances, it's also going to have some S3 buckets and some SNS topics and SQS queues. And he's looking to minimize the manual work that is required to create this infrastructure. So they're going to be launching dev environments and prod environments and testing environments and they don't want to have to do this manually every time. So, what service would enable Phillip to automate much of this effort? Okay, our last scenario here has to do with Caroline, and she is the CTO at a manufacturing company. And a cloud server that was needed to support their manufacturing process was accidentally deleted. They've managed to fix that issue, but they want to follow up with the individual that accidentally deleted that server so they can help them k how to not do that next time. So if they wanted to figure out who this individual is, what service would they leverage so they can go back and find out who performed this action? Next, we'll be walking through a summary of what we covered in this chapter and looking at the answers to these scenarios. So we have covered quite a bit within this chapter as we have looked at management and governance services on AWS. So let's quickly just take a look at what we've been able to cover. First of all, we reviewed the ecosystem of services that are provided for management of infrastructure on AWS. We then looked at how to create an audit trail utilizing a service called AWS CloudTrail. And then we explored how you track infrastructure utilizing services like CloudWatch and Config. And then, we introduced infrastructure automation with CloudFormation and this concept called infrastructure as code. We then were able to look at operational insights with a service called Systems Manager. And then finally, we reviewed the capabilities of

AWS Organizations when you are leveraging Control Tower. Let's take a look at our scenarios. So here in this first scenario, we had Elise, and he was concerned that some of these changes on servers might go undetected until there was a data breach. So, what service would allow him to continuously track the configuration of infrastructure? Well in this case, this would be AWS Config, and they could make sure that there is a set of rules enabled on Config that checks for the settings they care most about. Next, let's take a look at Phillip, and Phillip was the lead architect at a SaaS company, and he was looking to minimize the manual work required to spin up new infrastructure for their new application. So what service would enable Phillip to automate much of this effort? Well, this is going to be AWS CloudFormation because, here, Phillip and his developer team could go through and create templates, and these templates could be launched with CloudFormation, which would manage that infrastructure. Next, let's talk about Caroline, and Caroline is the CTO at a manufacturing company, and she was looking again to find out who deleted one of their production servers. They know it was an accident, but they want to be able to follow up with that resource. So what service could she use? Well in this case, this is going to be AWS CloudTrail because it is going to provide an audit trail of actions performed on AWS, irrespective of whether or not they were done with the console or with the CLI, or with one of the SDKs.

BOOK 7

AWS CLOUD SECURITY

BEST PRACTICES FOR SMALL

AND MEDIUM BUSINESSES

RICHIE MILLER

Introduction

This is the third book designed to get you ready for the Cloud Practitioner exam. Hopefully by this point you have already completed the Fundamental Concepts and AWS Core Services books in this learning path. Since we're covering security and architecture, we want to start off with just a simple overview of what we're going to be covering, as well as a few basic concepts. So first of all, here's what we'll be covering over the first few chapters. We will be reviewing these core concepts around security and architecture. We'll also be exploring the AWS Shared Responsibility Model, which is a critical piece to understand if you're going to be using the platform. We'll then be introducing the AWS Framework. We will then be examining fault tolerance and high availability on AWS. And finally, we'll be understanding the provided tools for compliance. So first of all, let's look at a concept that governs how we use AWS, and that is the Acceptable Use Policy. And this is AWS's policy for both acceptable and unacceptable uses of their platform. And for you to be able to have an account on AWS, you need to agree to abide by this policy. This policy covers several things, including some simple things that are prohibited, for example, sending unsolicited mass emails. That's not something they want you to do on their platform. And, if they were to detect that you were doing that, they could completely close your account. Also, hosting or distributing harmful content, things like viruses and malware, that's something that's also prohibited. One of the things that has changed within the Acceptable Use Policy, there are certain things you previously couldn't do without AWS permission that you can do. One example of that is penetration testing, and that's a type of testing that lets you see potentially where different holes are within your security. For example, you can see what ports are open on a specific server. It used to be you couldn't do that without talking to AWS, but they have provided a list of services where you are able to do this type of testing, and as long as the service you're wanting to test is in that list, you can do that without permission. But let's talk about one of the core security concepts that we need to understand when using AWS, and that is the concept of least privilege access. And the core of this concept is simply that when you're giving somebody permission to access AWS resources within one of your accounts, you should only grant them the minimum permissions needed to complete their tasks, and no more. You might say, well, this applies to other people. Well, one of the ways that you can even apply this to the work that you do on the accounts that you own is that AWS recommends that you don't use your route account as your account. There are specific things that the route account can do that no other account can do, so they recommend that you set up an IAM account to use on a daily basis, and this is just another example of least privilege access. But in terms of understanding how this plays out, if you're at a company, for example, and you have several employees and maybe one employee needs to have access to a majority of the systems, but another employee is only doing a maintenance task and only needs access to those two systems, well, you need to make sure that that employee only has access to those two systems. While it would be easier to give everyone the same access, we want to be sure that we are following least privilege access and only give them access to what they need to do the jobs that they have been given.

Chapter 1 AWS High-availability and Fault Tolerance

First, we're going to talk about the AWS Shared Responsibility Model, and this is something that everyone that uses the platform should understand at a detailed level. AWS puts it this way. Security and compliance is a shared responsibility between AWS and the customer. But since it is a shared responsibility, we need to have a good level of understanding of what is the responsibility of AWS and what is our responsibility as the customer. At a high level, we can put it this way. AWS is responsible for the security of the cloud, and the customer is responsible for security in the cloud. So AWS has a responsibility for those core systems that are running the entire platform. But the customer has control over the things that they are putting onto the platform and how they're using it. But let's look at this at a more granular level of detail. So first of all, for AWS, they have the responsibility for doing things like access control and training for their employees. So they control what employees can access what things and then make sure that they have the training to k what they need to do. As a part of that, AWS is responsible for those global datacenters and the underlying network. So when we looked at the Global AWS Infrastructure, they're responsible for those different availability zones and those regions and making sure that all of the connectivity exists between those. They're also responsible for the hardware for global infrastructure. So they're going to take care of replacing servers and switches and all the other bits of networking gear that they have. They also are responsible for configuration management for the infrastructure, so determining how bits of data get from one location to another, they control all of that. And they also handle patching of the cloud infrastructure and services. The core servers, the bare metal servers that are actually running some of your virtual servers or the servers that are running many of the services you use on AWS, they're in charge of patching those bits of cloud infrastructure. But, let's turn and look at what is our responsibility. It is our responsibility for individual access to cloud resources and training, so we need to make sure that we give least privilege access to those people in our company that need to access cloud resources, and we are then also responsible for training them to make sure that they know how to use the services that are available on AWS. Another key one, and one of the ones that is missed the most often, is the customer is ultimately responsible for data security and encryption, and we say here both meaning data going back and forth between different services or different locations, and data at rest, so data that is actually stored somewhere. It is our responsibility to be sure that we're following best practices there. For example, you could choose to put unencrypted data in S3; however, you also have the ability to configure the service to store that data so it is encrypted at rest. That is your responsibility. In addition, the operating system, network, and firewall configuration, that is on you as the customer. So if you're using Infrastructure as a Service, so let's say you're using just EC2 virtual servers, you're responsible for that operating system, including patching. If you're configuring your own VPCs, you're responsible for that network configuration and things like your access control list and security groups. You're also responsible for all the code that you deploy onto the cloud infrastructure. There's no way AWS can be fully responsible and own all of the code that you upload, so that is on you as the customer. You're also responsible for patching any guest OS and custom applications that you are leveraging, so it's important to note here, when you think of any solution that you're deploying on the cloud that you clearly understand what is the responsibility of AWS and what is the responsibility of the customer. And as you prepare for your Certified Cloud Practitioner exam, it will be important that you can look at scenarios and understand who has responsibility in those scenarios. If you're new to AWS, it can be challenging to k how to best create solutions on the platform. And over time, AWS has collected best practices that you should follow in these cases, and they have put these together into a single resource called the AWS Framework. And this framework is a collection of best practices that are organized across five key pillars that help you k how to best create systems that drive business value. So let's look at these different pillars. First of all, we have operational excElisece. This is what helps us k that we are both running, as well as monitoring our systems for business value. We want to be sure that we're being efficient with the time it takes to both build and deploy our solutions onto the cloud. Then, we're going to look at security, and this has to do with how we protect information and business assets, not just as we deploy them to the cloud, but also how we monitor to ensure that we are following the same standards of security throughout the life of those assets. Then, we're going to be looking at reliability. And later within this chapter, we're going to be covering the concepts of high availability and fault tolerance, and both of those fall under this overall pillar of reliability. This helps us make sure that our systems are up and running as often as possible. And next, we have performance efficiency, and this is how we can make sure that we are leveraging the resources that we have spun up on AWS and using only the amount that are needed to perform the tasks that we have for them. But we also want to be concerned with cost, and that takes us to the fifth pillar of cost optimization. We want to be sure that we're not paying any more than we need to achieve the level of business value that we need. And this is going to cover concepts like reserved instances, or spot instances, or different S3 storage classes that we have already discussed. , AWS has collected this information on a microsite about the Framework, and so as you go to get ready to build your custom solutions on the cloud, you can go review this resource to k how to best follow the recommendations within these five pillars. Now, we're going to talk about high availability and fault tolerance, and both of these fall under the reliability pillar of the Framework. And the quote that should be our driving force as we look at this concept comes from the CTO of Amazon, and he says that "everything fails all the time." So as we start to build anything, even when we're looking at a cloud platform that has many capabilities, we need to be sure that we are building king that failure can happen at any point in our architecture. So let's look at the concept of reliability on AWS, and we're really going to look at two different concepts.

The first is fault tolerance. This means that we're able to support the failure of components within your architecture. And if you remember, we have looked at this already when we've looked at services like SQS, and we looked at what we could do in using a queue to help mitigate challenges that would exist if some of our processing power were actually to go offline downstream from the queue. Then we also have another concept, and that's an overall concept of high availability. This means that we want to keep our entire solution up and running in its expected manner despite any issues that may occur. So let's look at how we build solutions on AWS with these things in mind. First of all, most managed AWS services provide high availability out of the box. And this is great because this limits the amount that we have to build. For example, if we're storing data using the S3 standard storage class within S3, that data is going to be stored across multiple availability zones. So we don't have to worry about building in a type of backup system for our production applications, that's already baked into the service. However, when we're building solutions directly on EC2, for example, vault tolerance must be architected. We have to figure out where to build that in. And this goes back to our different cloud deployment models. If we're looking at solutions that are Infrastructure as a Service, this is where we need to really consider fault tolerance because we're going to have to figure out how to build it into our custom solutions. And at a minimum, we know that we need to integrate with multiple availability zones so that we can deal with the potential failure of a complete availability zone within a region. Some services can enable fault tolerance in your custom applications, so there's a few to remember. We can build queues so that data or events that need to be processed can be held in the queue until they are ready to be processed. That way, if anything downstream goes offline, we can still have a functional application. But in addition, we also have Route 53, and we've discussed earlier within this path about how you can use Route 53 to detect if there are endpoints that are unhealthy and then route users to the right services that are available. Now, we're going to talk about the concept of compliance, and depending on what your experience is within the technical world, this might or might not be a familiar concept to you. But let's talk at a high level about some common different compliance standards. We have some standards like PCI DSS. And this is a standard for processing credit cards, so a part of the agreement that you'll have with the credit card companies for being able to process those cards if you're dealing with the data directly is that you'll need to meet certain standards. This means you need to be in compliance with PCI DSS. But we also have other standards like HIPAA, and you might have heard of this one during some of your trips to the doctor's office, especially if you're in the United States. This is a compliance standard for healthcare data in terms of privacy, making sure that only certain people have access to certain pieces of data. Then we also have some more technical standards like SOC 1, SOC 2, and SOC 3, and these are reviews of operational processes related to your datacenters. There also are some that are focused on government standards like FedRAMP, and this is standards for US government data handling. And then we have others that are standards around things like personally identifiable information with ISO 27018. These are different compliance standards, and there are services on AWS that can help you know how to navigate these compliance standards. So we're going to look at three in particular. First of all, we're going to look at AWS Config, which we have already mentioned within this path. And there are some conformance packs that are provided to help you with aspects of compliance. In addition, we have AWS Artifact, which provides access to AWS compliance reports. And then we have AWS GuardDuty, which provides intelligent threat detection so you can help monitor and try to detect if there are scenarios that are happening that are unusual that could lead to you being out of compliance. So let's look here at what we're going to cover over the book of this demo. First of all, we're going to examine how you get access to compliance reports utilizing AWS Artifact, and then we're also going to be looking at the concept of conformance packs within AWS Config. So I'm here in the AWS console, and the first thing I'm going to do is I'm going to navigate to Config. So I'll search for Config, we'll launch it, and I have already gone through the process of setting up Config here within this account.

In this point, we can go in and add rules directly, or we could go in and just go under Conformance packs.

Here in going under Conformance packs, we can simply go to Deploy conformance pack, and here, if we knew that we were going to be processing credit card information, we could look at the sample template for operational best practices for PCI DSS. And if we chose to deploy this template, this would help us monitor whether or not we were in conformance with PCI DSS standards. The next service we're going to look at is going to be AWS Artifact, and you can see here this is in my Recently visited services, so I could search for it or I can just simply click on the name. From within here, we're going to be able to get access to two different pieces of information. The first is going to be compliance reports, and the second are going to be any specific agreements that we have in place with AWS.

For example, if we were processing data, we would need to have a BAA agreement in place with Amazon, and we would be able to find that here. But here within AWS Artifact, we could go in and search for information around specific compliance standards that we were interested in. And in some cases we would be required to provide this information to other sources, whether they are governmental or just regulatory agencies for specific areas. And so, if we needed to find a specific standard, we could actually scroll through this list, find the right standard, and then we could actually get the artifact.

For many of these artifacts, we will need to sign a agreement with AWS to get access to that, but this provides a way to get access to these compliance reports that are needed. Now, we're going to talk through some samples scenarios to see how well you have done at absorbing the material that has been presented within this chapter. And first, we're going to be talking about Jack, and her company is building an application and they will be processing credit cards, and they're going to be processing those cards directly and not through a service. And so the bank that they're working with needs a PCI DSS compliance report for AWS. So for Jack, where should she go to get this information? Next we're going to be looking at Tim, and Tim's company is considering a transition into the cloud, and they do store some personal information, but they store it securely within their systems. And so Tim's CTO has asked him what the company's responsibility is for security on this once this is transitioned to AWS. So if you were in Tim's role, what would you tell his CTO? Next we're going to be talking about Elise, and she is a solutions architect at a startup. And they're building a new tool for digital asset management, and she's curious how to best leverage the capabilities of AWS in this application because they haven't built any applications in the cloud yet as a company. So, what resource would you recommend for Elise and her team to review? Next, we're going to run through a quick summary of what we've covered in this chapter, and then we'll run through the answers to these scenarios. We've covered quite a bit in terms of introducing security and architecture on AWS here within this initial chapter, and so let's quickly take a look back at what we've covered. First, we reviewed the core concepts around security and architecture. We also explored the AWS Shared Responsibility Model so we had a clear understanding of what is our responsibility as the customer and what is AWS's responsibility when we're leveraging the cloud. We also introduced the AWS Framework and we talked about the five different pillars that it provides for helping us k how to best take advantage of the platform. We then examined fault tolerance and high availability in terms of concepts around reliability that we need to put in place with our custom solutions. And, we also understood the provided tools for compliance. Let's now review our three scenarios that we presented within the previously. And so first we had Jack, and his company was looking to process credit cards and she needs to get a PCI DSS compliance report for AWS. So, where would he go? Well in this case, she would go to AWS Artifact. This service exists to be a solution for getting these types of compliance reports, whether we need to give them to organizations like banks we're working with or governmental agencies. The next scenario has to do with Tim, and Tim's company is considering a transition to the cloud, but the CTO has asked, what is the company's responsibility in terms of security? And so here, what should Tim tell his CTO? Well, he should tell him to review the Shared Responsibility Model, because in terms of this data that they're storing because they need to understand that concepts like data security and encryption are the responsibility of the customer and not the responsibility of the cloud provider. And finally, we have Elise, and Elise was curious how to best leverage the capabilities of AWS for this custom application that is being built at her startup. So, what would you recommend? In this case the Framework would be a great resource for Elise and her team to review because they're going to be able to look at the five different pillars and understand how to best take advantage of those capabilities within the cloud.

Chapter 2 AWS Managing IAM Users

Now, we're going to be talking about AWS identities and how we handle user management on the platform. I quickly want to remind you of a term that we introduced in the last chapter, and that is least privileged access. This has to do with granting permission for a user to access your AWS resources and granting them the minimum permissions needed to complete their tasks and no more. And we're going to see in this chapter how you actually make this happen on AWS. We will first be introducing a service called AWS Identity and Access Management, or IAM. We will then be reviewing the different IAM identity types. We will be enabling authentication, or MFA, and we'll also be introducing a service called Cognito that will enable you to take this kind of authentication and authorization to your own custom applications. So next we're going to be talking about the AWS IAM service. So at a high level, this is the service that controls access to AWS resources. So if you want to give someone within your company access to AWS, let's say through the console, for example, and you want them to be able to spin up EC2 servers but nothing else, this would be the service where you would go and both create their user and also go in and configure what that user can do. This service is free. So, unlike many of the other services on

AWS where you pay based on the resources you create within the service, this service is included at no charge to everyone that has an AWS account. And it manages both authentication and authorization, and we kind of mentioned both of these earlier, but we didn't use the terms. Authentication is what verifies users for us and lets them log in and manages their credentials, for example. Authorization is where we actually configure what that user can do. So we could say you have access to all EC2 actions but nothing else, for example. IAM also supports a concept called identity federation, and this allows us to use an external identity provider to actually handle that authentication portion of IAM. This might not make sense if you have a company of three people, you can simply go into IAM, and you can create all of those users and manage their permissions. But if you have a company of 2000 resources, all of whom need to have access to AWS, using an external identity provider, especially if you already have one in use, would totally make sense. You don't need to k how identity federation works for the exam, but you do need to k that this is an option when you're working within IAM. Next, let's talk about the three different identity types that exist within IAM, and the first one is just a user. And this is an account for a single individual so that they can access AWS resources. So let's say we have a new employee named Jack, and he is one of our developers. And so we could go in, give Jack an account, and then give her permissions to access specific AWS resources. However, let's say it's not just Jack. Let's say Jack is a part of a new development team that we're bringing on board, and there are five developers. Well, we could easily go in and create an account for each of those and then go in for each of them and then add the exact same permissions on all five of them.

However, there is a better approach. So another identity type within AWS IAM is a group, and a group allows you to manage permissions for a group of IAM users. So you would still need to go in and create users for each of them; however, you could then go in and add them to a group and then assign permissions to that group. And when we're talking here about an IAM identity, we're really just saying anything that you can assign permissions to. And so we've talked about users, we've talked about groups. We have a third type that's a little bit different, and that is a type called a role. And this enables us to either have a user or a service assume permissions to perform a specific task. We have run across this already, although we didn't call it out. So when you launch an EC2 server, it has the option to specify a role for that server. So you could say that that server needs to have access to an S3 bucket, for example. Maybe a part of the web application on that server is that it needs to upload photos to an S3 bucket. Well, it can't do that by default. The permissions won't allow for that. However, you could give it a role and then give that role the permission of being able to write to that S3 bucket. And this is what you can do utilizing roles within IAM. Next, let's talk about how you assign permissions, and this is what leads us to the concept of policies within IAM. So a policy is just a JSON document, and it defines permissions for and IAM identity, so again, a user, group, or a role. And it defines both the services that the identity can access and what actions can be taken on that service. So let's go back to the example that we mentioned earlier with our EC2 server. We might want it to have read access and write access to a specific S3 bucket. But we wouldn't want it to have permissions for everything on S3 because that would also include deleting the bucket, and that's not a part of what that web application would need to do. So we can say that it can access the S3 service for just the read and write actions on a specific bucket. Also, policies can be either customer managed or managed by AWS. So let's look at the portion. So if you need custom policy needs, for example, in looking at our web application, if we know that we want to give that web application access to read and write to a specific bucket, we could write a custom policy to do that for us. But AWS also provides what are called managed policies, and these policies have already been created. So if you want to have something like access to an entire AWS account or if you want to have something like full access to S3, there are policies that AWS has already created and they maintain and you can choose to use those without creating your own custom policies. Here is an example policy.

This is written as a JSON object, and you can go in and have specific permissions. You can see here that we do have a statement here that is allowing, and in this case it is allowing all S3 actions on a bucket and on the contents of that bucket. Right below that, we have another statement, and this is a statement denying access except for having access to that specific bucket and its contents. This is just an example policy. As a note, you will not need to know how to create a policy or any of the specific aspects of policies for the exam. But I did want to give you an example in terms of what one of these policies looks like. Let's next talk about some different best practices when working within IAM. The first best practice is multi-factor authentication. And what this does is, this gives an extra layer of security for our users. So when they log in, they will log in with their username and their password and then a token. And this is either going to be a physical or a virtual device that's going to generate that token for login. And you might have seen this in other places. Many web services offer multi-factor authentication, but this is just another layer of security. Then we also have least privilege access, which we have already talked about. So when we're creating and managing users within IAM, it is ideal that we encourage our users to set up multi-factor authentication, and we configure permissions to be geared towards least privilege access.

Now, we're going to begin working within the IAM service and we're going to be creating and managing IAM users. We will first be creating a new IAM user, we will then be configuring permissions for that IAM user, and then we can go in and create an IAM group and attach permissions to that specific IAM group. I'm here within the console, and you'll see here that IAM is listed under my Recently visited services.

So I could either search for it, I could use the Services but in this case I'll just click on IAM. Then, I'll make sure that Users is selected within the left navigation, and then I can go in and actually work to add a new user. So I'll select the option for Add user. We'll go ahead and give this name.

We'll say that Tom is a new developer on our team, and in this case we have a choice. We can choose to give Tom just programmatic access, and this is going to be with an access key and the secret access key, or we can choose to give console access, or we can do both. In this case, we will just grant Tom access to the AWS Management Console. And in this case, we could choose to have an password or a custom password.

And we're going to also set it so that Tom will have to reset his password when he logs in next to the console. So, this has what we need, so we can click the

Next option to set some permissions, and in this case you can see that it gives us several choices. We can go in and add the user to a group, and we're not going to do that just yet, we will do that later within this demo. We also can copy permissions from an existing user or we can attach existing policies directly. So I'm going to go ahead and select this option. , we have an option in here to look at the managed policies. You can see here that it's a managed policy when under Type you see that it is AWS managed. In this case, we can select those, or we could choose to create a new policy using the Create policy button.

In this case, Tom needs to have access to S3, so let's enter S3 and let's search for what policies are available. And we can see here that there are four different policies that show up, and they are all AWS managed. Well in this case, let's say that Tom is going to manage all of our S3 buckets for the organization, but we probably want to have him have S3 full access. So we can go ahead and attach that to his user. We could choose here to go in and add tags for this specific user. This could be valuable if you wanted to document different departments or different organizational structures, but we're not going to use this for the moment. And then we can go into Review. , when we get to Review, here we can see that we have the username, we have the different settings that we put in place, as well as the specific policies that were added in. , you'll notice here that it also has another policy for IAMUserChangePassword.

Because we selected the option to force him to reset his password, it automatically added this managed policy to his user. Once we have all of that in place, we can simply hit Create user. While this was relatively easy for us to put into place, this would become problematic if we were trying to enter in a bunch of users and trying to configure specific permissions for those users, especially if we weren't creating them all at the same time. So in this case, I'm going to go ahead and I'm going to click Close. And from here, we're going to navigate to the Groups option. And here we're going to create a new group. So here, we'll call this devteam1 is going to be our group name. And so from here, we're going to go in and add the AmazonS3Full Access managed policy. We'll then hit Next Step, and we can just create a group. There are not a lot of steps involved with creating a new group. This group is currently in place, but it has no users attached to it. So, let's navigate back over to the Users tab, and from here we're going to go into the user that we previously created. First, we're going to go in and we're going to remove the AmazonS3FullAccess managed policy because we're going to choose to not manage that within the specific user. But instead, we're going to manage it from the group. So we've detached that policy. Tom has no access to S3. But then we're going to go over to the Groups tab, and from here we're going to select the option to Add user to groups. We're going to select the devteam1 group, and then we'll say Add to Groups.

It’s important to note here that you can have users in multiple groups, that's part of the benefit. So, if you had someone that was a member of devteam1, they can be a part of that specific group. But let's say they're also an architect and they're part of the enterprise architecture group for the organization. You could create another group for that, and then you could add him to that group as well, and he would actually have the sum of both of the permissions from those groups. So we've been able to walk through and create an IAM user, we have configured permissions for that user, and then created a group, added them to the group, and then attached permissions to that AWS IAM group.

Chapter 3 How to Enable Multi-factor Authentication

Next, we're going to walk through the process of enabling authentication. Since this is one of the best practices for working within IAM, I want to be sure you have the tools to k how to do this. So, first of all, we will be enabling authentication for the root user, and then we'll talk about enabling authentication for an IAM user because these two processes are different and you do them from different places.

So I'm here in the management console. I have logged in as a root user, and it's important to note here that you can only manage the authentication for the root user as the root user. So I'm going to first go under my name here on the top, I'll open up the and I'll go to the option for My Security Credentials, and this is where you manage all of the information around security credentials for the root user.

This is where you would go to change your password as the root user or to update other things like your root access keys. I'm going to open the tab here for authentication, and I have an option here to activate authentication. I purposely went through on this and the IAM account that I'm using for this demo and removed authentication before this demo. You should always have authentication enabled, it's just a best practice that you should follow. So I'll hit the option here to activate authentication.

You have several choices here. You have a virtual device, you have a U2F, and you have other hardware tokens. For most of you, you should use the Virtual MFA device option. This is what will let you use an application like Google Authenticator or you can use a password manager that has it built in like 1Password. So I'm going to go ahead and select Continue, and you'll note here that it gives you a list of compatible applications, and this list lets you k other things that you can use to actually use authentication.

And you can go through and see also the hardware keys that are supported. But you can see here if you're using iPhone or Android, we have solutions like Authy, Duo Mobile, LastPass, Microsoft Authenticator, and Google Authenticator.

So in this case, I'm going to go back over to the console and I have a password manager that I use that I'll use to set this up. So the first step is going to be I need to actually show this QR code. , you also can use a secret key to make this work, but I'm just going to go ahead and select the option to show the QR code.

I will be deleting this authentication device after I finish this demo and then I'll be going back in and adding another one. So for security reasons, you wouldn't want anyone else to have access to this QR code because then they would have access to be able to generate your secret keys. So I'm going to go in, and here with my password manager, it lets me actually just drag over the QR code, and then it's actually entered in.

And so what they're going to have you do is you're going to type in two consecutive codes, and this is important because they want to be sure that it has properly synced, and so it's going to validate these two codes before this adds it to your device because once you add authentication to your device, you will have to log in with that, and if you need to disable that or if you lose access to the device that has it on it, you will need to contact AWS support to try and get access back to your account. So I'm going to go ahead and enter in the second code, and then I'll hit Assign MFA. And we can see that it has been added. And so I will have to enter in the secret code every time I actually log in to this root account.

Next, let's go look at how we do this for an IAM user. So I'm going to choose the user, and from within here, I'm going to go to the tab for Security credentials.

And you can see here that there is not an assigned MFA device, and so I'll go to Manage. I'm going to choose the same option for Virtual MFA device, and just as before, we would go through the process as we just did to show the QR code, add that to our password manager, and then enter in two successive keys. And then once we did that, we would have it in place for our IAM user so they would have to enter in that token every time they logged in.

Chapter 4 Amazon Cognito

So next, let's talk about a new service, Amazon Cognito, and this service gives you the capability to manage authentication and aspects of authorization for your custom web and mobile applications through AWS. So, let's go ahead and dive through this a bit more, and we'll talk about some use cases that tie in with it. So, Amazon Cognito, first of all, is a fully managed user directory service for custom applications. So, IAM deals with permissions for AWS resources, but what happens if you basically want to build something like IAM for your own custom applications? Amazon Cognito fills that role. But it does more than just that. It also provides UI components from many platforms. So if you want to have a sign in or sign up UI, for example, for your iOS application or for your React web application of your Android application, you can get those out of the box with Cognito. It also provides some advanced security controls to control account access. There's actually some pretty advanced functionality baked into Cognito that you can choose to take advantage of. But the other part of this, and this is where it intersects with IAM, is it also enables you to control access to AWS resources. Let me give you an example. Let's take the example that we mentioned earlier within this chapter. Let's say you have a web application that lets users upload some of their photos, for example. Well, you would want to be sure that users have access to a part of a specific S3 bucket that's just for them. Well, there's a way with Cognito that you can configure access to specific pieces of AWS infrastructure that you would want a user to actually have access to, but without having to have them sign up for an IAM user account. This can also work with social and enterprise identity providers. So we talked a little bit earlier about identity federation, and Cognito has wide support for that with providers like Google, for example. And let's be honest, most people seem to have a Google account, so you could let your users log into your custom application with Google and have that correspond to a Cognito identity. But it also supports Amazon and Facebook and Microsoft Active Directory, as well as any SAML 2.0 provider. And so Cognito gives you the ability to take this level of authentication and authorization and tie it into your custom applications. Now, we're going to talk through some specific scenarios to see how well you've absorbed the material that we've presented within this chapter. So first of all, we have Kate, and she manages a team of DevOps engineers for her company. Each member of that team needs to have the same access to cloud systems, and it's going to take her a long time to attach permissions to each user for access. So if you've got a team of 10 and need to go through to each individual user and go in and add those permission, she's just saying, man, it's taking a really long time. So, what approach would help Kate manage the team's overall permissions on AWS? Next we have Danny, and he works for a startup that is building a mapping visualization tool. And their EC2 servers need to access some specific data that's located within S3 buckets. He created a user in IAM for these servers, and then he uploaded those keys directly to the server. So is Danny following best practices for this approach? And if not, what should he do? Okay, finally we have Don, and he is leading the effort to transition his organization to the cloud. His CIO is concerned about securing access to AWS resources with a password, and he's asking Don to research approaches for additional security for their users. So what approach would you recommend to Don for this additional level of security? Next, we're going to go through the answers to these scenarios, as well as taking a quick look back at what we've covered within this chapter. So we have covered a great deal within this chapter, and before we dive in and go look at our scenarios, let's take a minute to look back at what we've covered. First of all, we introduced the IAM, or Identity and Access Management service, and as a part of that, we reviewed the three different IAM identity types, users, groups, and roles. We then were able to go through the process and enable authentication, and we did that after we went through and learned how to create IAM users and groups within IAM. And finally, we introduced Amazon Cognito, a service that enables us to configure aspects of AWS access for our custom applications and our users within those applications. So, let's take a look at our three scenarios. And first, we had Kate, and she was trying to figure out how to manage the team's permissions. So what would you recommend for her? Well, in this case, using an IAM group would give her one location for setting permissions for every single member of the team. However, we do need to add a caveat here. We have the principle of least privilege access. We want to be sure that no one has access to more than what they need. So for the members of Kate's team, it totally makes sense to have an IAM group. But, if she simply put all of her technical resources in a single group and gave them all admin access, that would not be following least privilege access. So we do need to use a group here, but we want to be sure we're not giving them more access than what they need. Next we have Danny, and he has EC2 servers that need to have access to S3 buckets, and so he just created a user in IAM and then uploaded that access key and secret access key to the server directly. So is he following best practices? No, he's not following best practices here. And part of the reason I bring this scenario up is even this week I was working with a client and I had pretty much this exact scenario happen where I found out the client was creating a user for something and uploading that directly to the server. The solution here is to use an IAM role within EC2 because this mitigates much of the security risk as opposed to taking the approach that Danny is taking. So roles are around for just this reason, to give our services the ability to work with other services on AWS, but to have it happen in a secure way. So, Danny should transition away from his approach and move to using an IAM role with specific permissions, in this case to access S3. Finally we have Don, and he's leading the effort to transition his organization to the cloud, but he needs an extra level of security. So, what approach would you recommend to Don? Well in this case, authentication should be what they move towards. This is built in to AWS, and because of that, it's something that should be relatively easy for them to implement. And it moves from the user simply having to just have a password, which obviously we k that passwords can be either hacked or they can be stolen from other services, and especially if users are using the same password on multiple sites, it can expose organizations to risk, but with authentication, you have to be able to provide this other token to be able to log in. So this should give Don the security that he needs.

Chapter 5 How to Integrate On-premise Data & Data Processing

Since we're talking about architecture on AWS within this book, one of the areas that we need to touch on has to do with data. So let's talk about what we are going to cover within this chapter. First of all, we're going to look at some different approaches for how you integrate data from your own data center. This is different than what we talked about earlier with just data transfer services. We're going to talk about how you integrate the data you have within your data center with experiences on AWS. We're also going to be examining approaches for how you process your data. Because in so many architectures, there is a processing step that needs to happen with data before you can dive in and fully analyze it. But then we'll be looking at the different data analysis approaches and services that you can leverage on AWS. And then finally, we're going to talk about an exciting topic, and that's how you integrate machine learning and AI, those capabilities that AWS has integrated into the platform into how you analyze your own data. So first, we'll be looking at how you integrate your data into AWS. And really, we're going to be talking about two different solutions. One of them is AWS Storage Gateway. And this is a hybrid cloud storage service, so merging together the storage you have between your data center and AWS. And then we have AWS DataSync, and this is an automated data transfer service, so different than Sball, but here we're talking about how you actually can set up a system from your network to sync data with AWS. So first of all, we have AWS storage Gateway, and this integrates cloud storage into your local network. And the way this works is you can deploy either a virtual machine or a specific hardware appliance running the Storage Gateway software on it onto your network, and it integrates with S3 and EBS. within Storage Gateway, there are three different gateway types that you can leverage. One of them is a tape gateway, then we have a volume gateway, and finally a file gateway. So let's look at those three different gateway types. So first of all, let's look at the file gateway. This enables you to store files within Amazon S3, but yet keep certain files local to your network on the storage gateway device so that you can have low latency local access, and it can have this local cache pull certain files or files that are the most recently accessed. Then we have the tape gateway, and this enables you to have basically a tape backup experience. So if you don't have a lot of historical IT background, one of the ways that data is backed up, especially within the enterprise, is on tape devices, and you can end up with a tape library where you have a machine that's able to read data and write data to multiple tapes. Well, the tape gateway for Storage Gateway basically acts like a virtual tape library or VTL. And so you can have that same backup experience still using your same backup software, but have it store to the cloud where it's much more durable than actually storing on tapes. Then we have volume gateway, and this gives you the ability to have iSCSI volumes that are needed by certain local applications, but have that data actually stored in the cloud.

So next, let's talk about AWS DataSync. , DataSync works by having the DataSync agent deployed as a virtual machine on your network. And in this case, we have wider support than just with the previous approach with Storage Gateway. Here, we can integrate with S3, EFS, and FSX for Windows File Server. We also here see a greatly improved speed of transfer due to this custom protocol that AWS has developed and different optimization. And in this case, you are charged per GB of data transferred between your data center and AWS. So next, we're going to cover how you process data on AWS and the different services that are provided for you to do this, because in many cases, if we're looking at how organizations analyze their data, there's a step for converting and preparing data before you actually analyze it, and that's what we're covering here. And we're really going to be talking about three different services. First of all, AWS Glue. This is a managed, extract, transform, and load service. We're going to be talking about extract, transform, and load quite a bit. And so we'll call this ETL , but let's talk about what that means. Usually it means you have data stored somewhere, you need to pull it out of where it's stored, that's the extract; then you have transform. This means you're going to do things like normalizing the data. Maybe you need to change the way that phone numbers are represented or group things together or change the structure of the data. You need to do something to change it. Then we have the load step. This means you're going to put it in a new location so that you can then go in and analyze it. So that's what we mean by extract, transform, and load. Then we have another service, and that is Amazon EMR or elastic map reduce, and this is a big data cloud processing suite, and it uses popular tools, and we'll cover what those tools are shortly. And then finally, we have AWS Data

Pipeline, and this is a workflow orchestration service across separate AWS services. So if you want to manage the process of how data gets from point A to point B, especially if it needs to go through a specific type of AWS service in the middle, this is where Data Pipeline can be very, very helpful. So let's look at each of these in turn. First of all, we have AWS Glue, and as mentioned, it is a fully managed ETL service on AWS. It also supports Amazon RDS, DynamoDB, Redshift and S3. So if you're looking for the AWS services that it integrates with, those are the ones that you can look at. it supports a serverless model of execution. That means you don't need to spin up specific servers or specific instances. You just use the service, and it handles the management of the infrastructure for you. Next, we're going to talk about Amazon EMR or elastic map produce, and what this does is this enables big data processing on Amazon EC2 and S3. And it supports popular open source frameworks and tools, which we will cover in just a bit. And it operates in a clustered environment without additional configuration, and this is really one of the benefits of the service. If you were choosing to do this on your own, just directly in Amazon EC2, there would be a lot of configuration that would go with getting these tools set up, and then having them operate in environments where you have more than one server where they all need to work together, and this is just included as a part of the service. And because it includes these popular tools and it has this clustered environment, it supports many different big data use cases. So let's look at the different frameworks that are supported. First of all, we have Apache Spark, Apache Flink, we have Apache Hive, Apache Hudi, we have HBase, and Presto. So if you look around in terms of what people are doing within big data, generally these are the tools that are involved. So EMR gives you the ability to take advantage of these tools without having to spend a massive amount of time configuring them. Next, we have Data Pipeline, and Data Pipeline is also a managed ETL service on AWS. It does manage the data workflow through AWS services, and it supports S3, EMR, Redshift, DynamoDB, and RDS. And there also are ways with this where it can integrate your data stores within your data pipeline.

So now that we've talked about how we get our data to AWS and how we process it, we're going to talk about how you actually analyze the data that you have within AWS. And so we're going to be talking about several services. First of all, we're going to be talking about Amazon Athena, which this is a service that lets you query data not stored in the database, but actually stored in Amazon S3. Then we're going to look at Amazon Quicksight, which is AWS's BI tool that gives you interactive dashboards. And then, we'll be talking about Amazon CloudSearch, and this is a search service that can be used for custom applications. So let's dive in and look at these. First of all, we have Athena, and this is a fully managed serverless service. So you don't have to configure any of the underlying infrastructure. You can simply leverage the service. And the great thing about Athena is it does enable you to query data stored within Amazon S3. So if you're taking more of what we would call a data lake approach and your company is just dropping a bunch of data within Amazon S3, there are formats that you can store it to make this process more efficient, but as long as you have that data there and you want to run a query a determine your sales year over year, you could do that utilizing Amazon Athena. You also can write queries just using standard SQL. So this becomes advantageous for people that are used to working within databases. They can take that exact knowledge and use it to then query this data stored within S3. With Athena, you are charged based on the data that you have scanned for a query. So depending on the amount of data that it needs to search through, that's how you are charged for Athena. So next, let's talk about Amazon Quicksight, and this is a fully managed business intelligence or sometimes you'll hear it called BI service. And this enables some dynamic data dashboards that are based on data that you have stored within AWS. There are a couple of different pricing models depending on how you're leveraging it. So we do have a and a pricing model, and it depends on how you're using it, how many authors you need versus how many people that are just reading the data. And there are multiple versions provided based on your needs, so like a standard version versus an enterprise version, and they have different capabilities and different cost points. And so you could dive into the documentation and see which of those fits your specific needs. So here you can see an example from Quicksight. This is using a sample dataset that you can load into the platform once you have an account. And this gives you the view of an author who is able to go in and actually create new visualizations based on data that you have stored within AWS. we're going to look at Amazon CloudSearch, and this is a fully managed search service on AWS. So in some cases, you want to go through an analyze data in dashboards like we were talking about with Quicksight. Or in some cases, you might want to run through a bunch of data like we have with Athena. But maybe you want to build a custom application and make a lot of data available to your users. And this is where CloudSearch can be very beneficial.

The great thing is it does support scaling of search infrastructure to meet your demand. So you can trust that this is a service that can scale for you. You are charged per hour and instance type of your search infrastructure. So this is where you need to know what size of an instance you need to have, and then you will be charged based on that. It does enable developers to integrate search into their custom applications. That's the goal of this service. So if you want to be able to have users search through a ton of PDF documents that your organization has stored or other custom information, this becomes a great resource to plug into.

Chapter 6 How to Integrate AI and Machine Learning

Now, we're going to talk about the exciting world of AI and machine learning on AWS because that you have data that you have included into the platform and you've processed it and you've analyzed it, there's a lot you can do with that data. And AI and machine learning give us capabilities that we did not have available to us previously for analyzing large amounts of data. So let's look at some of the different services. We are not covering all of the services on AWS. I'm simply covering a of the services, things that you will need to k for your Certified Cloud Practitioner exam. But there are many more. Services beyond what I'm covering here. So we're really going to be looking at three different services. The first is Amazon Rekognition. And this is a computer vision service that is powered by machine learning. And by this, we mean we can get insights out of images that we have stored on the platform. Then, we're going to look at Amazon Translate, and this is what allows us to translate text from one language to another. And finally, we're going to look at Amazon Transcribe, which is a solution, and this is also using machine learning. So let's look at each of these. First of all, we have Amazon Rekognition, and this is a fully managed image and video recognition deep learning service. That’s a mouthful. But what this is it means we can pull data and insights out of both images and video, and it can identify objects within images. So if you want to try to just quickly detect within an image, what objects are there, you can simply pass the image into Rekognition, and then you'll get a response back that contains those images. However, it does get interesting when we're looking at videos because you can identify objects and actions. So there are certain things that we wouldn't know if we only looked at a still image. But when we can see it in motion, we can begin to detect actions that are happening within that video. Another part of this is that it can detect specific people using facial analysis. So you can go in and say here's an image of an individual, and then you can have it go through a bunch of other images and see if it finds that person within those images. This falls under the category of facial recognition, and obviously people have a lot of varied opinions about facial recognition. But if you're looking to implement something like that or a custom authentication system, you could use Amazon Rekognition for that. It also supports custom labels for your business objects. So let's say you had a store, a retail store, and you wanted to take pictures of your products, and then you wanted to be able to detect when people are checking out just based on a picture of their shopping cart the items that they had, and we've seen Amazon implements similar technology with their Go stores, you could use custom labels for your business objects. There's a way to train the system to be able to detect those custom objects. Next, let's talk about Amazon Translate, and this is also a fully managed service. And in this case, we're translating from one language to another. And it currently supports 54 different languages. It also has some cool features, like just language identification. So it's possible that you're receiving information and you don't even k what the source language is. So you can get that from the service as well. And it also can work both in batch and in So in terms of batch, we mean we have a bunch of text after the fact, and we want to get it translated. And means we're actually able to stream that in and get the translation back as we're streaming it in. we also have Amazon Transcribe, and this is a fully managed speech recognition service. This means we want to take audio either again through or in a batch mode, and we want to be able to get that converted into text. So you can have this where you do record speech and then convert it into text in your custom applications. And one of the great things about this is there is a specific subservice here that is just for medical use. If you're working within the medical area and you want to be able to transcribe information that's going to have medical terms in it, there is a part of this service for that. This also supports both batch and transcription, and it currently supports 31 different languages globally. So next, we're going to dive through and look at some scenarios to help you know how well you have absorbed the material within this chapter. So first of all, here we have Ronald, and she is a data scientist for a financial services company, and they need to be able to process a dataset before they're able to analyze it. Ronald doesn't want to manage servers. She really just wants to define how the data needs to be transformed. So if that was the case, what service would you recommend to Ronald? Next we have Jesse, and Jesse is a member of the IT team for a biotech company, and she's currently working to identify an approach for controlled lab access. So in other words, they need to be sure that that lab stays locked except to the people that need to be in there. He wants to leverage AI to determine access based on facial imaging. So they have a camera posted there. He wants to be able to just scan somebody's face and determine if they need to be in the lab. So is there an AWS service that can help with this approach? And finally, we have Rob, and his company sells custom services around machine learning. His head of sales is trying to find a great way to visualize their sales data because they're tired of having to always go to the engineers to get this data. Their data is currently stored within Redshift, and so what they're doing is they're trying to figure out a way that they can have a experience for their salespeople to get access to that data. So what AWS service would allow this access to the data by resources? We'll cover the answers to these scenarios, as well as a review of what we've covered within this chapter, next. Let's quickly take a step back and review what we've covered. So first of all, we reviewed some different approaches for integrating data from your own data center. Then, we examined approaches for processing data, and we looked at concepts like ETL and the different services that can support it. We then explored some different data analysis approaches, everything from using Athena to look at the data in S3 to diving into using BI dashboards with Quicksight, as well as even looking at creating a custom search solution using CloudSearch. And then after that, we integrated machine learning and AI into our data analysis with services like Translate and Transcribe and image and video analysis with Rekognition. So let's look at our scenarios. First of all, we have Ronald, and he was looking to find a way to process data without having to manage the underlying servers. So what would you recommend? Well, in this case, AWS Glue becomes a great choice. It gives her that ETL capability, but in a serverless way without having to manage any of the underlying infrastructure. Then, we have Jesse, and she is looking here to use AI to determine access to the lab based on facial imaging. So what would she use? Well, in this case, Amazon Rekognition becomes a great choice because there are ways within Rekognition to store the images of specific people and then detect those in other images. So they could integrate that in to the camera that they have posted for lab access. Next, we have Rob, and he was looking for a way to have their sales team get access to sales data, but in a way where they didn't have to go in and query a database, so in an interactive dashboard. So what AWS service would allow this access to the data by resources? Well, in this case, that would be Amazon Quicksight. This gives a great way to create some dashboards based on the data that's stored in Redshift.

Chapter 7 Disaster Recovery Architectures

Now, we're going to be talking about disaster recovery on AWS. When we talk about disaster recovery, according to the AWS definition, we're talking about how we prepare and recover from a disaster, which can be any event that has a negative impact on your company's business, so this could include things like hardware or software failures, network outages, power outages, physical damage to a building like a fire, or flooding, human error, or some other significant event. Let’s look at this from two different perspectives. First of all, let's say that you are a company that has your own data center, and if that data center is located within your town and there is a major event that happens, let's say an earthquake, for example, and your network connectivity gets disrupted, how do you handle that? That's something you need to prepare for. But there also is another scenario. Let's say that you're leveraging the cloud for your infrastructure and let's say you're using AWS and you're basing everything in one of the AWS regions. While a complete region outage is pretty much unheard of at this point, it certainly could happen, especially if there were a significant natural disaster. So how do you prepare there? How can you leverage the AWS global infrastructure to limit the effect of this disaster? So over the book of this chapter, we're going to be talking about several things. First of all, we'll be helping to understand the need for a disaster recovery strategy, and then we'll be reviewing the four different disaster recovery approaches that are recommended by AWS. We'll then be exploring the factors that you need to k when you are selecting an approach, and then ultimately, we will be examining specific scenarios and disaster recovery needs. Next, we're going to talk about the different disaster recovery architectures that are recommended by AWS, and we're going to be looking at four of these architectures. So let's think about it in terms of a spectrum, and we're going to start on the far left of the spectrum with backup and restore. Then, as we move over, we will look at another option, which is called pilot light. Then we'll continue to move to warm standby. And then finally, we'll look at multisite. Let’s talk about what this spectrum means. First of all, we have a line going to the right that is increasing cost and complexity. This means that backup and restore is going to be cheaper than pilot light and much less complex than warm standby, for example. But we also have a line going the other way, and that is that we see recovery time decreasing. So if we look at backup and restore, it will have a longer recovery time than what we see with multisite. So let's begin to look at each of these to understand what that means. First, when we look at backup and restore. With this approach, you're going to take all your production data and you're going to back it up into Amazon S3. And you can do something like storing it in just the standard or an archival storage class. So let's say, for example, that you have a social network site. All of that data is going to be stored inside of S3. And we also can even take EBS data from our EC2 instances and store that in S3 as snapshots as well. What happens is if we have a disaster recovery event, we start a process to launch a completely new environment. So we're going to spin up all of our servers and our database instances, everything that we need for our environment, but we are not going to have any of that running until we get to a disaster recovery event. This approach has the longest recovery time. We're going to talk more about the recovery time, but in essence, that's just the amount of time it takes for us to get back up and running. And while this will have that longest recovery time, this approach will also have the least cost. Then we have pilot light, and with pilot light, you keep some key infrastructure running in the cloud. Not all of it, not enough for a complete environment, but just key components. And this is designed to reduce your recovery time over what you get with the backup and restore approach. It’s important to note, though, it does incur the cost of having this infrastructure running in the cloud. And for the items that you don't keep running, you'll have some AMIs that are ready so that you can launch your servers at a moment's notice if there is a disaster recovery event. it's important to note here that this does give you a quicker recovery time because the core pieces of the system, not the entire system, but those core pieces, are running and are kept up to date. So this is something that you would have to maintain over time. If we take another step up, we end up with warm standby, and what this is this is a scaled down version of the full environment running in the cloud. So this is not just core components, this is everything, but it's just maybe not everything at its full scale. So what we could do is we could have our critical systems running on less capable instance types. So if we need a super large server in terms of an instance type on our cloud environment, maybe we just go with a medium sized server here for warm standby, and what we can do is we can actually scale up if we have a disaster recovery event for those specific servers. But this also incurs the cost of running this infrastructure continually in the cloud. Let’s talk about the final option, which is multisite. Multisite means that at all times we have a full environment running in the cloud. So if we're talking about the scenario where we have our own data center, that means we have our data center up and running and we have the cloud up and running, and both of those are up the entire time. If we're talking about a cloud deployment model, this could mean that maybe we have data in the US East 1 region, and then we also simultaneously have it in the US West 2 region so that both of them are running all the time. This utilizes the instance types needed for production, not just recovery. This means that we have our full environment at its full scale up and running. And this can provide a near seamless recovery process. So if we want to make this seamless, if we do have one of those disaster recovery events, this is the best approach to take. However, this does incur the most cost over the other approaches, so you will pay more for multisite than you do any other, but you will also have the least amount of recovery time.

Chapter 8 Selecting a Disaster Recovery Architecture

Now, we're going to talk about the process that you take to select a disaster recovery architecture for your organization. So there are really two different terms that we need to know when we're considering our approach. The first is Recovery Time Objective, or RTO, and the second is Recovery Point Objective, or RPO, so let's take each of these in turn. First of all, we have Recovery Time Objective, and what we need to note here is that Recovery Time Objective is about time, and it's about our systems being back up and running. So this is defined as the time that it takes to get your systems back up and running to the ideal business state after a disaster recovery event. So for your organization, you might have a standard that says, if we have one of these significant events, we need to be back up and running within 8 hours. That would mean that your RTO is 8 hours, but we also have another term, and that is Recovery Point Objective, or RPO. This means the amount of data loss, in this case, specified in terms of time for a production system during a disaster recovery event. So what this means is, let's say we have our production system that's tracking our orders for our company. We might say that our Recovery Point Objective is an hour. That means that we might lose data for an hour when we have one of these significant disaster recovery events. But by the end of the hour, we are back to receiving full production data. And so this is defined in terms of time, and it's important to remember that Recovery Point Objective is all about data, whereas Recovery Time Objective is all about time. Let’s look at this in terms of our different scenarios again, and this is the visualization that we've already seen. In both RTO and RPO, we're going to see the least of those values when we're looking at multisite. But as before, that's going to mean that we have the highest cost. With backup and restore, we will see the least cost, but we will also see the highest RTO and RPO. And then with pilot light and warm standby, you can customize these based on what your needs are in each of those specific areas to determine what systems need to be up and running. For example, with pilot light, you might choose to have key production databases up and running with a smaller scale in the cloud so that you can decrease your overall Recovery Point Objective. Let's now review some scenarios to see how well you have absorbed this material. So first of all, we're going to have Rob, and his company runs several production workloads within AWS. And he is tasked with architecting the disaster recovery approach. In this case, they don't have an approach yet, and he's trying to figure out which approach they should adopt. His organization wants there to be a seamless transition if there is a disaster recovery event. For Rob, which approach would you recommend for his company? Next we have Judit, and she is at a startup, and they do not currently have a disaster recovery approach. They are leveraging the cloud, but they don't have an approach in place yet. And for her company, the goal here is to minimize cost, and that's going to be more critical than minimizing the RTO or recovery time objective. In the case of Judit and her company, which approach would you recommend? Finally, we have Elenor, and she is documenting her company's disaster recovery approach. And for them, they keep a few key servers up and running in AWS in case of an event, and these servers have some smaller instance types than what you would normally have within a production environment. So in this case, if she's trying to document this, which disaster recovery approach most closely matches this scenario. Next, we're going to be summarizing what we've learned within this chapter, as well as giving answers to these scenarios. We have covered several different aspects of disaster recovery on AWS. And this includes, first of all, understanding the need for disaster recovery strategy. We talked about whether you are running in your own data center or even in the cloud, there is still a need to have a disaster recovery approach. And then, we reviewed the four different disaster recovery approaches that AWS recommends all the way from backup and restore up to multi site. We then explored the factors to k when you are selecting an approach. We talked about RTO or recovery time objective and RPO, recovery point objective. And we went through and examined some specific scenarios and disaster recovery needs. Since we looked at those scenarios, let's go dive in and find out what the answers are. So our first scenario had to do with Rob and his company, and so they were really wanting to have a seamless transition. So which disaster recovery approach would Rob use for his company? Well in this case, it would be multi site, and the reason here is they want to have as seamless of a transition during a disaster recovery event as possible. That means they would need to leverage the multi site approach. They would have full production instances up and running both in their own data center, as well as within AWS.

Next we have Judit, and her company was the startup that wanted to minimize cost, but still have an effective disaster recovery approach. So what would you recommend? Well, in this case, the answer is going to be the backup and restore approach. And the reason is that Judit's company is looking to minimize cost, and that is the driving factor. She’s going to have more time to get the systems back up and running, so a higher RTO. So here,cost is what we're optimizing for, and backup and restore is the most approach to disaster recovery. So next, we have Elenor, and she is documenting her company's disaster recovery approach. They keep a few key servers up and running in AWS with smaller instance types and what production would need. So which approach is this? You might be confused here if the answer is warm standby or pilot light because it seems to include examples of both, and you would be correct. However, the key thing to note here is that they only keep a few key servers. They don't have a complete environment up and running. They are using smaller instance types than what production would need, and that might lead you to think oh, this is warm standby. But when we see here that they're only using a few key servers, we would then say that this is the pilot light approach because they are only keeping kind of core systems up and running in the cloud, and they will have to launch all of the other instances that support their environment if there is a disaster recovery event.

Chapter 9 How to Scale EC2 Infrastructure

Now we're going to be talking through how you architect applications on Amazon EC2. We'll be walking through several different aspects of this. We'll start off by reviewing scaling approaches and the services that support scaling for Amazon EC2, and we have spent some time talking about this already, and we will revisit the concepts of horizontal and vertical scaling, but we'll go even deeper and talk about the services that enable that. Then we will examine approaches for how you control access to your Amazon EC2 instances. So making sure that it is only available to whoever you want it to be available to. Then we're going to be exploring services to protect infrastructure from hacking and attacks, because we live in a world where this has to be a consideration, when you're deploying any infrastructure out to the public internet. Then we'll be introducing the developer tools that are provided by AWS to help with any of our development work on the platform. And then finally, we will be reviewing approaches for launching predefined solutions on Amazon EC2. First, we're going to be talking about scaling our EC2 infrastructure. And while we have already introduced this topic, we're going to be taking a bit of a deeper dive. So let's quickly revisit the concepts that we have already covered. We talked about different ways that you can scale on EC2, the first being vertical scaling or scale up, and that means if we've reached the end of the demand that our server can support, we can simply make it a bigger server, make it to have more resources by having a larger instance type. But then we also have horizontal scaling, and this is where we scale out. Instead of making our server bigger or making our servers bigger, we're going to actually just make more servers, and then we'll be able to route users to whichever server has the lowest demand. This is horizontal scaling, and we talked about the fact that in the cloud, horizontal scaling is a much more sustainable approach than vertical scaling, and this is baked into EC2 with a couple of specific services. So first of all, we have the concept of an Auto Scaling group, and this allows us to have a group of EC2 instances that work together with shared rules for scaling and management. Then we have a concept that we have already introduced, which is an Elastic Load Balancer, which allows us to distribute traffic across multiple targets. So let's learn more about the new concept that we've introduced here, Auto Scaling groups. So within EC2 Auto Scaling groups, first of all, there is a launch template that defines the instance configuration for the group. So if you know, for example, that you're going to want to have a Windows 2019 server, and it needs to be a certain instance type, and you know that it's going to need to have this certain security group associated with it, you can define all of those settings within the launch template. And then every instance that gets launched within the Auto Scaling group will have those characteristics. You also can define the minimum, maximum, and desired number of instances within the Auto Scaling group. So let's say that we have a web application that we're running on

EC2, and we k that we need to have at least two servers running, so two can be our minimum. And let's say that our maximum here is six, but we want to have four as our desired number. Just on a normal day, at a normal point in time, it needs to have four servers up and running. You can define those, and the Auto Scaling group will manage the group of instances to that. It performs health checks on each instance, so you can define how it knows if an instance is healthy or not. If you have a web application, for example, that's running on this EC2 server, you can choose to give the specific URL that it should check against that server to make sure that it's working properly. If it returns a proper status code, it assumes the instance is healthy. But if it doesn't, then it would assume the instance is unhealthy, and it would take action. An Auto Scaling group exists within one or more availability zones in a single region, and you usually have it span multiple availability zones because this adds a level of fault tolerance that even if you had a complete availability zone go down, your Auto Scaling group could still be up and running. This can work with on demand and spot instances. Let's look at the following scenario. So first, we have a region, and in this case, this will be And inside of our region, we have configured a VPC with an internet gateway, and this VPC will house our custom application. And we have supported that across two different availability zones, a and b, and we created our Auto Scaling group, and it exists within both availability zones. And we chose to have a desired capacity here of two. So it's going to try to keep it balanced between the different availability zones, so it will launch one instance in availability zone a and one instance in availability zone b. we want to have users be able to be routed to a server, the server that has the most room to handle their request, and so we're going to have a specific type of ELB called an Application Load Balancer that's going to be between our users and the specific servers that they're going to be accessing. It knows how to work effectively with the Auto Scaling group, so it knows what instances are healthy and which ones are not. So in the beginning, all of these instances are healthy, so the Application Load Balancer can route user traffic to either of the servers. However, let's say that something changes and the server that we have in availability zone a is no longer healthy. Well, the Auto Scaling group will note that, it will let the Application Load Balancer k that, and it will stop sending traffic to that particular server while the Auto Scaling group terminates that server and then goes in and actually starts a new server. Once that startup process is complete, it can communicate with the Application Load Balancer and let it k that it is safe to send traffic back to availability zone a. Let's talk about another concept for how we scale, and that has to do with the AWS Secrets Manager. So you need a way when you're scaling out to multiple servers to securely integrate things like credentials and API keys and tokens and really any other secret content in a way that can't be compromised. This wouldn't need to be with our startup code, for example, on the server. It wouldn't need to be with our custom code that might be sitting in a repository somewhere. There needs to be a safe way that we store this information. For example, if our web application we're going to work with RDS as a database, it would need to have a secure way to get the credentials for that database. And Secrets Manager integrates natively with RDS, DocumentDB, and Redshift, and it can auto rotate credentials with these integrated services. So you wouldn't want to keep the same username and password for RDS for 3 years, for example. You do need to change it periodically just to make sure that there are no vulnerabilities. And so this is included by default within Secrets Manager that you can configure this auto rotation. And another benefit is that it does enable you to have access control to those secrets. So you know exactly what servers and what applications have access to which secret values that are stored within the Secrets Manager.

Chapter 10 How to Control Access to EC2 Instances

Now that we've talked about how we actually scale our EC2 instances, we need to take a minute and think about how we control who can access and can't access our EC2 instances. We're going to talk about 3 different ways that we can go about this process. The first is by using EC2 security groups, and these are controls for your resources within your VPC. Then we're going to talk about network ACLs, and these control both inbound and outbound traffic for entire subnets within the VPC. And then finally, we're going to talk about AWS VPN, which gives us the ability to create a secure encrypted tunnel between our network and the AWS infrastructure. So let's go in and look at each of these in turn. So first we have security groups, and these basically serve as a firewall for your EC2 instances, and they have the ability to control both inbound and outbound traffic. Unlike what we're talking about with access control lists, these don't work at the subnet level, or even at the VPC level. These actually work at the instance level. So you will associate a security group with a specific EC2 instance, and they can actually belong to multiple security groups. So you might create some security groups for common purposes. For example, if you have web servers, they might have certain configurations that are enabled on them. And then every web server that your company uses can have that security group attached. VPCs do have a default security group, and so if you don't choose to specify any other security group, that security group will be associated with your instance. But other than that use case, you have to explicitly associate an EC2 instance with a security group. It doesn't happen by default. It doesn't happen to everything within the same subnet. And by default on security groups, all outbound traffic is allowed. That means that your server can send any information out to the internet. When we have network ACLs, they work a bit differently. They work at the subnet level within a VPC. So when you define your network configuration, every server that gets spun up within a subnet will adopt the ACL for that subnet, and this allows you to both allow and deny traffic. Each VPC comes with a default ACL, and that default ACL allows all inbound and outbound traffic. However, if you go in and create a new custom ACL, it, by default, denies all traffic until you have gone in and added rules to that custom ACL. Let’s talk next about a different approach, and that is utilizing AWS VPN. And what this does is, this enables you to create an encrypted tunnel into your VPC. So you might not even want to make your VPC available to the public internet, but you still want to have a way to get access to manage the servers that are within that VPC. So you can use this to either connect your data center, or maybe you just want to connect a single machine to your VPC. And it supports both of these in two different services. First of all, we have the VPN, and then second we have the Client VPN. So let's talk through this and let's see what a VPN example would look like. So let's say that we have our own corporate data center, and we have several servers that need to interact with some EC2 instances that we have spun up within our VPC on AWS. Well, we could utilize the AWS VPN service. We would create a customer gateway, and we would need to enter that information into the service, and then we would have a VPN gateway that would exist within our VPC on AWS. And once these things are in place and once they know how to communicate with one another, we then would be able to have encrypted traffic traveling back and forth between our corporate data center and AWS. You may be asking, how is this different from AWS Direct Connect? And that's a great question. With AWS Direct Connect, you have a direct connection to the AWS global infrastructure that doesn't have to go over the public internet. However, when you're using AWS VPN, that traffic does go over the internet; it is just encrypted the entire way. Next, we're going to talk about how you protect your infrastructure and your data from different cyber threats that exist, and there are several services that support this for us on AWS. And first we're going to be looking at AWS Shield, which is a managed distributed protection service for apps on AWS. Shortly, we'll go through a definition of what that means. Next, we're going to be looking at Amazon Macie, which is a data protection service that is powered by machine learning, and so it enables us to watch things that we maybe couldn't watch if we were looking at manual processes. And then we'll be looking at Amazon Inspector, which is an automated security assessment service for EC2 instances. Before we dive into these, let's first give a definition to that term and that is distributed denial of service, or DDoS, and this is a type of attack where a server or group of servers are flooded with more traffic than they can handle in a coordinated effort to bring the system down. And in the time that we live in, this is something that can happen. If you want to have protection against this, AWS has provided a service that makes that available, and that service is called AWS Shield, and it does provide the protection against the DDoS attacks for your custom applications that are running on AWS, and it enables ongoing threat detection and mitigation. So it's not one of those things you simply turn on when you need it. It's something that you keep running, and it can create a scenario where it knows when a DDoS attack is happening. There are two different service levels for this service. We have AWS Shield Standard, and then we have AWS Shield Advanced. So depending on what your needs are, you might choose a different service level for this particular service. Then we have Amazon Macie, and this utilizes machine learning to analyze the data that we have stored within Amazon S3. One of the big benefits of this service is without us having to classify the data, it can detect personal information and intellectual property that is stored within S3. Many organizations worry that the information they have stored somewhere in the cloud can leak out, and this service is designed to continually watch and categorize data to make sure that that doesn't happen. It does provide a level of dashboards to show how your data is being stored and accessed, and it can provide alerts for you if it detects anything unusual about data access. This is the type of detection that we probably couldn't do if we were monitoring manually, but it can use machine learning to do anomaly detection and find out when there is a strange pattern of how our data is being accessed. Then third, we have Amazon Inspector and what this does, is it enables scanning of our Amazon EC2 instances for security vulnerabilities? So if we want to be sure that we are keeping our instances up to date, we're being sure that they're patched for any critical vulnerabilities that have been released in the community, if we want to be sure that we're following best practices, here we can utilize Amazon Inspector and run those in an manner. Here you are charged per instance, per assessment run. So if you choose to run an assessment across 10 of your instances, you would be charged accordingly. There are two types of rules packages that are included with Amazon Inspector. The first is a network reachability assessment. This is where we want to understand what is available to the internet from our servers. And then we have host assessment, and this is what is going to check our EC2 instances to make sure that they have been patched for critical vulnerabilities, and make sure that no common configuration errors have been made within the server.

Chapter 11 How to Deploy Pre-defined Solutions Using Developer Tools

So in some cases we're going to create a custom application, and we're going to go through all of the configuration needed to make that run in the cloud. But in some cases, we want to deploy predefined solutions on our AWS account. So how do we do that? Well, there is a couple of different ways depending on how you're going about it. First, if you're an organization that is looking to make certain services on AWS easy to deploy, you can leverage AWS Service catalog, and this is a manage catalog of IT Services on AWS for an organization. But then we have AWS Marketplace. This is a catalog of software that's going to run on AWS, but from third party providers that have made their specific configuration available on the cloud. Within AWS service catalog, it is targeted to serve as an organization service catalog for the cloud. So let's say that you have several different groups within your organization that need to spin up a WordPress server, for example, and that's going to be the blog that they're going to use for their different department. Well, if that's the case, you can configure everything that's needed to get this up and running and they can be able to launch it with pretty much just a click within the service catalog. This can include everything from maybe a single server image to even a custom application, so this can handle a wide variety of infrastructure needs on AWS. And one of the things it can do is it can enable you as an organization to leverage services that meet compliance. So if your IT group has already constructed a specific way to launch a certain type of server and make sure that it follows all of your organizational and industry best practices, you can then allow others to take advantage of that learning without having to recreate the wheel. It does support also a life cycle for services that are released in the catalog. So you could have a version 1 of a specific service and then update it to version 1.1 and everyone that's using that would be notified that there is an update to the service that they're using. Next, we have AWS marketplace and there are many similarities between service catalog and between AWS Marketplace, but marketplace is geared at vendors, or ISVs, that are going to publish their software for use on AWS. So this is a curated catalog of solutions for any AWS customer to run, and it provides everything from AMIs, CloudFormation stacks, and even solutions through the marketplace, and it enables some different pricing options to overcome some of the challenges of licensing in the cloud because many vendors had licensing that was tied more to physical servers, and so they provide different types of licensing terms for the cloud. the charges for what you use on AWS Marketplace, and some things are free that are just offered by the community and some things have an additional charge on top of your AWS infrastructure costs, and these charges will all appear in one place on your AWS bill. And here, you can see an example from the AWS Marketplace.

This is looking at some different public sector data that is available that you can access using the AWS marketplace, but there are many different categories of services that are available, and there are many, many different vendors that are supported within the marketplace.

In building solutions for the cloud, AWS has produced a suite of services to try to make that development process as easy as it can be, as well as making sure that it is easy for developers to follow best practices. So let's quickly review this suite of tools that are provided by AWS. First, we have AWS CodeCommit, and this is a managed source code repository using Git. Think of this as an alternative to using GitHub but one that's deeply integrated with AWS. Then we have AWS CodeBuild, and you can think of this as a build service, or a continuous integration service. This can run the build commands for your custom application to then create your output artifacts. Then we have AWS CodeDeploy, and this is a service that will take care of the deployment out to many different AWS services. And then we have AWS CodePipeline. So this knows how to work with all of the other services we've mentioned previously to create a pipeline so that we can go through and look at the entire process of building, testing, and then ultimately deploying our applications. But they have made it even easier with a service called AWS CodeStar, which gives us a great way to bootstrap this entire process for our custom applications. So let's look at each of these. So first of all, we have AWS CodeCommit, and this is a managed source control service on AWS, and it does utilize Git for repositories. However, unlike many Git providers, it uses IAM policies because it is an AWS resource. So you would control access to this just like you would control access to any other AWS resource. And it does serve as an alternative to GitHub and Bitbucket. However, in most cases, AWS has built a way to leverage GitHub with each of the other development services if you choose not to use CodeCommit. Then we have AWS CodeBuild, and this is a fully managed build and continuous integration service on AWS. And because this is fully managed, you don't have to worry about maintaining infrastructure. You only are charged per minute for the compute resources that you utilize. And here this can take care of building your application and creating those output artifacts. And then we have AWS CodeDeploy. And this is a managed deployment service for deploying your custom applications. And we certainly have been talking about EC2 in this particular chapter, but it works with more than just EC2. So you can deploy out to EC2, but also to Fargate, which is a container service. We also can look at AWS Lambda and even your servers that you're leveraging. It does provide a dashboard for deployments within the console, so you can keep track of the progress of your deployments and even kick off new deployments directly from the console. Next, we have AWS CodePipeline. So this takes everything a level up and creates a fully managed continuous delivery service on AWS. And so it knows how to integrate with the previous services that we've mentioned to handle building, testing, and deploying. So it can work with your source code in CodeCommit, it can then build and test within CodeBuild, and then it can use CodeDeploy to deploy out the output artifacts. It does integrate with other developer tools, including GitHub, earlier. Next, let's talk about AWS CodeStar. This is a workflow tool that automates the use of the other development services that we already have talked about. So it can create a complete continuous delivery toolchain for your custom applications, and it can do it with just a couple clicks and a couple configuration values. It also provides some custom dashboards and configurations within the AWS console. So if you want to k effectively how all of these different pieces work together, that's what is included within those custom dashboards. And here the great thing about CodeStar is you are only charged for the other services that you leverage. So it is one of the services, like some of the others that we have mentioned, where you don't pay for this service. It is simply a tool to make it easier to use other AWS services. But let's take a step back and let's look at some scenarios to see how well you've absorbed the content within this chapter. And so first, we're going to be looking at Elise and Elise is a solutions architect at a traditional financial services company. they recently made a transition to AWS and they've gone all in, but they are concerned, they want to be sure that each department is following best practices, and they want to create some compliant IT services that other departments can use, but without the risk of them going rogue and doing something that could cause them trouble down the road. For Elise's situation, what service would you recommend for Elise and her team? Next, we have Tom and his company leverages AWS for multiple production workloads, and recently, they had downtime due to one of their applications failing on EC2. Tom’s job is to avoid downtime if an instance stops responding. So what approach would you recommend to Tim to solve this issue? Finally, we have Jack and Jack's company deals with sensitive information from its users, and they have some reasonable policies in place for the data that they have stored in S3, but she's a bit worried that if some of these policies accidentally get changed, they might actually leak out some data or have a potential data breach and she's worried about that going unnoticed. So if you were Jack, what would you recommend to her company? Next, we're going to be walking through a summary of what we've covered within this chapter, but we're also going to be taking a look at the answers to each of these scenarios. First let's review what we have covered over the book of this chapter. First of all, we reviewed the different scaling approaches and services for Amazon EC2. We reviewed horizontal versus vertical scaling, and we talked about leveraging auto scaling groups and ELBs. We then examined approaches for controlling access to Amazon EC2 instances, and we talked about ways that you could leverage security groups, network ACLs, and AWS VPN. We then explored services to protect our infrastructure from hacking and attacks, so we looked at services like Shield, and Macie, and Inspector. And then we introduced the developer tools on AWS. We looked at Codecommit, and Codebuild, and Codedeploy, Codepipeline, and Codestar. And then we reviewed approaches for launching predefined solutions on Amazon EC2 from both the AWS Service catalog and the AWS Marketplace. So let's take a look at our scenarios. So we had Elise and Elise was trying to figure out how they could create compliant IT services that other departments could use. So what would you recommend? Well in this case, AWS Service catalog would be a great choice. These services are designed just for use within their organization, so here is where service catalog would work well, whereas marketplace would be more for services that can be launched on AWS. Next, we had Tom and he was trying to figure out how to deal with this downtime on EC2. So what would you recommend for Tom? Well, in this case, Tom needs to look at creating an EC2 Auto Scaling group alongside an Elastic Load Balancer because this would enable them to respond automatically if a server goes down and spin up another one. In addition, they could have multiple servers running at any point in time and route users to the healthy servers and not to the unhealthy ones. Finally, we had Jack and Jack's company is dealing with this sensitive information and she is worried about a breach going unnoticed. So what would you recommend to Jack and her company? Well, in this case, Amazon Macie would be a great choice because it gives them the ability to use machine learning to classify their data, find the data that's sensitive, and then monitor that data and its access patterns, and it can proactively alert them if it sees any anomalies happening within any of those patterns.

Chapter 12 How to Register & Study for the Exam

Congratulations. You have made it to the end of this path covering all the information that we have to prepare you for your Certified Cloud Practitioner exam. In this particular chapter, we're going to walk you through the core information about the exam. Next, we'll help you understand what you need to k to register for the exam. And then, we will walk you through tips on how you study and prepare for the exam. First of all, this is a proctored exam. Most of you are going to register online and then you're going to go to a testing center to actually take this exam. There are two different types of questions that you will find on this exam. First of all, you will have multiple choice where you're able to go through and see a list of possible answers and select the right one, some of them will also be multiple answer. So for multiple answer questions, you will have several options and you will need to pick the two or three options that are correct from the list or are incorrect. So with these two types of questions, you will end up with a score anywhere between 100 and 1000. A score of 700 or higher will be passing. This exam covers four key areas, and don't worry, we have covered these areas in this path.

The first area has to do with fundamental cloud concepts, and this represents roughly a fourth of what is going to be on the exam. This is what covers general information about the cloud. Then the next section has to do with technology and technology will represent about a third of the overall exam. This is where you need to understand concepts like the AWS global infrastructure, as well as all of the different AWS services that we've mentioned. Following this, we have security and compliance, and this particular category also represents about a fourth of the overall exam, and this has to do with concepts like the shared responsibility model, as well as how you can access AWS compliance reports. And then the final category is billing and pricing, and in this category, you need to understand the tools that are provided for you, things like the Cost Explorer, and the TCO calculator, and the simple monthly calculator, and understand how these can actually provide value to you when you're using AWS, and this represents around 15% of the overall exam. So if you have the knowledge that we have provided in these four areas, you should be prepared to go and take your AWS Certified Cloud Practitioner exam. Now, we're going to talk about how you actually sign up to take your AWS Certified Cloud Practitioner exam. The way that you do this is you will go to the Cloud Practitioner Certification page, and from there you will click on the button to schedule an exam. You will then be taken to the AWS training and certification portal. From here, you will need to create an account if you don't have one and then sign into this portal. It is important to note, if you have a partner account, you will need to be sure that you select that when you log in, as opposed to just a standard log in. But once you're in, you'll have the opportunity to select an exam that you're eligible for. You'll obviously be choosing the Certified Cloud Practitioner exam. There is an option for this exam you can take this at your home, assuming that your computer qualifies. So if you want to check that, you can go through the process of signing up with Pearson VUE and go through and see if your computer qualifies. This is not available in all countries, unfortunately, so you might or might not be able to take advantage of this depending on where you're located. But either way, you can either take it through an online proctor where you are at home, or you can go into a physical testing center, and different testing centers are supported based on the countries that you're in. But once you go through this process, you will be signed up and ready to take your AWS Certified Cloud Practitioner exam. So next, we'll be diving into how you study and get ready for the exam. Now we're going to talk about how you prepare and take your Certified Cloud Practitioner exam. First of all, we know that there are several steps to getting your AWS Certification. We need to learn. That's hopefully what you have done through this book, then sign up, and that's what we covered in the previously. But we get to both study, and actually taking the test. And those are the things that we're going to be focusing on here. First, let's talk about studying for the exam. So let's review each of the different areas that you need to k for the exam. Let's look at cloud concepts. So you need to k within this area the differences between traditional data centers and cloud platforms and understand why those differences are important. We also need to look at how AWS organizes its infrastructure globally and look at how scalability differs between the cloud and traditional data centers. Also know things like the difference between capex and opex expenditures. Then let's look at security. So within security, you first need to understand the shared responsibility model from AWS that we covered in this book. We also need to review the highlighted best practices for securing your account. Then we need to review the options for securing traffic within a VPC. And this is when we look at concepts like EC2 security groups and access control lists for VPCs. We also need to understand IAM and the different identity types, and also understand the principle of least privilege access. Then, within billing and pricing, you need to review the tools that can help you understand your AWS costs. You need to be able to explain each of those tools and when you would use them. You also want to understand the most ways to leverage core services, understand the cost impact between different S3 storage classes, or between EC2 and Reserved Instances and Spot Instances. You also want to review how costs differ from traditional data centers and review the different ways that organizations can manage and review their costs and also understand the different support plan levels. Next, we'll look at technology, and technology covers, again, a third of the exam for your Certified Cloud Practitioner certification. So first of all, you need to review each of the AWS services that are included in the services list that we have provided with the second and third book in this path. That's going to be critical. You should ideally be able to cover up the name of the service, read the description, and be able to give the name. That's going to help you know if you really understand what that service is. You also need to implement some basic solutions using the services that we've covered. So we have included several demos within this path, and you can choose to explore those services from within the AWS Console. You also want to review those architectural principles for fault tolerance and high availability that we talked about also within this book, and look at the different scalability approaches. So now that you've studied that, what do you do? Well, once it gets to be the day where you're taking the exam, after you've signed up for the exam through what we covered, we're ready to talk about how you take the exam. So first of all, let's go through some general best practices for this certification test. First of all, you do need to take time to analyze each question for its intent. There's a reason AWS is asking a specific question. So if we have spent time talking about a specific concept and you see a question around that concept, try to remember how we covered it within the book. We also want to review what is required for the answer on each question, and this is critical because you have both multiple choice and multiple answer. So it could be that you need to select two or three answers for a specific question, so make sure that you review that for each question. Another thing to note here is just skip a question if it takes too much time, and you can do that by leveraging the review capability. You can mark a question that you need to go back and review later. Then you can go back and spend time on those questions that you weren't sure on. And once you go through that review phase, just guess if you don't know the answer. Because this is a multiple choice and a multiple answer test, you still have the opportunity to get things right, even if you don't really know the answer. Next, you want to examine the clock after each 10 questions, so just make a mental habit of saying, question 10, let me look at the time. Question 20, let me look at the time. This way, it will keep you from looking at the time, especially if you have some anxiety around timed tests. This way, you can see how you're progressing towards that limit. You don't want to be looking at it after each question, so just choose to look after each 10 questions. Once you have these things in place, I believe you'll do well on your Certified Cloud Practitioner exam. We've covered a lot in this book, and you are so close to being ready to take your Certified Cloud Practitioner exam. Go ahead and go through the Exam Prep book, and then if you do well in the Exam Prep book, then you'll be ready to sign up for your certification exam and go and take the exam. Best of luck on your next step to becoming an AWS Certified Cloud Practitioner.

BOOK 8

TERRAFORM FUNDAMENTALS

INFRASTRUCTURE DEPLOYMENT ACROSS MULTIPLE SERVICES

RICHIE MILLER

Introduction

Before we dive into the wonderful world that is Terraform, first I wanted to level set a little bit about infrastructure as code. Throughout the book, we are going to be using a lot of terminology that comes from the concepts in infrastructure as code, and in order to get the most out of this book, I first want you to have a good idea of what I mean when I say infrastructure as code. So let's get into what you need to know about infrastructure as code. First we are going to define infrastructure as code. Then we'll get into the core concepts of what infrastructure as code is, and then finally, we'll talk about the benefits of using infrastructure as code. But why would you even use Terraform? Well, Infrastructure as code is provisioning infrastructure through software to achieve consistent and predictable environments. There are a lot of important terms in this definition. A few that I want to call out is the fact that this is being done through software. It's not a manual process. And the goal is to achieve consistency. That means every time you use this software to deploy infrastructure, it does it in a consistent way and that the environment you get at the end is a predictable environment. It doesn't leave you guessing. It's going to look exactly like the configuration files say it should look. That's very important, especially when you have multiple environments that will be running the same version of an application. To achieve this goal, there are some core concepts we have to understand. The first, and this should be fairly obvious, but need to mention it, is infrastructure as code as defined in code. You're going to be creating files using some sort of software and coding mechanism to define your infrastructure, and whether that format is JSON, YAML, or HashiCorp configuration language, infrastructure as code is going to be defined in code. The next big concept is that you should be storing that code somewhere in source control. You are using code, after all. You might as well treat it like code. The source control management that most people are familiar with is Git and GitHub. GitHub uses Git source control to store repositories of code. The code is versioned, and multiple developers can work on it simultaneously. Your infrastructure that you've defined in code should be stored in a versioned source control repository. When it comes to the actual code itself, there are two different approaches to implementing infrastructure as code. There is declarative or imperative. Many people like to eat burgers. In fact my favorite day of the week is Burger Wednesday, and if I were to instruct software to make me a burger in an imperative way, I would do that by telling it what the exact steps are to make me a burger. First get me the ingredients for a burger. You need to get the shell, the beans, the cheese, the lettuce, and the salsa. And then I need to tell the software how to assemble those ingredients to make me a burger. I would tell the software to put the beans in the shell, put the cheese on the beans, put the lettuce on the cheese, and the salsa on the lettuce, because that is the proper order that one should assemble a burger in. Now you can see this is very procedural in nature. I'm telling the software exactly what to do. Declarative takes a slightly different approach. Let's say in a declarative world I also want software to make me a burger. That software is going to have a rudimentary idea of how to make food, like a cook. Just like I can tell a cook I want a burger with the following toppings, I can use a configuration language like HashiCorp Configuration Language to declare what I want. In this case, I'm telling it I want something that is of type food and of the burger, and I'm going to give it a name I can refer to it with, in this case And then within my configuration block, I'm going to tell it the ingredients I want in my burger; beans, cheese, lettuce, and ketchup. And that's all I have to tell the software. It already has a predefined routine for how to get ingredients and it has a predefined order in which to assemble those ingredients. If I want to change the defaults, I might put additional information in this configuration block, but the idea here is I'm declaring what I want. I want a bean burger with these ingredients, and then I'm leaving it up to the software to figure out exactly how to implement what I want. Terraform is an example of a declarative approach to deploying Infrastructure as Code. Another core concept is idempotence and consistency. You're probably already familiar with the idea of consistency. Each time you do something, the results should be the same. But idempotent is one of those words that gets thrown around, but you may not necessarily know what it means. Let's use another example to define it better. Let's say my syster, who also loves burgers, has asked me to make her a burger, and I do it. I say, "Here's your burger." In an idempotent world, if she asks me again to make her a burger, I will go, "Um, you already have a burger." I'm not going to go ahead and make another burger because I'm aware of her state and the fact she already has a burger. If she gives me the same instruction again, I'm not going to do anything because her instruction already matches the state of the world she wants. She has the burger. In a world, each time she told me to make her a burger, I would make and give her another burger. Terraform attempts to be idempotent in the sense that if you haven't changed anything about your code and you apply it again to the same environment, nothing will change in the environment because your defined code matches the reality of the infrastructure that exists. And that's what's meant by idempotent. The last concept to look at with Infrastructure as Code is are you pushing or pulling configurations to the target environment. So again, to give a fun example of what I mean by push or pull, in a scenario, once my syster has expressed her desire for a burger, I would simply go, take this burger, and push the burger over to her, and hopefully, if she's feeling polite, she'll say thanks. In a scenario, once she expresses that she wants the burger, she'll take the burger from me and I'll say, sure, here you go. In the world of Infrastructure as Code, Terraform is a model. The configuration that Terraform has is getting pushed to the target environment. An opposite example would be a situation where there's an agent running in the target environment and it pulls its configuration from a central source on a regular basis. All of this is great, but first let's talk about the benefits of using infrastructure as code. Why would you go through all the trouble of defining your infrastructure in code as opposed to just manually going out and deploying it? One, you've automated your deployment, which means that you don't have to go through the manual steps every time you need to build a new environment. That makes deployment faster, and faster is usually better in the world of technology. You've also created a repeatable process. Each time you need to build out or update the environment, you simply apply the configuration. Your new repeatable process can also be used to create multiple consistent environments. This is especially important if you want your dev, QA, staging, and production environments to all match. Your reputable process is defined in code, and code can be reused. Once you figured out how to properly deploy, say, a database server for a particular application, you can take the code for that database server deployment and reuse it in any other application that needs a similar database server backend. Having those reusable components will make your life a lot easier. It follows a principle that developers call Don't Repeat Yourself, or DRY programming. Once you write the code for a process, then you should make that process reusable so you don't have to repeat yourself. Lastly, one of the great things about defining your infrastructure as code is you've actually documented your architecture in the process of defining it within code. I've encountered situations where I thought I understood an architecture, but when I went to go define it as code, I realized there were components that I either didn't realize were part of the architecture, or I didn't understand how they actually worked. By defining my infrastructure deployment with code, I now had a deeper and better understanding of how my architecture was actually working. That's a huge benefit when everything is documented in code. Hopefully, in this first introductory chapter, I've allayed some of your concerns about Infrastructure as Code. It really isn't all that scary, and it does make your life easier. At the end of the day, manual processes are generally the enemy. Humans are fallible, we make mistakes, we forget things all the time, and if you are relying on manual processes to deploy environments consistently and repeatedly, at some point someone's going to miss a step; it's just the nature of human beings. Automating your infrastructure deployments makes a lot of sense. Lastly, when in doubt, go have a burger and think about it. I find that walking away for a moment on a particularly difficult project and having something delicious, like a burger, really helps my thinking process. Coming up in the next chapter, we're going to dive into Terraform proper by deploying our first Terraform configuration.

Chapter 1 Installing Terraform & Using the CLI

Now that we've laid a solid foundation for what Infrastructure as Code is, it's time to dive into Terraform, and I find the best way to do that is to go right in and get something deployed so you can start getting your head around the core concepts that make up Terraform. But what are we going to cover in this chapter? Well, first, we're going to talk about what Terraform even is. You probably already have some idea, but now we're going to get into the core components that make up Terraform, the basic workflow you'll use with Terraform to deploy infrastructure, and how to get Terraform installed on your workstation so you can follow along. Before we dive into actually deploying a Terraform configuration, I'd like to present you with a scenario that's going to help place some of the tasks and information you'll be learning into a context. As an IT practitioner I'm always eager to learn with any new tool, and I suspect you're going to feel the same way, and having a construct where you would be practicing these skills helps make sense of what might be just abstract concepts. Then finally, we're going to walk through a demonstration of deploying a basic configuration based on the requirements we define in the scenario. You can think of it as a Terraform: "Hello world", if you like, but I promise it's going to be a little more useful and practical than your usual hello world example. Let's explore what

Terraform is, the core components that are used with Terraform, and how to get it installed. Terraform is simply a tool to automate the deployment and management of infrastructure. The term infrastructure can be a bit nebulous, but I like to think of it as any layer of technology that a developer consumes without having to deploy and manage it. Networking, virtual machines, even containers all fall under the moniker of infrastructure. The core of Terraform is an project maintained by HashiCorp. There are paid versions of Terraform available as either Terraform Cloud or Terraform Enterprise. We're not going to cover those services in this book. We'll be sticking with the core version only. Terraform is also a vendor agnostic, meaning it doesn't prefer any particular cloud or service. You can use it against AWS, Azure, DigitalOcean, VMware, etc. Pretty much any infrastructure service you can think of probably works with Terraform. The core software for Terraform is a single binary compiled from Go. HashiCorp offers compiled versions for multiple operating systems, so chances are there is a Terraform binary that will work for you. Terraform configuration files use a declarative syntax rather than an imperative one. You are describing how you want the world to be, and Terraform is in charge of handling the heavy lifting. The actual configuration files are written in either HashiCorp Configuration Language, a derivative of JSON, or in JSON directly. Unless you are using another programming language to create Terraform configuration files, I'd recommend sticking to HCL. It's much more human readable and human writeable. Finally, Terraform uses a push style of deployment to create infrastructure. Terraform is going to reach out to the API for any given service and tell it what to create. There's no agent to install on a remote machine. That's a relief for those of us who have developed agent fatigue over the years.

One less thing to patch and maintain is a positive. There are four core components you should be aware of in Terraform. The first is the executable itself. This is the single binary file you invoke to run Terraform. It contains all the core Terraform functionality. The configuration that you're going to deploy will be contained in one or more Terraform files, which typically have the file extension .tf. When Terraform sees one or more Terraform files in a directory, it will take all of those files and stitch them together into a configuration. The next component is how Terraform talks to all the various services out there. The provider plugins are executables invoked by Terraform to interact with a service's APIs. For instance, AWS would be considered a provider, and if Terraform wants to talk to AWS and provision resources, it uses a provider plugin to do so. The most common plugins are hosted on the public Terraform Registry at registry.terraform.io. And then finally, once resources have been created, Terraform likes to keep track of what's going on, so it maintains state data which contains the current information about your deployment. It's a mapping of what you've defined in your configuration to what exists in your target environment. When you want to do an update of your environment, Terraform compares your updated configuration to what is in the state file, calculates the changes needed to make the two match, and then makes the changes and updates the state data. Installing Terraform is exceedingly simple. You simply download the executable compiled for your operating system, make sure that it's added to your path variable, and start using Terraform. Terraform is also available in common package managers like apt, yum, Homebrew, and Chocolatey. You could even grab it as a Docker container. In this demonstration, we're going to run through a couple quick items. First I'll show you where you can get Terraform installed. Once you've got it installed, we can try out some of the basic commands so you can learn the command structure favored by Terraform. If you would like to follow along, and I hope you do, you're going to need a system where you can install Terraform, a code editor to view the files. We are using Visual Studio Code. This is my preferred code editor of choice, but you use whatever works for you. I like this because I can see all of my files in the left pane, I can see the contents of those files in the center pane, and I can bring up a terminal from the bottom if I want to run commands all from within Visual Studio Code.

In the top folder base_web_app, we have the base configuration that we'll be working with, and it's called main.tf. In the commands folder, we have the commands that you can run for each of the chapters in this book. Then below the commands directory, we have a directory for the solution for each chapter beyond chapter three. Don't worry about what's in those right now. We'll discuss that more when we get to chapter four. For now, let's expand commands and open up m3_commands. Now if you want to play along, the first thing you're going to need to do is install Terraform if you don't already have it, and you can get it if you go to terraform.io/downloads. Let's take a look at that page right now. Here is the download page for Terraform.

If we scroll down a little bit, we can see all the different operating systems and the downloads for each of those operating systems.

This would be one way to install Terraform. If we scroll up a little bit, we can see that there are instructions for setting up the repository for APT or Yum if you wanted to install it that way. If you're using macOS, you can use Homebrew, and if you're using Windows, you could use Chocolatey to install Terraform. Let's head back to Visual Studio Code. Now I'm running Windows, so I used Chocolatey to install Terraform. Let's go ahead and bring up the terminal and see what version of Terraform I'm using.

First, I am going to run terraform version, and I am running Terraform version 1.0.8. That is the same as what we saw on the download page, so I'm running the current version on Windows AMD 64. And if I needed to upgrade my Terraform, I could run choco upgrade terraform and that would upgrade my version, but I'm on the current one. If we want to get some information about how to use the Terraform CLI, we could run either terraform or just type in terraform. The output from just running Terraform will list out the main commands you'll use, as well as other commands that are available.

If we look at the general usage for the CLI, we can see it's terraform, followed by any global options, then the subcommand that you want to run, followed by arguments. If we scroll down to the bottom again, we can see the global options include things like to specify what directory to run these commands from, can be used to get more information about Terraform or a specified subcommand, and is an alias for the version subcommand.

One other thing I want to point out is when you're specifying arguments with Terraform, even if the argument is multiple characters, you can still use a single dash instead of a double dash. Terraform will accept either, but the preferred syntax is a single dash. Now we have a base configuration in the base_web_app directory, but before we look at that base configuration, let's first get some context by introducing our scenario. To help put some context around what we're going to be doing in this book, I have a scenario involving the fictional company ACME. For our scenario, you have just started as an ITOps admin at ACME, a global risk assessment company. Congratulations! And welcome to the team! They're excited you're here and they already have a project lined up for you to work on. Your friend, Samantha the developer, has requested that you provision a development environment that's going to be part of a new application ACME is developing to turn their existing product into a SaaS product for their clients. The application is a basic web application right now. It's got a web frontend that will serve up content to potential customers. It's nothing super complicated. ACME has recently started using the public cloud for deploying its new applications, and you've been asked to spin up this environment in AWS, Amazon Web Services. You could, of course, simply log into the AWS console and set up the environment manually, but someone told you about this new software called Terraform, and this seems like an ideal project to take Terraform for a test run. In fact, Samantha has found a really basic Terraform deployment file she thinks you could get started with. The base configuration Samantha found includes the following. We're going to be deploying to the AWS region, and within that region we are creating a VPC with a single public subnet. And inside that subnet, we are creating a single EC2 instance that is running Nginx as a web server. We're also going to have to create routing resources and a security group to allow web traffic to reach that web server. You think this sounds like an excellent start. Before we dig into the configuration file, let's first talk a little bit about HCL syntax so you know what you're looking at. Before we look at the configuration, there are three Terraform object types you need to know about. They are providers, resources, and data sources. Provider blocks define information about a provider you want to use. For instance, we are going to be using the AWS provider and that provider wants to know what AWS account and region you're going to be using. Resources are things you want to create in a target environment and they are the bulk of what you'll be writing. Each resource is associated with a provider and will usually require some additional information for a configuration. A resource could be an EC2 instance, a virtual network, or even a database. Data sources are a way to query information from a provider. You aren't creating anything, you're simply asking for information you might want to use in your configuration. Just like resources, data sources are associated with a provider. A data source could be a current list of availability zones in a region, an AMI to use for an EC2 instance, or a list of templates on a vSphere cluster. But, what do these object configuration blocks look like? HashiCorp configuration language uses block syntax for everything in the file, it's a simplified version of JSON that is easier to read and it supports inline comments. Each block is going to start with the block type keyword that describes what type of object is being described in the block.

Next is going to be a series of labels that are dependent on what type of object we're working with. The last label in the series is usually the name label, which provides a way to refer back to the object in the rest of the configuration. Within that block, we are going to have one or more key value pairs that make use of available arguments for the object type. Each key will be a string and the value could be any of Terraform's different data types, which we'll get into in a later chapter. You can also have nested blocks inside of the main block. Nested blocks will start with the name of the nested block and curly braces. Inside the nested block will be more key value pairs. This might seem a little too abstract so let's see how the syntax would be applied to an EC2 instance in AWS. The object type we're describing here is a resource. We're creating an EC2 instance so we use the keyword resource, the type of resource is an EC2 instance, which based on the documentation for the AWS provider, uses the label aws_instance. Finally, the name label for our resource is web_server. This gives us a way to refer to it, especially if we've got multiple EC2 instances in our configuration.

Inside the block, we can specify a name for our EC2 instance. This is the name that we will see in the AWS console. Finally, we could use a nested block to specify an ebs_volume to attach to our EC2 instance, and inside that block, we could specify the size of the ebs_volume we want. If we have multiple ebs_volumes to attach, we can repeat the nested block multiple times. I've mentioned a few times the ability to refer to other objects inside of a Terraform configuration. HCL has a defined syntax for doing so. The general format to refer to a resource is the resource type, the name label, and then the attributes you want to reference from the resource. If you want the whole resource, you can skip the attribute. As an example, let's say we want to reference the name of our web server. The syntax would be the resource type aws_instance, the name label, web_server, and the attribute, name.

By doing this, we can get the value that is stored in the name attribute of our web server. Now that we have a little background about reading HCL syntax, let's take a look at that base configuration. Comments in HCL are supported by using the pound sign, and in this file, we've used comments to break up the file into providers, data, and resources.

Let's first look at the provider block. In our provider block, we're using the provider keyword to say this is a provider object, and then we're specifying the type of provider, in this case, AWS. This will let Terraform know we're using the AWS provider. Inside of the block, we have a set of pairs. We're telling the provider what AWS account we want to use and how we are going to access it by specifying our access key and secret key. And we're also telling it what region we want to use by specifying the argument region and setting it equal to If we scroll down into the data area, we have a single data source here.

We use the data keyword to specify that it is a data source. The data source type is aws_ssm_parameter. So, this is a service manager parameter, and we're giving it a name label of AMI. Within the configuration block, we have a single argument name, and we're setting it equal to a path to a parameter. This particular parameter grabs the latest Amazon Linux to AMI ID for the region we're currently using. We will make use of this value later when we create our AWS instance. Scrolling down a little bit more, we get into the resources portion of our configuration, and we start with networking.

We're going to create an AWS VPC, and we start the block by specifying the resource keyword followed by the resource type, in this case, aws_vpc, and then we're giving it a name label of vpc. Inside of the configuration block, we're setting the CIDR block that should be used by the VPC, and we're also enabling DNS hostnames. Looking at the next block, we are going to create an aws_internet_gateway, and we want to associate that internet gateway with the VPC we just created. To do that, inside the configuration block, we have the single argument vpc_id, and then we're using the reference syntax to reference the ID of our VPC. So that is aws_vpc.vpc, because that's the name label we assigned to our VPC resource, and then .id is the attribute that we want from that resource. You might be wondering, how do I know what arguments and attributes are available for a resource? And the short answer is you have to read the documentation for the provider and the resource. The longer answer is what we're going to explore in a future chapter as we add additional resources to this configuration. Scrolling down a little bit more, we can see that we are creating an aws_subnet with the name label subnet1.

We're assigning it a cidr_block, and we're referencing the same vpc_id that we just used for the internet gateway, and we're setting map_public_ip_on_launch to true, so when we spin up an EC2 instance in this subnet, it gets a public IP address. Scrolling down a little bit more, we are going to create an aws_route_table called rtb. We're going to associate it with our vpc, and here is our first nested block. In our nested block, we can specify a route to add to that route table. In this case, we're creating a default route and pointing it at our internet gateway. In this way, traffic can get out of our VPC through that internet gateway.

The last portion of the networking is associating our route table with our single subnet, and we will do that by creating an aws_route_table_association called Within that configuration block, we're going to specify the subnet_id of our single subnet, and the route_table_id of the route table we just created, and now there's an association between those two objects. Scrolling down a little bit more, we are going to create an aws_security_group that allows port 80 from anywhere to talk to our EC2 instance. We are associating this security group with our VPC, and we're creating a single ingress group using a nested block, and inside of that ingress nested block, we're setting the from_port and to_port to port 80 to allow port 80 in, we're setting the protocol to tcp, and the cidr_block is set to all 0's /0, which means allow traffic from anywhere on port 80. And then below, that we have an egress block, and this egress nested block allows outbound traffic to anywhere. Lastly, we have our EC2 instance. We're creating a resource of aws_instance type and naming it nginx1.

For the AMI ID, we are now going to be referencing our data source, and we can see the syntax for that is a little different than regular resources. We first have to specify it is a data source by saying data dot the type of data source dot the name label and then the attribute that we want from that data source, in this case, value. So this will return the AMI ID for Amazon Linux 2 in the region we're currently working in. If you're curious about what the term is, that is a function, and we're going to cover functions a little bit later, so don't worry about that for now. Instance_type sets the instance type to t2.micro. We are trying to keep this thing as small as possible to stay on the free tier. The subnet_id will reference the single subnet that we have created, and then the argument vpc_security_group_ids, you see that's plural, that's expecting a list of security group IDs. We only have a single security group ID to give it, but we still need to put it in a list. Lists are enclosed in square brackets, and then the elements in the list are separated by commas. We only have a single element for the list, which is the security group we created to allow port 80. And then lastly, we are sending some user data to our instance, and this is simply a script that will run when the instance starts up for the first time. In the script, we are installing nginx and starting it up, we're deleting the default index.html file, and replacing it with something else. If you're not familiar with the EOF syntax that you're seeing right there, that is a way of specifying a block of text that should not be interpreted in any way; it should just be passed directly to the argument as is. And this is an easy way for you to specify a script without having Terraform try to interpolate it. The syntax is simply two of the less signs followed by a keyword, in this case, EOF, the text you want, and then closing it with that same keyword, EOF again. That is everything that's in the configuration. Now we need to deploy our configuration. But how do we go about doing that?

Chapter 2 Terraform Workflow & Deployment

Terraform has a basic workflow that allows you to provision, update, and remove infrastructure. Let's dig into that workflow now. If you'll recall from earlier, Terraform makes use of provider plugins to interact with services like AWS. Before it can use those plugins, it needs to get them. This is done as part of the initialization process, and the command to do so is terraform init. Terraform init looks for configuration files inside of the current working directory and examines them to see if they need any provider plugins. If they do, it will try and download those plugins from the public Terraform Registry, unless you specify an alternate location. Terraform will also need to store state data about your configuration somewhere. Part of the initialization process is getting a state data back end ready. If you don't specify a back end, Terraform will create a state data file in the current working directory. Once initialization is complete, Terraform is ready to deploy some infrastructure. The next step in the workflow is to plan out your deployment with terraform plan. In this case, Terraform will take a look at your current configuration, the contents of your state data, determine the differences between the two, and make a plan to update your target environment to match the desired configuration. Terraform will print out the plan for you to look at, and you can verify the changes Terraform wants to make. You don't have to run a terraform plan, but it is pretty useful to know what Terraform is planning to do before it does it. You can save the plan changes to a file and then feed that back to Terraform in the next step. It's now time to actually make changes in the target environment, and you do that by running terraform apply. Assuming you ran terraform plan and saved the changes to a file, Terraform will simply execute those changes using the provider plugins. The resources will be created or modified in the target environment, and then the state data will be updated to reflect the changes. If we run terraform plan or apply again without making any changes, Terraform will tell us no changes are necessary since the configuration and the state data match. There is one more command I want to bring up, which might seem a little strange, and that's terraform destroy. If you are done with the environment, the command terraform destroy will do exactly that, destroy everything in the target environment based off of what is in state data. This is a dangerous command, and Terraform will ask you if you're sure. We're going to use this command in the book to save money when we're done with the chapter, but in the real world, please take care. With the basic workflow fresh in our brains, let's get our base configuration deployed. We'll start by initializing the configuration, then we'll plan our deployment, and finally, we'll apply the plan to create resources. If you're following along, and again, I hope that you are, you're going to need an AWS account and AWS access keys. I'd recommend creating a separate AWS account to use for this book so it doesn't conflict with anything else, but that's entirely up to you. Quick disclaimer. Some of the resources deployed in AWS may cost you money. I tried to use the smallest instances possible, but there is a chance you will be charged some small amount of money for what you're provisioning in AWS, so consider yourself suitably warned. Let's get our basic configuration deployed. Now before we do that, let's make a copy of our base configuration and edit that copy. So first I'm going to open up the terminal. I'll go ahead and do that now. Before we run through the actual workflow, there is one tiny change we need to make in the main.tf file. Go ahead and open that now. You can see the AWS access_key and secret_key have placeholders in them. I'm going to update those values with a valid AWS access key and secret key.

While I am filling this out, I want to provide a quick disclaimer. You should never hardcode your access key and secret key into a Terraform configuration. We're doing it right now because we haven't yet learned a better way of doing it, but rest assured, in the next chapter we are going to remove this from the configuration and never do it again. This is purely for demonstration purposes. In fact, I've already invalidated this access_key and secret_key by the time you read this book. With that being said, I'll go ahead and save this file, and now we can run through the initialization process, and I'll do that by running terraform init.

It's going to go ahead and initialize the backend it will use for state data and download any provider plugins that it needs for our configuration. And lastly, it will create a special lock file called .terraform.lock.hcl.

So if we scroll up a little bit, we can see where it initializes the backend, initializes the provider plugins by downloading the latest plugin from the public Terraform registry, and creating that lock file as the last thing. And if we look over in our directory, we can see there's that .terraform.lock file, and there's also a new directory called .terraform, and inside that, if we expand it, that is where it downloads the provider executable that will be used to talk to AWS. Now that our Terraform configuration is initialized, we can go ahead and run terraform plan. So I will run terraform plan, and I'm going to add a new argument here, This will write the plan out to a file, and I'm specifying the file as m3.tfplan. So I'll go ahead and run that now. And as part of the plan, it is going to reach out to AWS and determine what it needs to create to match our AWS environment to what's in the configuration. And that ran pretty quickly.

We can see that it's saying in the plan there are seven resources to add. And if we go ahead and expand this all the way up, we can scroll up and review what's in the plan. So let's scroll up to the top, and we can see it starts with the instance that's going to be created. You should note, anything with a green plus sign indicates that the resource or attribute is going to be created. So we can see there is our AWS instance. If we scroll down a bit more, we can see the internet_gateway, the route_table, etc.

So it's going to create seven resources in total. It's not going to change any, and it's not going to destroy any. If we want to apply our plan, we can simply run terraform apply and feed it our file, m3.tfplan. So I'll go ahead and do that now. And if we had run terraform apply without specifying a plan file, it would first print a plan of what it's going to do and then ask for confirmation of the changes that it's going to make. Because we supplied a tfplan file, it doesn't have to confirm those changes because it assumes we've already reviewed that plan.

So this could take a few minutes, so I'll go ahead and jump to where the deployment has completed successfully. Our deployment has completed successfully. We can see seven resources were successfully added.

Let's go over to the AWS console and get the public IP address of our EC2 instance and validate that the web page is available. Here we are in the EC2 console. I'll go ahead and refresh our view of instances.

There is our instance that has been created. I can click on that and see that it does have a public DNS. So we can go ahead and grab that address, and I'll open up a new browser tab, and we can go to that address. Going back to Visual Studio Code, since this was simply a demonstration environment, the last thing we can do is destroy the environment so it doesn't cost us any money, and we'll do that by running terraform destroy.

Once you run terraform destroy, it will plan out the changes it needs to make to destroy everything that's in your environment, and then it will ask for confirmation. In the read out, the red dash indicates that something is going to be destroyed or removed, and now it's asking if we're sure we really want to do this. And we do, so I will type in yes, and now it will go through the process of removing all of those resources, and we no longer have to worry about paying for them. I encourage you to do this when you finish any exercise and you know you won't be coming back to the environment for a while.

It's very simple to stand it back up by simply running terraform plan and apply again when you're ready to work with the environment. In summary, Terraform is a tool used to automate infrastructure, which is way more fun than manually deploying stuff. Terraform itself is a single binary available for just about any operating system out there. The configurations Terraform uses are written in either HCL or JSON, although HCL is way more popular. Finally, the basic workflow for Terraform is initialization, plan, and then apply. The base configuration we just deployed is pretty simple and it leaves lots of room for improvement. In the next chapter, we're going to take a look at how we can use variables and outputs to improve our configuration.

All programming languages have a way to submit information into the software and retrieve output. Terraform is no different. In this chapter, we are going to explore how to use input variables, local values, and outputs to improve our Terraform code making it more dynamic and reusable. The base configuration we deployed to AWS had all of its values hardcoded and provided us with no output. It's time to change that. We'll first start with learning how to supply input values to Terraform for use in a configuration, and then we'll learn how we can construct internal values inside the configuration for reuse. We'd also like to get some information out of our configuration once it's deployed and that is done through outputs. Finally, we are going to make a bunch of changes to our configuration, but what if we get something wrong? It sure would be nice to validate our config before we try and deploy it, and we'll see how Terraform has some tools to help. Terraform can accept values as input, transform values inside a configuration, and return values as output. With that context, let's explore how to work with data inside Terraform. There are three different concepts to consider when working with data in a Terraform configuration. The first is called input variables, or just variables for short. Input variables are used to pass information to a Terraform configuration. The variables are defined inside the configuration, and the values are supplied when Terraform is executed. Local values, sometimes just called locals, are computed values inside the configuration that can be referenced throughout the config. In other programming languages, these would usually be called variables. The values for locals are not submitted directly from an external input, but they can be computed based on input variables and internal references. Data is returned by Terraform with output values. The outputs are defined in the configuration, and the value of each output will depend on what it references inside the configuration. Just like locals, the output value can be constructed from one or more elements. Since everything starts with inputs, let's take a closer look at input variables. Variables are defined inside of a block just like everything else in Terraform.

A variable block starts with the variable keyword followed by a single label, and that is the name label. All the other properties of the variable are defined inside the block and all of those properties are optional. You can have a variable with no arguments and that's acceptable, although it's not really preferred. Let's take a look at the optional arguments inside the variable block. The type argument defines the data type associated with your variable and it provides a certain level of error checking. If you say the variables should be a number and someone submits a string, Terraform will throw an error. Now you might be wondering what data types are available to me. Don't worry, we'll cover that in the next section. The description argument helps provide some context for the user when they get an error and it will also be useful when we package configurations up in chapters, but we'll cover that later in the book. The default argument allows you to set a default value for the variable. If no value is submitted for the variable, Terraform will use this default value. If you don't set a default value and none is submitted when the configuration is invoked, Terraform will prompt you at the command line to supply a value. The last argument I will cover is the sensitive argument. It accepts a Boolean value of true or false. If it's set to true, Terraform will not show the value of the variable in its logs or the terminal output. This argument is useful when you have to submit potentially sensitive values like a password or an API key and you don't want them showing up in clear text in your logs or terminal output. Let's take a look at a few examples of actual variables and how to refer to their value inside a configuration.

The first example shows a variable with the name label, billing_tag. No arguments are provided and none are needed. This is a quick and dirty way of adding a variable to a configuration. Since no default value is specified, you'll need to provide one at execution time. Our second variable has the name label aws_region, and this time, we have some arguments. We're going to set the type to string since the value will be one of the AWS regions and those are strings. We've got a helpful description here and we're setting a default value of So if no value is specified at execution time, Terraform will use Finally. This is not a sensitive value so we've set it to false. We don't actually have to set sensitive to false as it is false by default. To refer to the value stored in the variable, we simply use the var identifier dot the name_label. For instance, to refer to the value stored in our aws_region variable, the syntax would be var.aws_region and you would get back the string stored in the variable.

Speaking of strings and data types, let's talk about the different data types that exist in Terraform.

Chapter 3 Terraform Data Types

We can group the data types supported by Terraform into three categories. The most basic are the primitive data types. These are string, number, and Boolean. A string is a sequence of Unicode characters, a number can be an integer or a decimal, and Boolean is either true or false. The next category is collection data types, and they represent a grouping of the primitive data types. A list is an ordered group of elements, a set is an unordered group of elements, and a map is a group of pairs. In each case, the values stored in any of these collection data types must be of the same data type. The last group is structural data types, and they're very similar to collection data types, except they allow you to mix the data types stored in each grouping. Aside from that difference, tuples are functionally equivalent to lists and objects are basically equivalent to maps. It's useful to be aware of structural data types, but chances are you're not going to use them for basic configurations. They're more of an advanced topic. Let's take a look at some examples of the collection data types to help clarify things. Here's a couple examples of lists. Notice that each element of the list is of the same data type, all numbers in the first list and all strings in the second.

The third list mixes data types, which would be invalid for a list, but valid for a tuple. Our Map example has three pairs. The keys are going to be strings, and the values must all be the same data type. In this case, they are all of type string. You can create more complex structures using the object data type, but that's really beyond the scope of this book. If you want to use a collection for a variable, how do you construct it, and how do you reference the values inside? Let's take a look.

Let's say we'd like to have a variable with a list of AWS regions. The type argument takes the form of the collection type we'd like to use and what data type will be stored in it. In this case, we have a list collection type that will be storing string values. For our default value, we have provided a list of four regions, each as strings. Lists are an ordered data type. We can refer to an element by number, starting with 0. If we want the first element in our list, which is our syntax would be var.aws_regions and a 0 for the first element enclosed in square brackets.

We can get the whole list by only specifying the name label and skipping the square brackets. What if we want a map holding AWS instance sizes? The type argument is basically the same. We want to have a map with strings as the value held in the map. For the default, we can define the keys as small, medium, or large and associate an EC2 instance size with each key. If we want to refer to the value stored in one of those keys, there are actually two ways of doing so.

The first is var... The second is var., followed by the key_name in quotes inside of square brackets. We can retrieve the value stored in the small key by writing var.aws_instance_sizes.small or var.aws_instance_sizes, then small in quotes and brackets.

Armed with our new knowledge of using variables, let's check in with the folks at ACME and see how we can improve our base configuration.

Samantha is excited that you got the environment up so quickly, but the folks over in ops have some requests about how the environment is deployed. Let's review the current architecture and the requests for improvement. The current deployment architecture is a single EC2 instance in a public subnet inside a VPC in the region of AWS. The ops team doesn't want you to change the architecture yet, but they do want you to make some code improvements. Jack is from the ops team, and he has a little experience with Terraform. He's come up with a list of possible improvements for your code. For starters, those AWS credentials can't live in the code file. It's just not safe to throw those things around. Jack would like you to find a better way, preferably a way that doesn't store the credentials in a file at all. Speaking of values, Jack would like you to use variables wherever possible so the configuration can be more dynamic and possibly reusable. ACME is also instituting default tags for their AWS resources, and Jack would like an easy way to apply default tags to all the resources in the config without doing a lot of find and replace. Finally, it would be nice to know the public DNS hostname of the EC2 instance without having to go to the AWS console. You tell Jack, no problem. We'll start by adding some variables. Now it's time to start adding some variables to our configuration. Before you get started, if you destroyed the environment from the previous chapter, go ahead and recreate it now because we're going to be making changes to the configuration and then seeing how those changes apply to what's been deployed already. With that in mind, let's take a look at our current configuration by opening up the main.tf file in. This is our current main.tf file. We wanted to find some variables for this configuration and the first thing we can do is create a file called variables.tf in the same directory. Remember, Terraform will put together any .tf files it finds in the same directory. By keeping the variables in a separate file, we can easily look between the main.tf file and the variables.tf file as we add new variables. I'm going to go ahead and hide the file tree and we'll split the view between variables.tf and the main.tf file. Now we can add variables in the variables.tf file and make the changes in the main file. Let's first start by getting rid of those AWS access key and secret key values. We'll start by creating a new variable, and remember, this starts with the variable keyword. We'll give it the name label, aws_access_key. This is going to be of type string. We can add a description of aws_access_key. We're not going to set a default for this variable because the whole point is getting the access key out of the configuration, but we should set one more argument and that's setting sensitive to true. After all, we don't want this access key to be exposed in the logs or in the terminal output. Now that we have our first variable, let's go ahead and replace the hardcoded string with a reference to this variable. We'll do that by removing the current value and now we'll add a reference to our variable. Remember that goes var. the name of the variable, which is aws_access_key.

If you're using VS Code or something that has similar plugins. it might even helpfully finish that for you. Now let's go ahead and do the same with the AWS secret key. So I'm going to copy this variable and paste it down below, and I'm going to change the name from access key to secret key, I'll change the description, and now we'll replace the secret key value with a reference to our variable. Our access key and our secret key are no longer hardcoded into our configuration. Let's also take this opportunity to add a variable for our region in case we wanted to deploy to a different AWS region. We'll start with the variable keyword and we'll set this variable to aws_region. Just like the keys, this is going to be of type string. We'll set a description of AWS Region to use for resources, and let's set a default value for this variable of Now this is not a sensitive value, so we won't set the sensitive argument since it defaults to false. Let's go ahead and replace the region with our variable. We'll set it to var.aws_region. Our provider is now using all variables for its values.

Let's scroll down a little bit more and see where else we could use variables. In our networking configuration, we can see the CIDR block has a hardcoded value, enable DNS hostnames has a hardcoded value, and in the subnet, the CIDR block and the map_public_ip_on_launch both have hardcoded values. Scrolling down a bit more, in the security groups, you could potentially set variables for the port numbers if you would like to, and scrolling down beyond that, there is an instance type which is hardcoded for the AWS instance, that's another good place where we could add a variable.

Try to add all these variables to your configuration. When you're done, you can go ahead and look in the file layout for the M for solution and that will show you the variables that I added to the configuration and how I reference them in the main.tf file. Let's see how you did.

Looking in the variables file, we've got our access key, secret key, and region that we created. Scrolling down some more, we've got enable_dns_hostnames, the vpc_cidr_block, the vpc_subnet1_cidr_block, and map_public_ip_on_launch. And scrolling down a bit more, we have the instance type and there should be references in the main.tf file for each of these variables.

The next thing that we're going to talk about is local values and how we can use those to add those common tags that Jack was asking for.

Local values are values computed inside of the configuration. You can't submit values directly to them, unlike input variables. The syntax for locals starts with the keyword locals, and that's it for labels on the block. The rest of the information goes inside of the block. Inside the block are key value pairs. The value can be any supported Terraform data type, string, list, object, the supported data types are the limit. Here's an example of a locals block. The first key value pair defines a local with the name instance_prefix and the value. The next key is common_tags, and its value is a map defining some common tags. You can refer to other values inside of your configuration for the values on local. For instance, the project key is being assigned the value in the variable project. This seems pretty useful for our ACME requirements. You can specify the locals block multiple times in your configuration if you want to, but the name of each key must be unique within the configuration since that is how you reference a locals value.

To refer to the value stored in a local, the syntax starts with the local keyword. Note that local is singular, not plural, followed by dot and the name_label. To get the value in the instance prefix, the syntax would be local.instance_prefix. If we've got a collection data type in our local, the same syntax we saw from the variable example applies. We could get the value stored in company by writing local.common_tags.company. If we'd rather get the entire map, we only need to write local.common_tags. Let's head back to our configuration and update it based on this new information. ACME is looking to add three common tags to start to all resources we've defined in our configuration. We can define the common tags in a locals value and then use that value throughout our configuration. Let's start by creating a new file called locals.tf. In our locals.tf, we will start by defining a locals block. Within our locals block, let's go ahead and define a map of common tags. We'll start with common_tags =, and then we'll use the curly braces to specify a map data type, and then we'll add our pairs.

The three values they want to start are company, project, and billing_code. But where are we going to get this information from?

Let's use variables to get this information. Let's open up our variables file and add three variables for company, project, and billing_code. We'll scroll down to the bottom of our variables file, and we'll go ahead and add those three variables. Here's a chance for you to take the wheel again. Try to add those three variables. Again, they are company, project, and billing_code. I have added those variables to my configuration, and as you can see, I set a default of ACME for the company and specified no default value for the project or the billing_code. Now let's go ahead and add these variable values to our locals. And we can do the same thing we did before, which is hide the File Explorer. We'll split variables out to the right side so it's easier to work with and showed the locals on the left side so we remember exactly what we're working with. We'll start by adding company. Next, we'll add project. And for this one, Jack has requested that the project be the company name dash the project name for the value.

But how do we go about referencing a variable inside a larger string? We're going to use interpolation syntax, which sounds real fancy, but it's actually quite easy. We start by adding quotes to indicate that this is going to be a string, and then we need to reference our variable. We start with a dollar sign followed by curly braces. This let's Terraform know that we're going to be referencing a value from a variable or some other object within our configuration. Now we can add the variable reference, which will be var.company. Next, we're going to add a dash after the curly braces and another reference to the project value stored in the variable project. Now we've created a string from our two variables that is of the form

Lastly, let's add our billing_code. And We have successfully created our common_tags local value. The next thing to do is add this common_tags value to our main.tf file for each AWS resource that supports tags. Let's go ahead and add the first one together. So I'll switch over to the main.tf file, and let's go down to our first resource, which is the aws_vpc. Within the configuration block, I'll go ahead and add tags, and I'm going to set tags equal to local.common_tags. Now this map will be submitted to the tags argument, which expects a data type of map, and those tags will be applied to the VPC. My challenge to you is to add this tags argument to all the other resources in our configuration that support a tags argument. The only resource that does not is the aws_route_table_association, so you can add the tags argument to all the other resources in the configuration. I have successfully added the tags argument to all of my resources, and hopefully you have too. The last thing we're going to do is add an output so that we know what the public DNS hostname is for our EC2 instance.

Chapter 4 Output Values Syntax & Architecture Updates

Output values are how we get information out of Terraform. Outputs are printed out to the terminal window at the end of a configuration run. It also exposes values when a configuration is placed inside a chapter, something we'll cover later in the book. The syntax for an output starts with the output keyword followed by the name_label for the output. Inside the configuration block, the only required argument is the value of the output.

Just like the value of a local, the value of an output can be any supported Terraform data type. You can return a simple string or a complex object. Optional arguments include the description, which is only seen when looking at the code for a configuration, so it's not all that useful. The sensitive argument will set an output to sensitive, meaning that the actual value will not be printed in the terminal. This is useful when you want to pass a value from one chapter to another and avoid having it printed in clear text in the logs or the terminal. That will make more sense when we get to those chapters.

Here's our example of an output with the name_label public_dns_hostname. We're setting the value equal to the public_dns attribute of our web_server EC2 instance using the same reference syntax we saw in the previous chapter. We've included a description of the output, too, for our own personal reference. Sensitive is not set, so it defaults to a value of false. That's good, because we want this value printed to the terminal window so we can use it. Let's head back to the configuration and add an output. Just as we did with the locals and with the variables, let's go ahead and add a file for the outputs. Within the outputs, we are going to define a single output. We'll start with the output keyword, and we'll set the name to aws_instance_public_dns. For the value, we're going to reference our EC2 instance. So let's go ahead and go back into mode here and bring up our main.tf file. We'll scroll down to our instance definition, and for that we'll go ahead and copy the type, paste it in, we'll add the name_label, nginx1, and we'll add the property of public_dns.

You can see that because we've initialized our configuration previously and I have the Terraform extension installed in VS Code, it knows all the attributes that are available for the aws_instance resource type, and so I don't have to remember all of them or look them up. I'll go ahead and save the file, and now we've added our single output that we wanted. Now you might be wondering, how do I know that I got all of this configuration syntax correct? Well, your code editor should help you a little bit by highlighting improper syntax, but Terraform can also help you with the validate command. Let's learn more about that now. Before we try to apply our update, it would be nice to know if the configuration is syntactically correct. Our linter does its best to help, but Terraform can also lend a helping hand. Terraform has a command called validate that will help you make sure your configuration is correct. Before you run the command, you'll need to run Terraform init. That's because it's checking the syntax and arguments of the resources in the providers, and it needs the provider plugins to do so. When you run validate, it will check the syntax and the logic of your configuration to make sure everything looks good. If it finds any errors, it will print out the error and the line where it found the issue. Sometimes it will even make a suggestion. Terraform validate does not check the current state of your deployment; it's just verifying the contents of your configuration. It also carries no guarantee that your updated deployment will be successful. Your syntax and logic might be correct, but the deployment could still fail for any number of reasons; insufficient capacity, incorrect instance size, overlapping address space. Validate does what it can, but it can't do everything. Why don't we try to use validate against our updated configuration? The configuration we have now should pass validation, but we want to see what validation does, so let's go ahead and add a couple errors to our main.tf file and then see how Terraform catches them. Let's start by scrolling down, and we'll put some square brackets around enable_dns_hostnames as if it were a list and not a Boolean value. That should definitely throw an error. We can also use a variable reference that does not exist, so let's go ahead and delete block off var.vpc_cidr, and that should also throw an error.

Now that we've made those changes, we'll go ahead and save the file and we'll bring up the terminal to run terraform validate. I'm already in the working directory, and I've run terraform init, so all I should need to do is run terraform validate.

And as you can see, Terraform has come back with an error. Let's go ahead and expand the window so we can see the full error.

Here it's telling us that we have a reference to an undeclared input variable, vpc_cidr.

Well, we knew that, so let's go ahead and fix that problem. We'll scroll down here and we'll add _block back to our variable name label. I'll go ahead and save the file, and now we can run terraform validate again. Now we can see that Terraform says we have an incorrect attribute value type. The value attribute for enable_dns_hostnames should be a Boolean, and we've supplied a list, so let's go ahead and take those square brackets off and save our file once more, and we'll go ahead and run terraform validate a third time

Our configuration is valid. Now you may find that you have other issues in your configuration if you've been working on your own, so go ahead and remediate those now. The next step in our process is to supply values for the variables we've defined in our configuration. But how do we go about doing that? Let's find out. When it comes to setting the value for a variable, there are at least six ways of doing so. That's a lot. The easiest way to set a value is to set the value with the default argument. We've already seen that in our configuration. You can also set the variable at the command line when executing a terraform run. You can use the flag followed by the name of the variable and the value you want to set it equal to. You can repeat this flag for each variable you'd like to set. You can also have all your variable values in a file and submit that file with the argument. Inside the file will be each variable name label as a key, followed by an equal sign and the value for the variable. There are two other ways to submit values from a file. If there is a file in the same directory as the configuration named terraform.tfvars or terraform.tfvars.json, which needs to be properly formatted JSON, Terraform will use the values it finds in that file. Additionally, if there is a file in the same directory as the configuration ending in .auto.tfvars or .auto.tfvars.json, Terraform will use those values as well. The final option is to use environment variables. Terraform will look for any environment variables that start with TF_VAR_ followed by the variable name. If you don't submit a value for a variable in any of these ways, Terraform will prompt you for a value at runtime. I know that's a lot of options. And what if you set the same variable in multiple ways, what's going to happen? There is an order of precedence. Here's what it looks like, but I have to admit, I always have to look it up. Terraform evaluates each of these options, with the last one winning. If you find that your variable has the wrong value being set, this might be the culprit. Now that we know how to set values for our variables, let's go make use of our updated and validated deployment. Reviewing the variables in our configuration, there are a few that don't have a default set. We're going to have to set the aws_access_key and aws_secret_key. Scrolling down to the bottom of our variables, we can see that the project and the billing code also need a value supplied. All the other variables have a default set and we can go ahead and keep using that default. Let's take a look at the potential syntax if we wanted to submit values for all of these variables at the command line. I'll go ahead and expand commands over here and we'll take a look in m4 commands. In our m4 commands, we have already initialized and validated our Terraform configuration.

Now we can pass our variables at the command line if we'd like to and the syntax for that is the name label of the variable and then another equals and the value you want to set that variable to. Now, as you can see, this command can get very long. There has got to be a simpler way to do this and we know there is. Let's go ahead and create a file called terraform.tfvars and populate it with some of our variables and values. I'll go ahead and create a new file called terraform.tfvars in the same directory as our configuration. And let's go ahead and split screen things again so we can add the values and see what the values should be. We'll start with the first variable, billing_code. I'll go ahead and grab that value and paste it over in our file and I'm going to set it equal to the value described in the command.

Now let's set the next variable, which is project, go ahead and paste that over and set that one equal to And the next two variables are AWS access key and secret key. We don't want those in a file, instead, we can store them inside an environment variable. Let's go ahead and save our terraform.tfvars file, we'll go ahead and close that out, and then looking at our m4 commands, we can see we are going to export 2 environment variables. If you're working in Linux or Mac OS, you can use the export command. If you're using PowerShell, you can use the $env: and then the name of the environment variable. Once again, the environment variable is going to be TF_VAR_ the name of the variable that you want to set a value to. So the first one is going to be aws_access_key. I'm working in PowerShell so I'm going to go ahead and paste in my access key and secret key so I can set them as environment variables. , I've updated the two commands with my AWS access key and secret key. Let's go ahead and copy both those commands and paste them in the terminal down below. Now I have those environment variables set, they're not stored in a file with our configuration and they won't be shown in the terminal output or in any logging we enable for Terraform. Let's go ahead and clear the terminal and now we can go ahead and run Terraform plan with our output file and we don't have to worry about including any variable values in the command because we have defined them in files and environment variables. We'll go ahead and run Terraform plan now, and because the only change we made is to create variables, local values, and outputs, and add some tags to our resources, the only real change on the AWS side is to add those tags. So let's go ahead and expand the terminal up and take a look at what's changing in our configuration.

Now we can see that there are six changes with nothing to add and nothing to destroy. If we look at what's being changed about our VPC, the yellow tilde means that something is being updated or changed, and the green plus sign means a value is being added so we can see the tags that are being added to our VPC. Let's go ahead and run the terraform_apply to update the tags on all of our resources. This should go very quickly because we're simply modifying the tags for our resources, it's not having to create or destroy anything.

Now we can see that the output we get at the end is, in fact, the public DNS of our AWS EC2 instance. We can use that as opposed to going into the console. At this point, we have accomplished all the goals that Jack set out for us. So far, we made our configuration a bit more viable. We used input variables so we can supply the proper values at runtime and get those credentials out of here. We also saw the multitude of ways to set values for our variables. It can get confusing quickly, so I recommend keeping it simple. We also managed to get some information out of our configuration with output values. And finally, we validated our configuration before trying to deploy it to catch any syntax or logic errors in our code. With a viable configuration in place, it's time to turn our attention back to the architecture. Our current design isn't exactly resilient, so we're going to add another instance and load balancing to the deployment. We've updated our configuration to include variables, locals, and outputs. Now it's time to update the architecture of our deployment to include resiliency by adding new resources. Our existing architecture is a single EC2 instance running in a single subnet on AWS. If something were to happen to that instance or the availability zone associated with the subnet, our application would go down. That's probably okay for development, but not if this application is going to make its way to production. We will start our process by updating the architecture design, determining what new resources we need to add. Once we know what resources we need to add, we can consult the official HashiCorp docs to see what arguments are required for each resource. Armed with the knowledge gleaned from the docs, we will set about updating the configuration with new resources and data sources and apply the updated config to our existing deployment. We will also take a moment to talk a bit more about the magical state data Terraform uses to map a config to a deployment. What's in that data, and how should you interact with it? First, we will start by planning and infrastructure update with our friend Jack. Adding variables, locals, and outputs to the configuration was a great start, but now it's time to add some resources to improve the architecture. Let's see what ACME has in mind. Our current architecture is using a single subnet in a single availability zone with a single EC2 instance. That's a lot of single points of failure. Jack from the Ops Team has some suggestions to improve the reliability of the deployment. First, we'll start by adding a second availability zone in AWS. If you're not familiar, each availability zone in an AWS region is a separate physical data center, and each subnet is associated with one, and only one, availability zone. Adding a second subnet in a separate availability zone will protect from a zone failure. We also only have a single EC2 instance. Jack suggests that we add a second instance in case the first instance fails. Of course, adding a second instance doesn't magically fix things; we need a way to make both instances accessible, and we'll do that through a load balancer. Lastly, Jack wants to make sure we maintain the readability of the code. He noted that we made a separate file for variables, locals, and outputs, and he thinks it would be a good idea to split the resources out as well. Perhaps we could make one for base networking, another for instances, and one for the load balancer. Not only does that make it easier to read the code, it might make some files reusable in other configurations. What does this updated architecture look like? Here's our current architecture with the single subnet and EC2 instance. We didn't specify an availability zone for our subnets, so AWS picked one for us. And here's the new architecture. We now have two subnets that should be in separate availability zones, meaning we're going to need to specify an availability zone for each one. We now have two instances that will be identical in nature except for the subnet they attach to, and we are adding an application load balancer, which will serve as the public endpoint for our application and direct traffic to our instances. If you're not overly familiar with AWS, or even if you are, you might be wondering exactly which resources you'll need to add to the Terraform configuration to create this architecture. Well, I'm not going to leave you hanging to figure that out on your own. This is a Terraform book, after all, and we're not here to learn the intricacies of AWS. Here are the new data sources and resources we'll need to add to our configuration. With our two subnets, we now care which availability zone each one is in. We could add a variable to specify the availability zone for each subnet, but there's a better and more dynamic way.

We can add a data source that gets the list of availability zones in the current region and use that list in our subnet settings. For the load balancer component, there are actually several resources that need to be added, and I have to admit, it's not immediately obvious what they are. The first is the aws_lb resource itself, which will be the application load balancer. Next will be the aws_lb_target_group, which defines a group that the application load balancer can target when a request comes in. To service incoming requests, we need an aws_lb_listener that listens on port 80 for inbound requests. And lastly, we need to associate our target group with our EC2 instances. The aws_lb_target_group_attachment resource takes care of that. Well, that's all the new resource types. We're also going to add an additional subnet, EC2 instance, and a security group for the load balancer. Why don't we head over to the configuration and add some placeholders for the new resources?

Chapter 5 How to Add New Resources

Let's get started by updating the file structure a little bit for our webapp. I'll go ahead and expand the folder out now, and we'll start by creating some new files. Let's go ahead and create one for the load balancer, and we will create one for the instances, and we can rename our main.tf network because the only thing that's going to be left in it once we move stuff around is networking components. now that we've created our files, let's go ahead and move the instance configuration into its own file.

Scrolling down to the bottom, I'll go ahead and grab this entire body of text that defines the instance, cut it, go into the instance file, and paste it in there. Our instances will now have their own file to reside in. I'll go ahead and save that. For the load balancer, we haven't actually created any of the resources yet, so let's instead add some placeholders for the resources we know we need to create.

I often add comments that let me know what resources I need to create before I actually create them, so I'll go ahead and add some comments to this file now. I'll add the aws_lb, the aws_lb_target_group, the aws_lb_listener, and the aws_lb_target_group_attachment. Now I know what needs to go in this file, but, of course, I still have to create the resources and understand all the arguments that go into each resource.

How am I going to get that information? The answer is to read the documentation. There is no shame in going to the docs to try and figure out how to configure a resource or work with some Terraform syntax. Whenever I am writing a new Terraform configuration, I usually have the code editor open in one monitor and multiple docs tabs open in another monitor. So why don't we go check out those docs?

The documentation for the providers and the resources within them can be found at registry.terraform.io. The provider that we are interested in is the AWS provider which we can find by simply clicking on Browse Providers and it gives us a list of the most popular providers on the first page. We want to work with the AWS provider, and lucky us, it's right there. Let's go ahead and click on that. The front page will tell you some information about the current AWS provider and you'll note there is a tab for documentation. Let's go ahead and click on that to go to the documentation. The main page of the documentation explains a little bit about how to use the AWS provider. It gives some examples of how to instantiate it, as well as how to authenticate to the provider. We'll cover that more in a later chapter.

Right now, we are mostly concerned with adding our data source and our resources. The easiest way to find those is usually to search through the filter box so I'll go ahead and start typing in availability zones. After typing just a portion of the phrase, we can see under Data Sources in the matching results, we have aws_availability_zone and aws_availability_zones, that's the one I'm interested in so I'll go ahead and click on it, and this is the documentation for the data source. It gives us a description of what the data source does and then it provides us a simple example for usage. This is great. This explains how to use this data source.

First, we declare it using the type of data source and giving it a name label, and then there is some optional arguments that go in the configuration block. When we want to use this data source. It gives us an example of how to use it with a subnet, which is pretty convenient because that's exactly how we want to use this data source.

If we want to inspect some more information about this data source, we can go to the arguments reference. I'll go ahead and click on the link. This takes us down to the Arguments Reference portion of the page.

These are the arguments you can supply inside of the configuration block. We might want to specify the state argument that filters the list of availability zones that are returned by the data source. We could say just give me the available ones. Below the argument reference is the attribute reference. These are the attributes that are exposed for the data source. The one that we're most interested in is the names of the availability zones because that's what we're going to use to configure our subnet, and based off the information here, it is a list that is returned of the availability zone names, which means we can reference each zone by its element within the list.

If we go back up to the example usage, that makes sense with what it's showing us here that we use the names attribute along with an element of that list to retrieve a single availability zone name.

Now that we know this, we can go ahead and just copy a portion of this example and put it right into our configuration file. There is no need to recreate the wheel here. With that text copied, let's go over to our configuration, and since this is going to be part of our network configuration, let's go to the network file, we'll scroll all the way up to the top, and we'll add another data source in the data area. We've added our new data source for the availability zones.

Now we can make use of them in our subnet. The next thing to do is add the additional subnet, security group, and EC2 instance.

Now that we've successfully added our availability zone's data source, it's time to update our subnet's security group and EC2 instances. Let's first start by updating our existing subnet to use an availability zone. I'll go ahead and scroll down in the file to our subnet. There it is. And we're going to add a new argument here for the availability zone. The argument is going to be availability zone, and I will set that equal to the data.aws_availability_zones.available data source. And the attribute we want, remember, is names. So we'll do .names. And for this first subnet, let's take the first element from the list. So we'll do square brackets and 0, since lists are We’ve added our availability zone.

Before we create a second subnet, the CIDR block is defined with the variable vpc_subnet1_cidr_block. Wouldn't it be easier if we had a variable that had all of the subnet blocks defined in it? Why don't we go ahead and create that first. So, we'll go ahead and open up variables, and we'll scroll up to that variable definition for the subnet1_cidr_block, and instead we'll change that to vpc_subnets_cidr_block. Instead of type string, we'll make it a list of strings. And for the default, we can update this to a list of strings. I'll add the square bracket, so it's a list, and I will add a second element to the list of 10.0.1.0/24 for our second subnet and close the square bracket, and we can update our description to CIDR Block for Subnets in VPC. Let’s go back to our subnet configuration in the network.tf file.

And we'll update this variable to vpc_subnets_cidr_block, and we'll select the first element out of the list. You'll see the reason I did this in a moment as we add the second subnet for our configuration. Before I add the second subnet to our configuration, my challenge to you is to go ahead and try to add these resources on your own.

You're going to need to add a second subnet, a route table association, and a second EC2 instance in the instances file. Go ahead and try to do that now. If you run into trouble, you can always check the solution, and come back in a moment to see my updated solution. we're back. Let's go ahead and see what I did in my solution.

For the second subnet, I made a copy of the existing subnet resource, and I changed the name label to subnet2. For the cidr_block, I changed the element to 2 to reference the second element in the list, and for availability_zones, I changed that to a 1 as well to reference the second availability zone. By setting it up in this way, we could add a third or fourth subnet and just make sure that we update the subnet's cidr_block appropriately. Scrolling down into the routing, let's take a look at those route table associations.

Once again, I simply copied the existing route_table_association_resource, changed the name label to subnet2, and changed the subnet_id reference to subnet2. Both of these subnets are going to use the same route table. Moving over to the instances file, for the second instance, once again, I made a copy of the existing aws_instance resource, I changed the name label to nginx2, and I changed the subnet to subnet2. To differentiate the two web pages, I changed the echo command for the first one to say Team Server 1, and if we scroll down, the second one is now Team Server 2. The next thing we need to do is create an additional security group for our load balancer so it allows port 80 traffic from anywhere.

So let's go back to the network file. And if we scroll down here, we already have a security group for our instances that allows

HTTP access from anywhere. Let's make a copy of the security group. I'll go ahead and update the name label of this new security group. We'll call it alb_sg, and we'll update the name to nginx_alb_sg.

We're gonna be using the same VPC ID, and the ingress block is already correct. We want to allow port 80 access from anywhere. We do want to make a change to our existing security group for the instances. Now that we have a load balancer in front of them, they should only accept traffic from addresses that are within the VPC. So we can scroll up and change this cidr_block reference to var.vpc_cidr_block. Now, it will only allow traffic from addresses that are in the vpc_cidr_block. Let's go ahead and save the network file. The next thing to do is add our load balancer resources. But before we do that, we need to know how to construct each of those resources. Let's head back to the docs.

Chapter 6 How to Add Load Balancer Resources

Back in the docs, let's try to search for aws_lb, and, that returns over 1000 results. That's not going to be very helpful. Let's go ahead and clear the filter, and I happen to know from experience that this is under Elastic Load Balancing v2. So let's scroll down to that. There's Elastic Load Balancing v2. That includes the application load bouncer and the network load balancer. We'll go ahead and expand that out, and we can see it split up into Resources and Data Sources. That means if we had an existing AWS load bouncer and we wanted to use it as a data source, we could. But in our case we want to create an AWS load balancer resource, so let's go ahead and click on that resource. And just like the data source, this gives us an example usage for both the application load bouncer and the network load balancer.

The example here is very close to what we actually want, so let's go ahead and copy this and place it in our configuration and then make a few simple updates. I'll go ahead and copy the text and go over to the configuration, and we'll paste it directly in the file. Now let's update the name label and the name of our load balancer. We'll set the name label to nginx, and we'll set the name to web_alb.

Internal should be set to false because we're creating a public load balancer. The load_balancer_type should be application. That's the type that we want. For the list of security_groups, we should update it to the security group that we just created. So let's go into split screen mode and bring up our network configuration on the left and scroll down to our new load balancer security group and grab that name label, and we'll paste it over here. And now we need to update the subnets argument to the list of subnets that we're going to be using.

So we'll go ahead and delete this subnets argument and add square brackets to indicate a list, and now we can add our two subnets. So I'll go ahead and scroll up to our subnet definitions, and I'll grab the resource type, followed by the name label, and the attribute that we want is id, so I'll do .id, then I'll add a comma, and we'll copy this text, paste it, and update it to subnet2. We've included both of the subnets we want to associate with our load balancer. The next property is enable_deletion_protection. We're going to set that to false because we want Terraform to be able to delete this load balancer when we're done with it. For now, we're not going to configure the access logs, so I'll go ahead and delete this block. And lastly, we'll update our tags to reference our local value, local.common_tags. That's everything for the load balancer. My challenge to you now is to create the rest of the resources using the documentation. If you get stuck, you can reference the solution, and when we come back, you can see how I updated my solution. Here's my updated solution, and why don't I get out of split screen mode here so we can better see what's in the configuration.

I added the aws_lb_target_group, specifying the correct port of 80, the protocol of http, and the correct VPC ID.

Scrolling down a bit more, I created the aws_lb_listener, which needs to reference the ARN of the load balancer that we've created, the proper port and protocol, and we're going to set the default_action of type forward to send traffic to a target group, and then we'll specify the target_group_arn of the target group we just created. Scrolling down a little bit more, we have two target group attachments, one for each EC2 instance.

In there I've specified the target_group_arn, the target_id is going to be the idea of each EC2 instance, and the ports is going to be ports 80. That's everything that goes into this configuration. Now before we apply our updated configuration to our deployment, let's talk a little bit about Terraform state.

So far, we've talked about state data as the way that Terraform maps what's in your configuration to the actual deployment on target environment, but what's in that state data and how can you interact with it when necessary? Terraform state data is stored in a JSON format. You should not try to alter this JSON data by hand. Bad things can and will happen. We'll be looking at how you can use Terraform commands to work with it shortly. State data stores important information about your deployment, including mappings of resources from the identifier in the configuration to a unique identifier in your target environment.

Each time Terraform performs an operation like a plan or apply, it refreshes the state data by querying the deployment environment. The state data also contains metadata about the version of Terraform used, the version of the state data format, and the serial number of the current state data. When Terraform is executing an operation that potentially alters state data, it tries to place a lock on the data so no other instance of Terraform can make changes. Imagine if the state data was in a shared location and two admins tried to make conflicting changes at the same time, that's no good. Locking helps to prevent that situation from arising. Speaking of the state data location, you can store the state locally on your file system, which is what Terraform does by default, or you can specify a remote backend for the state data, that could be an AWS S3 bucket, an Azure storage account, an NFS share, or HashiCorp's Terraform Cloud service. A remote backend for state is useful when working on a team to collaborate and to move state data off your local machine for safety's sake. We're not going to cover remote state data in this book. Another feature supported by Terraform state is workspaces which enable you to use the same configuration to spin up multiple instances of a deployment, each with their own separate state data. We will cover workspaces in more detail in a future chapter. What does state data look like? Here is a rough sketch of what is in the state data.

We've got the current version of the state data format and the version of Terraform that was last used on the data. This is important because older versions of Terraform might not be compatible with the latest format of the state data. Terraform will let you know if that's a problem. The serial number is incremented each time the state data is updated.

The lineage is a unique ID associated with each instance of state data and prevents Terraform from updating the wrong state data associated with a config. The Output section contains the outputs we saw printed in the terminal window in the last chapter, and resources is a list of resource mapping and attributes.

Let's jump over to our configuration and look at the actual state file. Back in our configuration, we can see the state file is terraform.tfstate. Let's go ahead and open it. Starting from the top, the version is version 4, the terraform version used is 1.0.8, and the serial number is 32, that is incremented each time the state is changed. And then we have our lineage, which is the unique ID for this particular state data. Below that, we have the outputs. We have a single output defined and the value is stored in the state. Below that, we have a list of resources. The data source is considered a resource, in this case, and it has information about that data source. If we scroll down some more, we have our first actual resource, which is our AWS instance. It has the name label we've associated with this specific resource, the provider that was used to create it, and then information about that resource including its attributes. That is what you'll find if you look inside the state file, which leads me to another very important point. You do not want to make any changes to this file directly and honestly you probably shouldn't even open the file, in general. Let's look at some commands you can use to work with state data.

Chapter 7 Terraform State Commands & Providers

There are a subset of commands with Terraform specifically to deal with state. We won't cover all of the commands, but I did want to touch on some of the most commonly used ones. To see all the resources being managed by Terraform, you can run terraform state list. From that list, you might want to know more about a specific resource. You can find out more by running terraform state show and the resource address, which is the resource type and the name label. You can move an item to a different address in the same state file.

This can be useful for renaming resources or moving them into chapters. The syntax for that command is terraform state mv for move, followed by the source address and the destination address for the resource. Lastly, if you need to purge something from the state, you can do so by using terraform st rm and the address of the resource. You might want to remove a resource from Terraform management without destroying it. You could remove the resource block from the configuration and then remove the entry from state. Otherwise, the next time you ran terraform apply, it would attempt to destroy the deployed resource in the target environment, which leads me to my next and maybe most important point. The first rule of Terraform is to make all changes with Terraform. Don't try to manually edit state data, and don't make changes to managed resources with the cloud console or the CLI. Make changes in the configuration, and then apply those changes through Terraform; otherwise, Terraform will either undo your changes at best or get hopelessly confused at worst. With that advice in mind, let's head back to our configuration and get our updates deployed. Back in our configuration, let's go ahead and expand the commands directory and open up the m5_commands. First, let's try out a couple of those Terraform state commands. I'll go ahead and bring up the terminal down below, and let's first run terraform state list.

If we want to see the properties of a specific resource, we can use the address that's shown on the screen. Let's run terraform state show and then look at the information for the aws_instance.nginx1.

This now shows us all of the available information about nginx1. We can see all of the information that's , and it's a fairly significant amount. Next up, let's validate our configuration and run the update against our existing deployment. Before we try to run a plan for our configuration, let's first run terraform validate and make sure we don't have any mistakes in our configuration.

Now our configuration is valid. Let's go ahead and run terraform plan. I'll go back to my m5_commands. If you haven't already, you're going to need to export the environment variable TF_VAR_aws_access_key and secret_key. I've already done that, so I can scroll down to the terraform plan command. I'll go ahead and run that now, and we'll save the plan to m5.tfplan. we're going to be making some significant changes here.

We have 12 things to add, 1 to change, and 3 to destroy. I'll go ahead and expand the terminal so we can see what is going on in the plan. Scrolling up a bit, let's see what's being replaced. Well, subnet1 needs to be replaced because it's changing availability zones, so it's letting us know it's going to delete that subnet and recreate it.

Scrolling up a bit more, our security group for the NGINX instances is going to be updated because we're changing the ingress block.

Our route table association for subnet1 has to be replaced because the subnet is being replaced. And scrolling all the way up from there to our first nginx1 instance, that also has to be replaced in part because we're changing the subnet ID, but also because we changed the user data that's associated with the instance.

We are fine with all of these changes, so let's go ahead and run terraform apply to apply the changes to our target environment. This is going to take a little while because it's going to create that subnet and the load balancer and the EC2 instance. Generally speaking, the load balancer is what actually takes the longest to create, so I'll jump to when the deployment has completed successfully. Our deployment is successful, but our output is still giving us the AWS instance public DNS.

That's something we should probably change. Let's go ahead and open up outputs, and instead of the instance, we want to get the public DNS of our load balancer. Let's go back in split screen mode and we'll hide the terminal for a moment, and let's go to the load balancer. We'll grab the address for the load balancer, which is aws_lb.nginx.dns_name for the attribute. And we'll go ahead and save that output.

We'll bring the terminal back up. We can run a terraform validate to make sure our change didn't mess anything up.

And now, because we haven't made any changes to the resources, we can just run terraform apply directly, and we can do that by simply doing terraform apply and adding the flag so it doesn't prompt us to approve any changes. I would only recommend doing this when you're absolutely certain that no changes will be made that might be destructive to your target environment. Since we're only changing an output, it's not going to make any changes to our target environment, and now we have the public DNS for our load balancer.

We know that both instances are responding on the load balancer and our deployment is successful. In summary, we've added new resources to our configuration to make it more resilient and production ready. We also took a look at the docs for the AWS provider to help us with the arguments and syntax for all our new resources. Just remember, there is no shame in reading the docs or copying examples. State data is Terraform's map from the config to the deployment and it is very important. A corrupt state is a dire circumstance to find yourself in. Treat the state data with respect and all will be well. So far, we've only worked with the AWS provider. Now it's time to see how you add an additional provider, how to work with provider versions, and we're going to learn what provisioners are and why you probably shouldn't use them.

One of the key strengths of Terraform is its vendor agnostic and pluggable approach. Anyone can develop a provider plugin for Terraform and you can use more than one provider in a configuration. We will see how to add and configure providers in this chapter. Our configuration will continue to evolve in this chapter based on requests from both the development and ops teams at ACME. One of their requests will require adding a new provider to the configuration, but before we do that, we'll learn a bit more about how to add and configure a provider. We are also going to dig into the dependency graph that Terraform creates when planning a deployment and learn when it might be necessary to specify an explicit dependency in your configuration. Finally, we are going to examine the options that exist for post deployment configuration of resources. Once our EC2 instances are deployed, how do we perform the initial configuration and manage them going forward? First, it's time to check in with Samantha and Jack. Our deployment is shaping up nicely. We've got a application running up in AWS. But, of course, nothing in

IT is ever really done. The Dev and Ops teams both have new requests. Our friend Samantha has a couple requests from the development side of the house. For starters, she would like to give us the website files and have them dynamically uploaded to the web servers at startup. She would also like to get access to the request logging from the load balancer for analysis and debugging. Jack has a few things he'd like to see us implement as well. As Terraform is adopted by ACME, he wants to make sure we are all using the same major version of Terraform and the provider plugins. He would also like the Terraform files to be formatted consistently to help with sharing across the teams. Why don't we start with Samantha's two requests by updating our architecture to support her needs. It sounds like she needs an S3 bucket for logging, and we can also put her website files there to be picked up by the EC2 instances when they start up. In our architecture, we will add an S3 bucket and upload the website content to it. Then we will assign the EC2 instances a profile that has access to copy information from the S3 bucket. The load balancer configuration supports logging to an S3 bucket, so we can use the same S3 bucket to write those access logs out. S3 buckets need to have globally unique names, and that's something we can generate with the random provider for Terraform. With that in mind, let's see how we can add a provider and meet some of Jack's requests. Provider plugins are Terraform's superpower. We talked about providers in an earlier chapter, but now that you've had a chance to use them, it's time to go into greater detail. As we have already seen, Terraform provider plugins are available in the public registry at registry.terraform.io, but you can also get provider plugins from other public registries, privately hosted registries or even your local file system. We aren't going to get into that use case in this book, but it's useful to know. There are three types of provider plugins available on the Terraform hosted registry, Official, Verified, and Community. Official providers are written and maintained by HashiCorp. Verified plugins are written and maintained by a organization that has been verified by HashiCorp and is part of the HashiCorp technology partner program. Community provider plugins are written and maintained by individuals in the community, and have not gone through the verification process. There's nothing inherently wrong or bad about using community providers, but you should be aware of their providence and probable level of support. One thing all the providers have in common is that they are open source and written in Go. If you have the inclination to inspect or contribute to a provider, the code is readily available for you to do so. Providers themselves are a collection of data sources and resources, as we have seen when reviewing the documentation. Providers are versioned using semantic version numbering. You can control what version of a provider plugin you use in your configurations so you can avoid a situation where a provider is updated and it breaks something in your deployment. Within your configuration, you can invoke multiple instances of the same provider and refer to each instance by an alias. For example, an instance of the AWS provider is limited to one region and account. If you wanted to use more than one region in a configuration, you could do so with multiple instances of the AWS provider and aliases. Let's take a look at how you can specify the source of a provider plugin and the desired version. This will help us fulfill Jack's request.

We have already seen how to create a provider block in our configuration, and you might assume that is where you would specify more information about the provider, like its version and source. That assumption would be well founded, but unfortunately incorrect. Provider information is defined in the terraform configuration block using a nested block called required_providers. We haven't seen the terraform block before.

It is used to configure general settings about a Terraform configuration, including the version of Terraform required, settings for the state data, required provider plugins, provider metadata, and experimental language features. For the purpose of this book, we will focus on using the terraform block to define our required_providers. Each key in the required_providers block will be the name reference for a provider plugin. The convention is to use the standard provider_name, unless you're going to have multiple instances of a plugin from different sources. That's an advanced topic and one you're unlikely to encounter when you're first getting started with Terraform, so don't worry about it right now. The value for the provider key will be a map defining the source of the plugin and the version of the plugin to use. By default, Terraform assumes you are getting your plugin from the Terraform Registry, and it provides a simple shorthand for the address value. There is an expanded form of the address for alternate locations. The version of the provider can use several different arguments, including setting it equal to a version, a range of versions, or using a special sequence of a tilde followed by a symbol. That last one is not immediately intuitive, so let's look at an example. We've been using the AWS provider from the

Terraform Registry. If we wanted to add it to our required_providers block, we would set the key to aws, as that is the name of the provider in the documentation. For the source, we can use the shorthand of hashicorp/aws. Since we're not giving Terraform a full address of the plugin, it will try and find the plugin on the Terraform Registry. Under our version, let's say we wanted to stay on version 3 of the plugin, but we don't care about the minor version. Using the tilde and symbol tells Terraform to find the latest plugin that is of the form 3.x. If we wanted to stay on the minor version of 3.7, we could update the expression to 3.7.0, and that would keep us on the latest 3.7 release.

Most of the breaking changes in a provider will come from a major version release, so staying on the same major release of a plugin should keep things stable, although your mileage may vary depending on the provider. Once we've defined our required_providers, we can reference them in a provider block. The block starts with the provider keyword, and then the name of the provider used in the required provider block.

If you are going to create more than one instance of the provider, you can add an alias argument inside the block, providing a string for that instance of the provider. And then you can provide any additional arguments that are specific to that provider. Assuming we've gone with the convention and used aws as the provider_name for the AWS provider, our provider block stays the same, with aws as the name label.

If we want to create an additional instance of the AWS provider for a different region, we could give an alias of west in the block. To use the aliased instance of the provider with a resource or data source, we would add the provider argument to the configuration block. The value would be the provider_name.thealias, which would be aws.west, in this case. If no provider argument is specified, Terraform will use a default provider instance with no alias set. Armed with all this new knowledge, let's head over to our configuration and add a required_providers block for the AWS and random providers. Before we add the terraform and required_providers block to our configuration, let's take a look at the docs.

We can browse to the AWS provider by clicking on Browse Providers and going to AWS. And let's go into the Documentation tab, and in the beginning of the AWS Provider it provides some Example Usage, and there is the terraform block and required_providers block. That's exactly what we want, so let's go ahead and copy that text from the example. So I'll copy that text, and let's go over to our configuration. In our configuration, let's go ahead and expand the directory, and we're going to create a new file called providers.tf, and in the providers.tf file we'll go ahead and paste that text. now we are sourcing our provider from the public registry, and we're setting the version to stay on major version 3. Right now 3.63 is the latest version, but when they come out with 3.7, we'll automatically upgrade to 3.7. If version 4 comes out, we will not automatically upgrade to that, and that is what we want.

Now that we have our AWS provider added, there's something else I want to point out about the AWS provider documentation. Something that we glossed over when we were looking at the documentation earlier is the Authentication section for the AWS provider. This provides information about how to authenticate using the provider. We've been using static credentials up until now, defined in variables. There are many other options for authentication. We have environment variables; a shared credentials or configuration file which is generated by the AWS CLI; and also if you're running Terraform on AWS, you can leverage CodeBuild, ECS, EKS or the EC2 Instance Metadata Service.

Instead of using credentials with variables, an approach that I've often seen is using environment variables. Let's scroll down to the Environment Variables area. We can provide our credentials with two environment variables as opposed to defining variables inside of the Terraform configuration.

By doing that, we will prevent someone from accidentally credentials in a terraform.tf vars file and checking that into source control. That's bad. We don't want that to happen. So let's go back to our configuration, remove those variables, and from here on out we can use the environment variables instead. back in the configuration, let's go ahead and open up our variables file, and we are going to delete the aws_access_key and aws_secret_key from our list of variables. We'll go ahead and do that now; we’re going to keep the aws_region, because we need to define that for our provider. I'll go ahead and save that file, and now let's go to where our provider is defined. Right now that's sitting in the network.tf file. That's probably not the best place for it, so let's go ahead and remove that from the network.tf file, and instead we'll add it to the providers.tf file. So I'll go ahead and add it in there, and now I can remove the access_key and secret_key arguments, since we'll be supplying those values through the environment variables defined in the documentation.

Now let's head back to the documentation and we will walk through adding the random provider to our configuration.

Chapter 8 How to Add Random Provider

We are going to use the random provider and the random_integer resource in the provider to help generate a unique ID for our S3 bucket. Remember S3 bucket names need to be globally unique.

The easiest way to do that is add some sort of unique ID to the end of the name for your S3 bucket. Let's go ahead and search for the random provider, and it comes up as the first result. We can go ahead and click on that, and if we want to see how to use the provider, we can actually click on the USE PROVIDER That will give us an example of how to add it to our existing terraform block or add a new terraform block if we don't have it. My challenge to you now is to add this additional provider to the required_providers block in the configuration and set the version to stay on major version 3, but accept updates in the minor version. Go ahead and try to add the provider on your own. Let's see how you did.

Going back to the configuration, I have added the random provider to my required_providers block, and I've set the version to ~> 3.0, which will keep us on the major 3.0 version. The resource we want to add from random is the random integer, so let's go back to the documentation and see how we add that. Back on the website, let's click on the Documentation area, and before we expand the resources, one thing I want to point out about the random provider is that it doesn't have any configuration options for the provider block, which means you don't actually need to include a provider block in your configuration since there's nothing to configure.

With that in mind, let's expand the resources here and take a look at the random_integer. Here's the random_integer with an example usage.

What I would like for you to do now is add the random_integer resource to the locals.tf file in our configuration, and set the minimum to 10000 and the maximum to 99999. You don't have to include the keepers argument, just a min and a max and use the name_label of rand. Go ahead and we'll come back and see how you did. Let's go to my updated configuration and see how I added the random_integer resource to my locals.tf file. Here is my locals.tf file, and you can see I've added the resource random_integer with the name_label rand, setting a minimum value of 10000 and a maximum value of 99999.

When we use this as part of our S3 bucket naming, we will have a random integer that will be appended to the name of the S3 bucket. With those components in place, let's look at the resources we need to add for our S3 bucket and to allow access from the EC2 instances. With our provider situation figured out, we can turn to Samantha's request to add an S3 bucket for website content and logging. Before we jump back into the configuration, let's figure out what resources we will need to create. We are going to create an S3 bucket and place objects in that bucket for the website. To accomplish that goal, we are going to use the aws_s3_bucket resource and the aws_s3_bucket_object resource.

Our EC2 instances will need access to the S3 bucket, but we don't want to make the bucket public for everyone, instead we can create some IAM resources to help us grant access to the EC2 instances. We'll create a role using the aws_iam_role and grant that role permissions to the bucket with an aws_iam_role_policy. Then we can assign the role to the EC2 instances by creating an aws_iam_instance_profile and then adding an entry to our aws_instance block to use that instance profile. We also need to provide the load balancer access to the S3 bucket, and we can do that through a bucket policy that refers to a data source of aws_elb_service_account. That will give us the service principal account for the elastic load balancer in the region that we're currently working in, and we can grant that access to the S3 bucket. With all that context in mind, let's head over to our configuration and add some placeholders for each resource. Back in our configuration, let's add a file for the S3 configuration and we'll call it s3.tf, and within that file let's add placeholders for all the different resources that we need to create. So I'll add in the comments for the file, aws_s3_bucket, aws_s3_bucket_object, aws_iam_role, aws_iam_role_policy, and aws_iam_instance_profile.

In addition to these resources, let's go ahead and open up the loadbalancer file, and let's add a placeholder to the beginning of this loadbalancer file.

, now we've got all of our placeholders. For the website files that we'll be uploading to our S3 bucket, those are located in the root of the exercise files. So we can scroll down to the bottom here. Those are the website files. We've got an index.html file and an image file of the ACME logo. We'll go ahead and copy this website directory and paste it in our directory. We now have our website files. The other thing we need to do is create a name for our S3 bucket, and we can do that by defining a new local value. So let's open our locals.tf file, and we'll add another value . That will append a unique ID to our S3 bucket name. Going back to the s3.tf file, my challenge to you, if you really do want a challenge, is to try to add all of the necessary resources to this file. I will say that many of these resources require you to write an IAM policy or a bucket policy, and that's very difficult, so maybe skip those portions of the resource configuration and try to do the rest, along with the loadbalancer data source. And then you can take a look in the M6 solution directory to see how the IAM policies are configured for each of the resources that uses it. This is going to be a real challenge, so if you want to do it, go ahead and we'll come back to see my solution. Let's see how you did. And first, we'll start with the S3 bucket. The bucket argument is going to be the name of the bucket, and we're going to set that to our local value.

The acl Is going to be private because we don't want this to be a public bucket. We'll set the force_destroy = true, which allows Terraform to destroy the bucket. Now below that is the policy, and we're going to embed the entire policy here, which is in JSON. In order to do that, we are going to use the heredoc syntax that we saw when we configured the user data for our instances. And this will replicate this text exactly except for the interpolation that we've added for the values from Terraform. So let's take a look at what's in this policy. And don't worry about being a bucket policy or an IAM policy expert. This is a Terraform book, after all, and not one on AWS. So I'm just going to point out the relevant things for you if you're ever writing one of these policies. In our statement for the bucket policy, we want to allow the load balancer and the delivery logs service access to this S3 bucket. We do that by adding an effect of allow, and we're going to reference a principle here from our Elastic Load Balancer service account data source.

So if we go over to the loadbalancer file, this is the data source that you'll need to reference the service account used by Elastic Load Balancers in your region. Going back to the S3 file, for the action, we're giving it s3:PutObject, and for the resource, we're giving it the bucket name and then the path, This gives the Elastic Load Balancer permission to write data to that path in our S3 bucket. We're also going to give that same permission to the service delivery.logs.amazonaws.com. And we're going to give that service an additional permission of s3:GetBucketAcl.

This entire policy is available on the AWS docs, so don't worry about trying to memorize it or anything. You can always go back to the documentation and find it. Scrolling down to the next resource, we have our two bucket objects, which are the website components we want to upload to the S3 bucket. We first have the bucket argument that references the S3 bucket, and then we have a key, which is the destination on the S3 bucket where it should create that object, and the source is where to get that object from.

We're getting it from the website directory that we copied into our configuration directory earlier. Scrolling down a bit more, we get into the IAM portion of things by first creating the IAM role that's going to be used by our instances.

The name is allow_nginx_s3. And for the assume role policy, this allows EC2 instances to assume this role. That's all that does. Scrolling down a little bit more, we get into the role policy, and this is the policy that actually grants permissions to access the S3 bucket. We're naming it allow_s3_all, and we're assigning it to the role that we just created by name, and then we define the policy with the same heredoc syntax.

We're giving it the Action s3:*, which means you can do anything in the S3 bucket, and we're assigning it the resource of the bucket name and any paths along that bucket name. That's the policy that's assigned to the IAM role we just created.

Scrolling down a bit more, we get into the instance profile. The instance profile is what's going to be assigned to the EC2 instance. We're giving it the name nginx_profile. We're associating it with the role that we created earlier, and we're giving it the common tags like we have with everything else in this configuration that supports common tags. The next thing we need to do is update our instance and our load balancer to take advantage of this S3 bucket. But before we do that, we need to talk about dependencies.

When Terraform is trying to make the deployment match your configuration, it has to run through a planning process. Terraform goes through this process when you run a plan, apply or destroy. As part of the planning process, it needs to figure out the order in which to create, update or delete objects. To calculate a plan of action, Terraform will first refresh and inspect the state data. Then it will parse the configuration and build a dependency graph based on the data sources and resources defined in the code. Comparing the graph to the state data, Terraform will make a list of additions, updates, and deletions. Ideally, Terraform would like to make the updates in parallel, so it tries to figure out which changes are dependent on other changes. Changes that are not dependent on other changes can be made at the same time, while changes that have a dependency will have to be done serially. How does Terraform figure out the order in which changes need to happen? References. Let's look at an example that we have in our configuration right now. In our current configuration, we are creating a VPC, a subnet, and an EC2 instance. The VPC doesn't refer to any other resources in the configuration, so Terraform can create it immediately. If we look at the arguments in the aws_subnet resource, we have a reference to the vpc.id. The reference creates a dependency on the VPC resource. Terraform will wait until the VPC is created, and then use the ID to create the subnet. Our aws_instance configuration has a reference to the aws_subnet ID. That creates a dependency for the EC2 instance, so Terraform will wait for the VPC and then the subnet to be created before it tries to create the EC2 instance. Terraform can infer the dependency tree for this configuration implicitly. It doesn't need you to tell it that the subnet is dependent on the VPC. Sometimes a dependency is and you must explicitly tell Terraform about it. We actually have that situation in our configuration right now, you just don't know it yet. It's one of those things that you only figure out once it breaks, so let me explain. In our configuration, we are creating an aws_iam_role. Both the instance_profile and the role_policy directly reference the iam_role, so Terraform will wait until the role exists to create those two resources. To assign proper permissions to our EC2 instances, we have to add the instance_profile to our aws_instance configuration, which creates a dependency between the instance_profile and the instances. However, in order for the EC2 instances to actually access the S3 bucket using the iam_role_policy, it also needs to be created. If the EC2 instance starts up before the iam_role_policy is ready, access to the bucket will be denied, and that's bad. The solution is to add a depends_on argument to the aws_instance resource that references the iam_role_policy. With that explicit dependency, Terraform will wait until the iam_role_policy creation is complete before moving on to creating the aws_instances. Generally speaking, Terraform is pretty good at detecting implicit dependencies. The depends_on argument should be used sparingly, and only when an explicit dependency is required. Armed with that knowledge, let's go back to the configuration and update the load balancer and EC2 instances. My challenge to you is to go into the load balancer and add the access log configuration and go into the EC2 instance, add the instance profile, and add that depends_on argument. The depends_on argument is expecting a list of references to other resources within the configuration. So go ahead and try that now, and when we come back, we can take a look at my updated configuration. Let's see how you did. Let's first take a look at the load balancer configuration. In the load balancer configuration, you can see there's now an access_logs configuration block inside of the resource, and in there, we are referencing the bucket that we created.

We're going to use the actual reference and not the bucket name, so we create a dependency between the load balancer and the bucket. The prefix is going to be and enabled is set to true. We want to write logs there. now let's take a look at the instances. In our instances, I have added an argument, iam_instance_profile, and it's set to the name attribute of the profile that we created.

For depends_on, we're giving it a list of resources it should be dependent on, so we need the square brackets, and then in there, I am referencing the iam_role_policy.allow_s3_all. So the instance will wait until that role policy is created before spinning up the instance.

If we scroll down a little bit, we also have to update our second AWS instance, so don't forget to do that. We want them to be configured the same. The last thing we need to do is update our user data script. But before we do that, let's discuss post deployment configuration options. After a resource is created, sometimes you need to perform configuration. It could be loading an application onto a virtual machine, configuring a database cluster or generating files on an NFS share based on resources that are created. If you want to stay in the Terraform ecosystem, there are many providers and resources that can help you with activities. If you want to create a file, there's a file resource. If you need to configure a MySQL database cluster, there is a MySQL provider. Using native Terraform resources will often be the answer. Another option specific to servers is to pass data as a startup script to the server operating system. All the major cloud providers offer a way to pass a script, although the name of the argument changes. For AWS, we are already using the user data argument to pass a startup script. The downside to passing a script is that Terraform has no way to track if the script is successful or not. It's simply another argument in the configuration. If the script fails, you need to gracefully handle that, or you could go outside of Terraform and leverage configuration management software. There are many different config management options out there, which Terraform can hand off to for post deployment configuration. Ansible, Chef, Puppet are three examples. A common practice is to bake the configuration management software into a base image for a machine and have Terraform use that base image when it creates an instance. If all else fails, you can use Terraform provisioners. You're likely to encounter these out in the wild as you ramp up on Terraform, so let's dig into what provisioners are and why they're usually a bad idea. Provisioners are defined as part of a resource, and they are executed during resource creation or destruction. A single resource can have multiple provisioners defined with each provisioner being executed in the order they appear in the configuration. If you need to run a provisioner without a resource, there is a special resource called the null_resource that allows you to run provisioners without creating anything. If a provisioner fails, you can tell Terraform to either fail the entire resource action or continue on merrily. Which one you choose will depend on what the provisioner is doing. HashiCorp considers provisioners as a last resort when all other options have been considered and found lacking. Provisioners are not creating objects Terraform can fully understand and manage, which puts the onus on you and your team to ensure things like error checking, idempotence, and consistency are implemented properly. There are three provisioner types. The file provisioner will create files and directories on a remote system. The provisioner allows you to run a script on the local machine that is executing the Terraform run. is used as a workaround for functionality that may not yet be in a provider, and it's probably the provisioner you'll see most often. allows you to run a script on a remote system. Most of the time, the file provisioner and the can be easily replaced with a startup script through something like user data. There used to be more types that were specific to configuration management products like Chef or Puppet, but all of those have been deprecated. In case you encounter provisioners out there in the wild, let's look at some examples of how they're configured.

In the file provisioner example, we are first defining how the provisioner can connect to the remote machine to copy those files. It is also possible to define a connection block for all provisioners used in a resource. The connection types are either going to be SSH or WinRM. A provisioner can refer to the attributes of the resource it lives in using the self object.

For instance, here we are getting the public IP attribute of an EC2 instance the provisioner needs to connect to. The source and destination arguments define the files or directories that should be copied to the remote machine. The provisioner does not need a connection block since it is running on the local machine. You can pass it a command to execute and specify which interpreter to use for executing the command, for instance, Bash, PowerShell, Perl or any other interpreter that you have.

The provisioner will need connection information defined in the resource or in the provisioner. can execute an inline script, a script stored in a file or a list of paths to local scripts executed in the order they are provided, which is what I'm showing here in the example. As I said, HashiCorp recommends heavily against using provisioners whenever possible, but you still may encounter them in your Terraform travels. For our configuration,we're going to stick with user data for config. Let's head over to our configuration and update it to grab those website files from the S3 bucket onto those EC2 instances. Our goal here is to update the script that's defined in the user data argument for each instance. Were trying to grab the two files that make up our website from the S3 bucket, copy them down locally, and then move them to the /usr/share/nginx/html directory. So we're going to replace some of the commands that are here with new commands. The good news is Amazon Linux comes with the AWS CLI, so we can use the AWS S3 commands that are baked into the CLI, and the command line will automatically use the instance profile that's been associated with the EC2 instance to authenticate to the S3 bucket. If you'd like to, you can update the command to copy those files over. And when we come back, I will show you the updated script that I have. Here's my updated script.

We're using the aws s3 cp command to copy two files from the bucket to our home directory, that's and then we are removing the default index.html file from the nginx installation and copying the files from our home directory over to that nginx HTML directory.

With our configuration complete, let's step into the next phase, which is to get this configuration validated and deployed. One of the things that Jack from the ops team asked us to do is to make sure that our files are formatted properly, and we can do that by using the terraform fmt command. So I'll open up the terminal window now, and terraform fmt works on any files it finds in the current directory that you run it from.

So if we run terraform fmt, it will look at the current formatting for each of the files in the directory and then make updates to those files to bring them in line with HashiCorp standards for HashiCorp configuration language files. If you're curious about what has changed in those files, go ahead and open up and inspect those files, and you can see how the formatting has changed. The next step in our process is running terraform init again. You might be wondering why. And the reason is because we added a provider to our configuration, and Terraform needs to download that provider plugin from the Terraform registry. So let's go ahead and run terraform init now. if we scroll up a little bit, we can see it installs the HashiCorp random version 3.1.0.

It's going to continue to use the previously installed AWS plugin because our updated version setting doesn't change which version is installed. Now that we have initialized our configuration, the next thing to do is validate our configuration. So we'll go ahead and run terraform validate. And my Terraform configuration is valid.

You may get some errors, so go ahead and remediate those errors now, and then we'll resume by running terraform plan. Remember that we removed our AWS access and secret key from the variables, so we now need to set them as environment variables. If we expand the commands directory and open up m6_commands, here are the commands for Linux and macOS or for PowerShell to set the proper environment variable for the AWS access key and secret access key. Go ahead and update those values and run the command to set your environment variables. I've already run those to set my environment variables.

Once we have those environment variables set, now we can run terraform plan, and we'll send the output plan to m6.tfplan. I'll go ahead and run that now. And based on the plan, we have 11 things to add, 1 to change, and 4 to destroy.

We know we're creating a bunch of resources because we added them, but I'm curious to see what is changing or being destroyed. So let's scroll up, and we see that the target_group_attachment is being replaced because the target_id is being replaced, which tells me that the instances are also being replaced.

If we scroll up to one of the instances, we can see the instance is being replaced. And if we scroll down with the instance, the user data has been updated, which forces a replacement of the EC2 instance. It's interesting to note that adding an IAM instance profile does not require a replacement of the EC2 instance, so it's actually updating that user data that is forcing the replacement. Lastly, the load balancer is being updated in place because we have updated the access logs configuration. That doesn't force a replacement. We're just updating as is. Let's go ahead and run terraform apply "m6.tfplan", and that will apply the changes that were listed in the plan. This will take a few minutes to recreate those AWS instances, so I'll resume when the deployment is complete. My deployment is complete, although I had to run it a second time because the nginx profile I was trying to create already existed.

So I had to delete it and then let Terraform recreate it. So pro tip, make sure you don't already have a profile named nginx_profile. Now that the deployment is complete, let's go ahead and go to the address so we can generate some traffic on our website, which will then cause the load balancer to write data to the access logs. We've removed the ability to differentiate between the two different servers since it's not really necessary anymore. We know that that works. I'll go ahead and refresh the website a few times just to generate some web traffic that will be written to the S3 bucket.

Now that we have generated some traffic for our website, we can go over to the S3 console. Here's the S3 bucket that we created using Terraform. In there, we can see we have two paths. We have and the website. Let's go into the And there we have a folder, AWSLogs, and there we have one based off of our account. And in there, there is a test file that was run when we updated our configuration of the Elastic Load Balancer. It may take 5 or 10 minutes for the load balancer to process new requests and add them to the access log for the S3 bucket. So if you don't see access logs right away, don't worry. They will be there shortly. At this point, we have met all the requirements from both the development team and the ops team. In summary, we learned how to add a new provider to a configuration, and we also saw how to properly specify the version and the source for our provider using the required provider's block. We updated the architecture for our configuration to include an S3 bucket, and in the process, learned about Terraform's dependency graph. Lastly, we talked about why provisioners are a bad idea and other options for performing post deployment configuration. The next step in evolving our configuration is to add functions and looping into the mix. Looping helps us create multiple instances of an object efficiently and dynamically, and functions can help us transform data in our configuration to make it more useful and effective.

Chapter 9 How to Use Functions and Looping

Terraform has some more tricks up its sleeve when it comes to creating a dynamic and efficient configuration. A key feature of any programming language is the ability to create loops and use functions, and Terraform is no exception. We'll kick off this chapter with some new ideas from our old buddy, Jack. He's been reading up on iteration and functions in Terraform, and he has a few ideas to improve our configuration. That means it's time to do some learning of our own. We'll check out what looping constructs exist in Terraform and how they can be used to make our config more dynamic and flexible. We're also going to want to use some functions to fulfill Jack's requests. We'll see what type of functions are available, how they're used in a configuration, and how to test expressions using Terraform console. First, let's check in with Jack and see what suggestions he has. We fulfilled the requests from Samantha when it comes to the deployment architecture, but now Jack has a few suggestions on how our code could be more effective and efficient. To start with, Jack would like to be able to dynamically increase the number of instances deployed for the application. Two instances might be good for development, but in a production scenario he'll likely need more. He would also like to decouple the startup script from the configuration files and store it in its own file for possible updates and reuse. Jack also thinks it's a little cumbersome to set CIDR addresses for the subnets and the VPC.

He'd like to be able to just set the VPC CIDR address and let Terraform split it up among the subnets. Finally, he's noticed that we've been a little inconsistent with our naming of AWS resources. He'd like to be able to add a naming prefix and apply it consistently across all resources. You tell him, not a problem. Terraform and you can take care of it. The updates that Jack requested aren't going to change our architecture. The goal is to keep the deployment the same while improving our infrastructure as code. Let's start by checking out the looping constructs in Terraform. Terraform has several different ways to create multiple instances of an object or manipulate collection objects. We'll start with an overview of the various options and then drill down into the two that are most useful for our configuration. The first looping construct to consider is the count for chapters and resources. Count is used to create multiple instances of a resource or chapter when the instances are very similar in nature, the value for a count argument is an integer, and that includes 0. You can tell Terraform to create 0 of a resource by setting the count to 0, which sounds like an odd thing to do. It's actually super useful when you want to make the creation of a resource conditional on other factors in the configuration. The next construct is the for_each which is also used for chapters and resources. For_each takes a set or a map as a value. It's used instead of count in situations where each instance will be significantly different than the others. You have full access to the values stored in the set or map you submit, and those values can be used when configuring each instance of the resource. That gives you a lot more flexibility than a simple count integer. Dynamic blocks are used to create multiple instances of a nested block inside a parent object. They accept a map or a set for a value to construct the blocks. This is an advanced topic that we're not going to cover in this book, but I included it for completeness. Let's focus in on the syntax of the count and for_each since we will be using both in the configuration. The count can be used for resources or moduls. The syntax for either is the same, and since we haven't touched on moduls yet, we are going to use resources for our example. The count argument goes inside the resource and accepts an integer as a value. The integer determines how many of the resource should be created. In our example, the count is set to 3, so Terraform will create three EC2 instances. When the count argument is used, a special new variable is available called count.index. As Terraform loops through the creation of each instance, count.index will resolve to the current iteration Terraform is on. You can use this value anywhere in the resource configuration block. In the example, we are using count.index to name our EC2 instance's web's number of the iteration. Count starts at 0, making the first instance Using a count argument is going to create a list of resources. Each element of the resource list can be referenced by number. The syntax is similar to standard resource addresses. We start with the ., then we add a square bracket with the element number of the instance we want, optionally followed by the attribute of the instance, if needed.

In our example, if we wanted to refer to the name attribute of the first AWS instance, the syntax would be aws_instance.web_servers[0].name. If you would like to get an attribute of all of the instances, you can use an asterisk in the square brackets. That will return a list containing the attribute value for each instance. The for_each can also be used in resources or chapters. The value for the for_each argument will be either a set or a map. As a quick reminder, a set is an unordered collection of objects. A tuple and a list are ordered collections, so you cannot use a list or a tuple directly, but you can transform a list or a tuple with the toset function. In our example, we are using a map with a set of pairs. Terraform will look at the number of elements in the map or the set and create a corresponding number of instances. In this example, we have two entries in the map, so Terraform will create two S3 bucket objects. In a for_each loop, there are two special variables, each.key and each.value. During the looping process, each.key will be set to the key of the map item currently being iterated over. What about each.value? You can probably guess what it's set to, the value corresponding to the current key. If you are iterating over a set instead of a map, each.key and each.value will be equal to the same thing. Values in the map or set do not have to be a primitive data type, like a string or a number. It could be a complex object with nested values that you'd like to use in each iteration of the resource. Using a for_each argument is going to create a map of resources.

Each entry in the map can be referenced by the key name, just like we've seen when dealing with map data types in the past. The syntax is the resource type, followed by the name_label, and then square brackets with the key string in quotes, followed by dot and the attribute you're interested in. Just like the count syntax, if we want to get the id attribute of all of the instances, we can swap out the key string with an asterisk. The returned value would be a list of all of the IDs. Based on these two looping constructs, let's see if we can find some places in our configuration that would benefit from using count or for_each. Within our configuration, we should be on the lookout for anywhere we are creating more than one of the same resource. If you'd like to look through the config now and make some guesses, feel free to do so. Here's the list that I came up with, starting with the primary resources that we can update. We have two AWS subnets right now and possibly more in the future, depending on how the architecture evolves.

Each subnet is almost identical to the others, except for the CIDR address and availability zone, which makes them a good candidate for the count loop. Likewise, we are creating multiple EC2 instances that are fairly undifferentiated, except for the subnet they attach to. Looks like we'll be using a count loop for them as well. Lastly, we are creating multiple AWS S3 bucket objects, but those have different names and paths, so it might make more sense to use a for each loop to create them. Since we are going to use loops to create these resources, there are going to be other resources that will be impacted as well. We need to create an AWS route table association for each subnet, so we can use a count argument there. We also need to create an AWS load balancer target group attachment for each EC2 instance, so we'll use a count argument there as well. Let's jump over to the configuration and set a few things up.

Let's start by opening our network.tf file. I'll go ahead and expand the directory and open network.tf. And let's scroll down to the definition for our subnet. Scrolling down to the first subnet, we are going to update this first subnet resource for all of our subnets by adding a count argument. What's going to drive that count argument? Let's first set up a variable to define how many subnets we're going to create with this resource. Let's open the variables.tf file.

And we are going to add a value. Let's go ahead and add it below the vpc_cidr_block. We'll call the variable vpc_subnet_count, we'll set the type = number, we'll set the description to the number of subnets to create, and we'll set the default = 2.

Now that we have our variable ready, let's go back to the network.tf file and update the resource block. For the resource block, let's change it from subnet1 to subnets. This is going to create all of our subnets after all. And below there, we will add our argument for count, and we'll set the value of the count = vpc_subnet_count. Next we need to update our CIDR block, and we can use the count.index to select an item from the vpc_subnets_cidr_block variable, so we'll set this to count.index. On the first iteration, it will select the first element from the list, and on the second iteration, it will select the second element from the list, and so on. The vpc_id will remain the same. They're all in the same VPC.

The map_public_ip_on_launch will remain the same. The availability_zone will also need to update with the count.index to select the element from the names list. , we've updated the value to count.index. That's everything we need to change for this resource. Below it, we have our subnet2. We no longer need subnet2 because we're defining all of our subnets with that single resource block, so we'll go ahead and delete this resource block. Now the other thing we need to update our the route table associations for the subnets. Let's scroll down to that resource. Let's rename our first route table association resource And now we'll add the count argument to this resource. We'll set the count equal to the number of subnets because that's how many route table associations we need to create. Now we need to reference each subnet that we created with our subnet resource. We'll use the resource addressing that we learned earlier to create that reference. So it should be aws_subnet, and remember, we changed the resource to subnets. And we want to specify a particular subnet, so we'll add the square bracket, and within the square bracket we'll add count.index.

This way, in the first iteration, it will reference the first element in the list of subnets and the id attribute. And then on the second iteration it'll reference the second subnet, and so on. The route table id stays the same because we're associating all these subnets with the same route table. Now that we've updated this resource, we can delete the resource. And that takes care of updating our subnets and the route table association. My challenge to you is to update the instances with a count argument as well. In the instances file, you can update the first instance to nginx instances or whatever name label you would like to use, and add a count argument. You'll need to add a variable like instance_count for the number of instances that will be created, and within the configuration, you're going to need to reference the proper subnet for each instance. That's going to end up being a little more complicated than you would initially think so. For now, we can safely assume that we just have the two instances and two subnets, one instance per subnet. The other thing you'll have to update is the target load balancer group attachment, and that is in the load balancer.tf. Down at the bottom we have our two target group attachments. You're going to update it so that there is only one that is also using the count argument and referencing the proper target IDs to the instances that you're creating with the loop. So go ahead and try to do that now, and when we come back you can see my updated configuration. Let's see how you did. First, I added a variable for the instance_count, and I added it right below instance_type to kind of keep the same variables together. Instance_count is set to type number, and the default is equal to 2.

Now let's check out the instances.tf file. For instances.tf I renamed the resource to nginx instead of nginx1. The count is set to var.instance_count, and the subnet_id reference I updated to subnets and then the count.index.

We're actually going to change that a little bit in the future, but for now it's okay to leave it like that.

Under loadbalancer I set the count to var.instance_count, and for the target_id I updated the reference to nginx and the count.index for that element out of the list of instances. The last thing to update is our bucket objects. So let's go into s3.tf, and we are going to update the bucket objects to be a single bucket object. We'll start by updating the name label to website_content. Then, we will add a for_each meta argument, and we're going to use a map for our for each. So I'll set the curly braces to indicate a map. The first item in the map, the key will be website, and we'll set it to the path of the website file that we want to upload to our S3 bucket, which would be /website/index.html.

We're going to use the same bucket as the target for each bucket object, so we can leave that the same. The key, which is the path for the object on the S3 bucket, we can set that to each.value, which will use the value that's stored in each map key. And then the source is the path to the file that we want to create as an object. We're going to use the current directory by specifying dot, and then we'll use the interpolation syntax to set it to each.value. So on the first loop, this will evaluate to ./website/index.html. And that way we will create all of our AWS bucket objects. We can also delete the second resource here because we no longer need it. You could make this more dynamic by creating a variable that includes all of the items that need to be uploaded or even use a function of some kind to evaluate all of the files in a directory. If you'd like to do that, I leave that as an exercise to you. For now, we're going to leave this as hard coded values in the for each statement. Speaking of functions, we are going to need to use functions to meet the rest of the requirements that Jack has laid out for us, so let's dig a little deeper into functions and expressions within Terraform. Terraform includes functions and expressions to support the manipulation of data in HCL files. We've already seen the expressions and even some of the functions at work, but now it's time to examine them in more detail. We've been using Terraform expressions for a while now, in particular, the interpolation and heredoc expressions to include resource and variable values in a string or pass an entire string to an argument like user data. Terraform also supports arithmetic and logical operators like and, or, equals, greater than, etc. The evaluation will depend on the data type you are operating on and whether that data type supports the comparison. Terraform also supports conditional expressions, which are essentially an if statement followed by a value to return if true and a value to return if false. You can combine a conditional expression with a count argument to decide if a resource is created or not. The for expression is used to manipulate and transform collections. It can take any collection object type, map, list, set, etc., and it will return a new list or map. For expressions are a great way to work with the set of instances that a count or a for each argument generates from a resource block. Just like any other programming language, Terraform supports functions that help you transform and manipulate data. Unlike provider plugins, functions are built into the Terraform binary, so you don't have to initialize or download anything to use them. Since they don't use an external service or executable, they also evaluate much faster than a resource or data source from a provider. If I wanted to build a model of what a basic function looks like, it's going to be something like this. You have the function name and then parentheses, and then some number of arguments to go with that function. Some functions actually take no arguments, while others take many. Arguments are not named, unlike some other programming and scripting languages, instead, the arguments must be in the proper order. You could test functions by placing them in a Terraform configuration and running a plan, but that's a little time consuming and difficult to debug. For that reason, Terraform has a subcommand called console that opens up an interpretation console where you can have Terraform evaluate functions and other expressions. That is much more efficient than testing things in a configuration directly. Console will also load the current state data values of a configuration, allowing you to use real data to test your functions and expressions.Based on the current Terraform documentation, there are at least nine function categories with more possibly coming in the future. I'm not going to list all of those categories here. Instead, we'll focus on those you'll probably use as you build your first configurations. The first category is numeric functions. These are functions that are used to manipulate numbers. For instance, if I had a list of numbers and I want to get the smallest number, I can use the min function, and it will return, in this case, the number 7. There are also string manipulation functions. A possible use for a string function is working with Azure Storage account names. They cannot have uppercase letters, and if someone provided a string with uppercase letters for a storage account, you would receive an error. There is a function called lower that takes a string and will put anything that's capitalized into lowercase and return that string to you. There are functions to deal with collections. And when I'm talking about collections, I'm talking about lists and maps, basically. We will be using the merge function shortly in our configuration to merge the common tags map with another map. One interesting category is the functions for IP networking. If you've done any work with IP networking, you know that math in IP addressing is kind of funky, and for that reason, the standard numeric functions don't work very well. There are dedicated functions like cidersubnet(), which takes a network range, carves out a subnetwork in that range based off of arguments that you give it. There are also functions that interact with the local filesystem. One of the most common ones to use is the file function. That takes a path argument pointing to a file, reads the contents of that file out to a string, and returns the string. So if you need to get the contents of a file, you use the file function, very straightforward. Lastly, type conversion functions allow you to convert one data type to another. You probably won't use most of these functions often with the exception of toset(). If you'll recall from earlier, the for each argument takes a map or a set, and the toset() converts a list or a tuple to a set. That's pretty useful. If you're interested in looking at the other categories and functions, they are all nicely laid out in the Terraform documentation. Let's take a deeper look at some individual Terraform functions we will use in our configuration.

We can leverage the functions in Terraform to meet the requirements given to us by Jack. We'll start with the Startup script. Currently we define the Startup script using a heredoc expression, but Jack wants us to move that into a file. We could try using the file function, but we need to dynamically update the bucket name used by our script. Instead, we will use the templatefile function, which reads in the contents of a file and replaces variables in the file with values submitted as part of the function.

Another request from Jack was to simplify the networking by determining the subnet addressing dynamically. We can do that by leveraging the cidrsubnet function, giving it the VPC cidr_range and carving out space for our subnets. The function takes the cidr_range you would like to work with, the subnet bits to add to the existing subnet mask, and which network number you want out of the resulting subnetworks. Jack also wants us to consistently name all of the resources. We can do that by adding a variable for a naming prefix and adding a name tag for each resource that doesn't have a name argument. But we already have a list of common tags in a map. What are we going to do? No problem. We can use merge to merge our common tags map with a map of additional tags for each resource. Finally, we are going to be adding a naming prefix variable to the configuration. When we use it for our bucket name, we should apply the lower function to make sure our bucket name is always lower case, even if someone submits an uppercase value with the naming prefix. Ready to add some functions to the configuration? Well, before we try to add functions to our configuration, let's first test out some of these functions using the Terraform console. I'll go ahead and open up the commands directory and open up m7_commands, which has some examples for us to run to try the different functions and syntax. You do need to initialize the configuration before terraform console will work.

We've already initialized our configuration, so we don't have to worry about that. We can simply run terraform console. This starts the interactive environment where we can test different functions, and we can make use of variable values and resource and data source values within our functions. Let's first test out a basic numeric function, the min with the arguments 4, 5, and 16. I'll go ahead and copy this and paste it down below. The result is 5, which is correct. That is the lowest number. Now let's try using the lower function that takes a string. I'll go ahead and copy that one, and paste it down below, and It evaluated our string and set it to all lowercase. Now we're going to test out the cidrsubnet, and we're going to feed it the vpc_cidr_block as a value. The value it uses for the variable is the default value we defined for that variable. If you don't remember, let's go ahead and open up the variables file, and let's find that vpc_cidr_block. It's set to 10.0.0.0/16.

Based off of the syntax, we are going to add 8 bits to that to make it a /24. And the 0 argument says we're going to select the first available network from the set of subnetworks. So let's go ahead and run this command and see what the resulting value is. The resulting value is 10.0.0.0/24. That's exactly what we would expect. And this is an excellent way we can leverage the cidrsubnet function to automatically generate the cidersubnet ranges for the subnets we're creating in the count loop. Before we get to that, let's try a few other functions. The next one I want to try is lookup. The lookup function is used to look up the value in a map. You first have to specify a map in the argument. We'll use local.common_tags, and then the key that you want to look for. We'll specify company. If that key is not found, we can give it an alternate value to return, in this case, unknown. Let's try out this function now. And it returns the value ACME based off the key, company. If we instead use a key of missing, which I know is not in the common_tags map, and then it returns the alternate value we specified, Unknown. In addition to trying out functions, you can also just retrieve a value. Let's retrieve the value stored in local.common_tags, and it returns the map that's stored in our current local.common_tags.

You can also try out some arithmetic operators. One arithmetic expression we're going to use is the modulo operator to assign an instance to a subnet. Now that might not make sense right now. We'll get into that in a moment. Let's first start by moving our startup script to a separate file and making use of that template file function. We are going to move our startup script to its own file and make use of the templatefile function. So for now I will hide the terminal, and let's create a new file in that's going to hold our startup script. I'll name that file startup_script.tpl. You don't have to name it .tpl, that's something that I do just so I know it's a template file. Within that template file we're going to have our startup script. So let's open up the instances file and we'll copy the script that we've defined in user data. So I'll go ahead and copy this entire script and paste it into the startup_script file. You'll note in the script we are referencing the aws_s3_bucket.web_bucket.id attribute.

When the template file is evaluated, it's not going to be able to directly evaluate that expression. We need to put a variable that we can reference in our template file function. So let's change this instead to s3_bucket_name. That's a variable we can now reference in our template file function, and I'll update that for the second entry as well. I'll go ahead and save the file, and back in instances we will replace the current user data arguments with the templatefile function. So I'll go ahead and delete what's and start this off with a templatefile function. The first argument in the templatefile function is the path to the template file we want to use. To start off the path to the startup script, we can make use of a special variable that exists in Terraform, it's the path.chapter variable. This will resolve to the full path of the chapter that we're currently working in. Then we can add a slash and the startup_script.tpl. That's the path to the file we want to use, and now we can provide a map of variables and values to use in that template file.

So I'll add a comma, and then I'll start a map with curly braces. We only have one variable in our template file, which is s3_bucket_name, and I'll set the value to aws_s3_bucket.web_bucket.id, which is what we had in the initial startup script. Now it will pass that value and replace wherever it sees s3_bucket_name with that value. Go ahead and save that. The next thing to do is add the CIDR subnet function to our definition of the subnets. We are going to use the cidrsubnet function to get the CIDR ranges for our subnets. Let's go ahead and open up the network file and scroll up to where our subnets are. There's the cidr_block argument. My challenge to you is to use the cidrsubnet function to define the value for the cidr_block for our AWS subnets. Go ahead and try that now, and when we come back, I'll show you my solution for setting that cidr_block argument. Let's see how you did. We're basically mimicking what we did in the console.

So we have cidrsubnet the function, we're passing it the cidr_block we're using for our VPC, we're adding 8 bits to the subnet mask, and we're using count.index to select the subnetwork that's evaluated by adding those bits. As we saw at the console, the first one should evaluate to 10.0.0.0/24, and the next one will evaluate to 10.0.1.0/24. While we're still thinking about networking, there's something we need to update about our instances. Let's go over to the instances.tf file. You'll notice for the subnet_id we're using the expression aws_subnets.subnets[count.index].id.

That's going to work well when we have two instances and two subnets, but what happens if we want four instances distributed evenly across two subnets? This expression is not going to work anymore because count.index will go beyond the number of subnets that we have. We need a different expression here that evaluates properly, and we can use the modulo expression to do that. So let me show you how that works by bringing up the console again. Going with our example, let's say that we have four instances, and we want to place it in either the first or second subnet. We can use the modulo operator to do that. We would start with the count.index of the first instance, which would be 0, and then the modulo operator, which gets the remainder after a division. And then we'll look at the number of subnets we have, which is 2. That will evaluate to the number 0, so it will get placed in the first subnet.

Our next instance will be instance number 1, and when we do % 2 on that, we'll get a remainder of 1. So far, so good. That's going to go in the second subnet. Our third instance, if we do % 2 on that, because there is no remainder, that will resolve to 0, and it will put it in the first subnet. Our fourth instance will evaluate to 1, so it will be placed in the second subnet. All we need to do is encapsulate this modulo expression so that it puts our instances evenly across two subnets, and this will work for more than just two subnets. Let's go ahead and update our expression to be [(count.index % var.vpc_subnet_count)], and we'll put the whole expression in parentheses so it's evaluated before it tries to get the element. Now we can increase the number of instances beyond 2 and not worry about how they're distributed across our subnets. I'll go ahead and save this file now. The next request to deal with from Jack is consistent naming across all of the resources, so let's get started on that.

Chapter 10 How to Add Naming Prefix

Jack wants a naming prefix to be added to the configuration and then consistent naming across all resources. Let's go ahead and start by adding a variable to the variables file. We'll scroll down and open up the variables file. We'll create a new variable and name it naming_prefix. We'll set the type = string, the description = Naming prefix for all resources, and we'll set a default of globweb. Instead of using this naming prefix directly, why don't we add a local value where we manipulate this naming prefix a little bit?

So let's go ahead and open up the locals.tf file, and we'll add a new local value here called name_prefix, and we'll set that equal to the variable naming_prefix and add to it. We could update this for each environment as we create them, and that's something we'll deal with in a later chapter. The S3 bucket name does not currently use the name_prefix. So my challenge to you is to use this new local value name_prefix and add it to the S3 bucket name instead of and then make sure that the entire S3 bucket name is all in lowercase. Go ahead and try that out now. And when we come back, we'll see my solution. Here's how I approach that solution. I'm using the lower function here to make sure everything in the string is set to lowercase. I'm referencing the name_prefix local.

Did you know you can reference a local inside of ae locals block? You sure can. I'm referencing the name_prefix local, dash, and then the random integer result for the bucket name. The last thing to do is add common naming to all the resources within our configuration. Let's first start with the VPC. I'll go ahead and open up the network.tf file, and let's scroll up to where we define our VPC. We can add a name tag for our VPC by adding an additional tag, but we already have our local.common_tags. How are we going to combine that with a new tag? We can use the merge function.

Let me show you how the merge function works. Be sure to save both the variables and locals file before you try this expression. I'll go ahead and bring up the terminal again so we can test out the merge function in the terraform console. I exited out of the terraform console because it didn't have our new variables and locals loaded into it yet. It only loads those values when you first start up the console. To make sure I have the latest variables and locals, I am going to launch terraform console again. Now we can try to use the merge function. We're going to try to merge together the local.common_tags and add a new map that has a name tag in it.

We start with the merge function, and then in parentheses, the first map we want to use is (local.common_tags, and then we want to add another map to it, so we'll add a comma and start another map with the curly braces. Within that map, all we want to add is a name tag. So we'll start with the key, Name, and set it equal to the local.name_prefix value and add on for the string. Then we'll close our map with the curly braces and close our merge function with parentheses. And if I hit Enter, we have an updated map that will be submitted to the tags argument that has all the values we want. So I'll go ahead and close the terminal, and now we'll update the tags with the merge function. I'll add the merge function, add a comma after local tags and start a map. We'll set the key to Name. We'll set the value to the same value we just used in the console, local.name_prefix and add And then we'll close the map and close the parentheses for the merge function. That's all done.

My challenge to you is to go through the rest of the resources within the configuration. If there's a name argument, go ahead and update the name argument as needed. If there is no name argument, add a name tag to the tags argument using the merge function. A good example of a resource that does have a name argument, if we scroll down to our first security group, this does have a name argument, which will be applied as the name tag.

So we can update this one with the naming prefix. Any resources that don't have this name argument, you can add it to the tags argument instead. One other thing to note is for resources where we're using the count argument or the for each argument, you may want to use that value in the naming tag, so you name each instance of the resource differently. Subnets would be a good example of that where you want to name each subnet based off of the count index. Go ahead and try to make those changes now. When we come back, we'll go through the process of formatting, validating, and deploying our updated configuration.

Let's go ahead and format and validate our configurations. I'll exit the Terraform console by doing exit, and go ahead and run terraform fmt. All of our files are now nicely formatted, let's run terraform validate to make sure we don't have any syntax issues, and it looks like we missed something with our subnets, we're going to have to update our configuration.

Let's go into the loadbalancer.tf file, and scroll all the way up to our load balancer, and sure enough, we're referencing two resources that don't exist anymore. We need to update this. What we want is a list of all the subnets that exist.

If you'll remember from the reference syntax we looked at earlier, we can do this by using an asterisk, so I'll change the reference to aws_subnet.subnets, and then we'll use square brackets and the asterisk to indicate that we want to retrieve all of the instances, and then we'll use .id to get the ID attribute of all of those instances. The object returned is going to be a list, so we can get rid of this other reference of subnet2, and we can get rid of the square brackets, because the object being returned is already a list, we don't want to put a list inside of a list. So I'll go ahead and save this, and now, let's try running terraform validate again.

Our configuration is valid. That's why you always run terraform validate after you make changes, because you're always going to forget something in your configuration.

With a valid configuration, we can go ahead and run through the plan and apply process. Going back to m7_commands, if you haven't already exported your AWS access key and secret key as environment variables, go ahead and do that now. I already have that set, I can move onto the next step, which is running terraform plan, and sending the plan to the file m7.tfplan. Go ahead and copy that, and run it down below. This is probably going to make some significant changes, because we're changing the naming, we're changing some of the subnet references, it's going to make a lot of changes is what I'm saying. So, it's going to add 19 new things, change 3, and destroy 19 things; and that's because, like I said, we're making some pretty significant changes.

Fortunately we're still in a development context, so we can go ahead and run terraform apply to apply these changes to our deployment. Since we're recreating a lot of stuff, this is going to take a while to apply, so we'll resume when the deployment has completed.

Our deployment has completed successfully, it's made all the changes that we asked of it, let's go ahead and make sure our application is back up and running. Let's go over to the EC2 Management Console. Looking at the names for our EC2 instances, we've got I used the count index to help with the naming of the instances. If you'd like to go to your AWS console, you can verify the naming and the creation of all of your resources to make sure your configuration operated correctly. At this point, we have satisfied all of Jack's requirements. In summary, we explored the concepts of looping and functions in Terraform. We started with adding count and for each loops to make our configuration more dynamic. Then we leveraged functions to further improve the code and simplify the required inputs for deployment. In the next chapter, we'll examine Terraform moduls. Terraform moduls are used to package up common configurations for reuse, and they are incredibly useful.

Chapter 11 Terraform modules

A common feature of programming languages is the ability to import libraries or modules for common tasks, data structures, or functions. Terraform implements a similar ability through the use of modules. Terraform chapters stop you from reinventing the wheel by allowing you to use common configurations built by others. We'll start by first defining what a Terraform module is. You've been using a module this whole time and didn't even realize it. Once we've established what a module is and how it's used, we'll check in with Jack to see what improvements he thinks we could make by leveraging modules in our code. Then we'll implement those changes, first by using an existing module from the Terraform public registry, and then creating our own module for S3 buckets. But first, what is a Terraform module? Whether or not you realize that you've been using Terraform modules all along, what is a Terraform module? It is simply a configuration that defines inputs, resources, and outputs, and all of those are optional. When you create a set of tf or tf.json files in a directory, that is a module. The main configuration you are working with is known as the root module, and you can invoke other modules to create resources. Modules can form a hierarchy with the root module at the top. Our root module could invoke a child module, which could in turn invoke another child module. For instance, let's say we use a module to create a load balancer with a VPC and an EC2 instance. The load balancer module may use a module to create the VPC and another to create each EC2 instance. The motivation behind creating or using modules is to leverage a common set of resources and configurations for your deployment. Where can you get modules? They can be sourced from the local filesystem, a remote registry or any properly implemented website that follows the HashiCorp provider protocol. The most common source is the Terraform public registry. In fact, you may have noticed the browse module option on the public registry. Modules that are hosted on a registry are also versioned in the same way that providers are. You can specify a version to use when invoking a module. Staying on your preferred version can prevent breaking changes from impacting your deployments. Once you've added the module to your configuration, terraform init will download the module from the source location to your working directory. If the module's already in the current working directory, Terraform will not make a copy of it. As I mentioned in the discussion of looping, you can create multiple instances of a module using either the count or for_each meta arguments. The components that make up a module should already be very familiar to you. Modules generally have input variables, so you can provide values for input to the module, and output values that are based off of what the module is creating, and, of course, the actual resources and data sources within the module. A module is not required to have any of these components, but it probably would not be very useful without them. Now that we know a bit about modules, let's see what ACME has in mind for using them with our configuration. After learning about modules, you're probably already thinking of how the configuration could be improved and simplified. Looking at the current architecture, we've got our application sitting in a VPC, which we know is made up of subnets, routes, route associations, an internet gateway, and more. That seems like a common configuration that should go into a module. Also, our S3 bucket with the bucket and the IAM policies seems like something that might be used in other deployments at ACME. What does Jack think about that? You caught up to Jack in the hallway to talk about your module idea. Turns out, he's been researching them on his own, and he thinks it would be a great idea to add them to your configuration. He would like everybody at ACME to standardize on the VPC module from the Terraform public registry. He also really likes the idea of creating a module for the S3 portion of the configuration, with all the necessary IAM roles, profiles, and bucket policies that will allow a load balancer to write access logs and EC2 instances to grab website content. Before we try to implement these improvements, first, we need to check out the syntax used to create and instantiate a module. A module really is just a collection of Terraform files in a directory. In the same way that we have been crafting our configuration with files for variables, outputs, and different resource groupings, you can do the same with a module or just put it all in one big file. The contents of a module will include input variables, resources to be created, and outputs that the parent module might want to use. In this example, we are building a chapter for an S3 bucket. The input variable bucket_name can be used to name the S3 bucket. Then we have the actual resource being created. And in that resource, we can use the input variable bucket_name. The parent module is probably going to want some information about that bucket, and we can expose that information using an output value, passing back the bucket_id. Now is probably a good time to mention why input variables and output values are so important to a module. The only way for a parent module to pass information to a child module is through input variables. The child module has no access to local values, resource attributes, or input variables of the parent module. Any information a module might need has to be passed through those input variables. Likewise, the parent module has no access to the local values and resource attributes of the child module, the only way to pass information back to the parent module is through output values. The good news is that the input variables and output values support any data type available in Terraform, so you can pass a complex object, an entire resource, or a simple string. The choice is up to you. Bearing this in mind, let's take a look at how you instantiate a chapter, pass variable values to it, and reference outputs from it. Invoking a module starts with the module keyword, followed by a name label for the module. The rest of the configuration information goes inside the block. The source argument tells Terraform where to get the module from. The source can be the local file system, the Terraform registry, or any other supported source type. If the source supports versioning, you can specify the version argument with a version expression, just like the expressions we used for a provider version. If you'd like to use a specific instance of a provider within a module, you can do so with the providers argument. The value for the argument will be a map where the key is the name of the provider in the child module, and the value is the name of the provider alias in the parent module. If you don't specify a providers block, Terraform will use the default provider instance. The remainder of the block will be pairs, with the key being the name of an input variable in the child module and the value that you would like to pass to the child module.

Let's take a look at an example with our potential S3 bucket chapter. We'll start with the module keyword and a name label of XYZ_bucket. In the block we'll specify the source as the current directory and the S3 subdirectory where we have our module files. Since this is a local source, it doesn't support versions. Beyond that, we can pass the single variable bucket_name to the module, and that's it. The module will take care of creating the resources and making output values available. Let's take a look at how we can reference those values. The general format for referencing a module output is the module keyword, dot, the name label of the module, dot, the output value name. In our example, we can get the bucket id by using the syntax module.XYZ_bucket.bucket_id.

The output value can be any data type, and naming is up to you. We used the attribute name in this example, but we could've called the output value s3_bucket_id or whatever else makes sense in the context of the module. Armed with all of this module knowledge, let's start making use of it. First, we will replace the current VPC resources with the VPC chapter on the Terraform registry. That means going to the registry and reading about the VPC module. We've already gone into the Providers section, so now let's go into the modules section. We can filter by Provider for the modules that we want. The module we actually want is the vpc module, which happens to be the top module. So let's go ahead and click on that module. Within this module, I want to point out a few things.

It has a basic set of instructions on how to use the module. It also provides the source code for the module on GitHub. So if you want to view what's actually in the module, you can click through on that link and view it yourself. Scrolling down a little bit, we have a README section, which describes how to potentially use this module. And then it also gives us a list of inputs that are accepted by the module, output values that are given by the module, any dependencies and the resources that would be created by the module. A lot of this is dependent on the inputs you give the module. We're going to be setting up a fairly basic VPC, so we can simply copy this example and paste it into our configuration and then make some simple updates. I've copied the text, let's head over to our configuration, and I'll expand and open up the network.tf file. And, we are going to be replacing a bunch of resources with this. We're going to be replacing the vpc, the internet_gateway, the subnets, the route_table, and the route_table_associations, all with this module. That's a lot of resources that we no longer have to manage. Let's scroll back up to the top and paste in the module example. Since this module is from the Terraform Registry, we should definitely add a version argument to pin it to a specific version. This way, if the module is updated in a way that breaks our configuration, we can test it in a development environment before upgrading to the newest version of the module. If we go back to the browser, we can see the current version is 3.10. So let's go ahead and pin it to 3.10 for now.

We'll set the version = to 3.10.0, and this way it will only use that version until we change this argument. now we need to update some of the values that are used for the arguments here. We'll first delete the name argument since we'll be submitting that through our tags. Next, we'll update the cidr block to use our variable.

For the availability zones argument, we want to give it a list of availability zone names that's equal to the number of subnets that we're currently using. To do that, we can use the slice function to slice off a list of names from the availability zones data source. Let's see how we do that. The slice function takes a list as input and then slices off a portion of that list for use. We'll specify the data source aws_availability_zones.names. The next argument is the starting index for the slice. We'll start at the first element in the names list. The last argument is the ending index of our slice, and it is not inclusive, so it won't include that element in our list. We'll set that to var.vpc_subnet_count. So when our subnet count is 2, it will return 2 availability zone names and it will already be in a list format. For the private_subnets, we don't have any private subnets, so we can go ahead and delete the private_subnets. For the public_subnets, we are going to need to calculate a list of public subnet CIDR ranges for our public subnets. Let's look at how we've done this already. If we scroll down to our subnets, we compute the CIDR block using the cidrsubnet function and the count.index since we're creating our subnets and accounts. We're no longer generating our subnets in account, so we need an alternate way to generate this list of CIDR subnets. The way that we'll do that is with a for expression. I briefly mentioned for expressions in the previous chapter, now it's time to dig into those a little more deeply.

Chapter 12 For Expressions

For expressions are a way to create a new collection based off of another collection object. It's especially useful when you're dealing with resources that have a count or a for each argument. The input in a for expression can be any collection data type, list, set, tuple, map, or even object. The contents of the collection will be available for transformation in the for expression. The result of the for expression will be either a tuple or an object data type. Remember that these are structural data types, which means the values inside don't all have to be of the same data type. To help customize the result, you can filter it with an if statement. You can filter on any value from the inputs. Let's check out the syntax in a for expression to lend some clarity. First, let's see how you would create a tuple result with a for expression. The expression starts with either curly braces or square brackets. The square brackets indicate that the result will be a tuple. The brackets or braces that you use to encapsulate the for expression determine the result type. After the square brackets, the expression starts with the keyword for, followed by syntax that identifies the input value and the iterator term to use during evaluation. The structure is the iterator term, followed by in, and then the input value. After that we have a colon, which signals the start of the value which will be stored in each tuple element in the resulting collection. If that sounds esoteric and difficult to parse, I agree, and an example will clear things up. Let's say we have a local value called toppings of type list with three elements in it, and we'd like to create a new tuple with the word added to each element in the list. We can accomplish this with a for expression. The square bracket says that we want a tuple as the result. The t is the placeholder for each value in the input value local.toppings. After the colon is the resultant value we want to use for each element, which is simply the string Globo and the value stored in placeholder t. Remember that the input value doesn't have to match the result. Local.toppings could have been a map. Now let's check out the syntax for creating an object. The expression will start with curly braces to indicate that we want an object as a result. As a quick refresher, an object is basically a set of pairs where the values can be of different data types. In this expression, the input value is a map, which means we now need two iterator identifiers, one for the key and the other for the value. Next, we have a colon and an expression to evaluate for each entry in the object. The syntax is the object key, followed by equals and the symbol and then the object value. Again, an example will probably help. Here's a local value called prices of type map with three key value pairs. Let's say we'd like a new object where each price is rounded up to the next whole integer. We can do that with a for expression, where i is the map key, and p is the map value. The expression to evaluate keeps the same key i, but alters the value with the ceiling function. The resulting object has the value for each pair rounded up to the closest integer. We are going to use a for expression to dynamically generate a list of subnet CIDR ranges. Let's head over to the configuration and see how. We are trying to create a tuple of cidrsubnet addresses to pass to the public subnets argument. The number of elements in the tuple should equal the value stored in vpc_subnet_count, which means we'll need an input to the for expression that is a list of integers from 0 to the value in vpc_subnet_count. Fortunately, there is a function called range that will do exactly that. Let me pull up the terminal, and we'll start up terraform console to test out these expressions. First we will test the range function. The syntax is the range function and then the value you want it to count up to. We'll specify the variable vpc_subnet_count. Range will hand back a list of 0 and 1.

That sounds good; that's a good start. Now we have a list to use as an input value for the for expression. For the evaluation portion of the expression, we can use the cidrsubnet function, passing it the vpc_cidr_block variable, 8 bits to add to the subnet mask, and the element value in our input list. Let's try to construct a for expression with that information. We'll start the expression with square brackets because we want a tuple back, we'll have the for keyword to start our for expression, we can set an iterator for our list, we'll call it subnet, and that's going to be in the range, and we'll set the range to var.vpc_subnet_count, which we know will return a list with two elements, 0 and 1.

Then we'll add the colon and then the expression we want to use for the result of each element. We'll do the cidrsubnet function, we'll feed it the variable vpc_cidr_block, 8, to add 8 bits to the subnet mask, and then the subnet iterator. That will be 0 on the first evaluation and 1 on the second. We'll close that parentheses for the cidrsubnet function and close the for expression with a square bracket. I'll go ahead and hit Enter, and we get back a tuple that has two subnets in it. Perfect, that's exactly what we need. And this will be dynamic based off of the values of the cidr_block and the subnet_count. Let's go ahead and copy this entire expression, and I'll hide the terminal. Scrolling up to our vpc module, we can replace the value for public_subnets with our new expression. For enable_nat_gateway, we want to set that to false. We don't have any private subnets, so we do not need a NAT gateway. We'll also set enable_vpn_gateway to false, because we are not using a VPN gateway. For our tags, we can go down and grab the tags argument from our existing VPC resource, and go ahead and paste that as the value for the tags argument in the module. With our vpc module ready to go, we can remove the other resources. I have removed all the resources that we're replacing with the vpc chapter. The next thing we need to do is replace any of the references to our VPC resources with the chapter references. There are two output values that you'll need to use to update the rest of the configuration. The first is the vpc_id, and the second is the public_subnets list. For the vpc_id, we'll update the expression to module .vpc.vpc_id. Vpc_id is the output value from our vpc module.

We also need to update anywhere that the subnets are referenced. For example, let's go to the load balancer. In the load balancer we were referencing all of the public subnets with this expression. We need to update the value for the argument to use the public subnets that are created by the module. To do that, we will update the value to module.vpc.public_subnets, which is a list of all the public subnets.

My challenge to you is to go through the rest of the configuration and update it to use these module outputs. If you have any questions, you can always reference the solution that's in the m8_solution directory. Go ahead and when we come back we are going to create our S3 module. The S3 module we are creating should create an S3 bucket with a bucket policy that allows a load balancer to write logs and the proper IAM resources to grant access to an EC2 instance. Our inputs are going to include the bucket_name, the elb_service_account_arn, and the common_tags to be applied to all resources. Those can all go in the file variables.tf. The resources we need to create already exist in our configuration. We've got the S3 bucket itself, the IAM role, the role policy, and the instance profile. In terms of output values, we are going to use the S3 bucket and instance profile. We can simply return the entire object for each resource and make use of all the attributes within. Let's head over to the configuration and start setting up our module.

Let's create the S3 module inside of the directory. We'll start by adding a directory called modules, just in case we want to add any future modules to our configuration. Within that modules directory, let's add a subdirectory for our S3 module, and we'll name the directory, so that's pretty descriptive of what this module is intended for. Within that directory we'll create three files. We'll start by creating the variables.tf, followed by a main.tf file to hold all of our resources, and then an outputs.tf file for our output values. Within the variables file, I'm going to add some comments here for variables we want to create. My challenge to you is to create these three variables, the bucket name, the ELB service account, and the common tags to pass to this module. Go ahead and try that now, and when we come back you can see my updated solution. Here is my solution.

I've got a variable named bucket_name that's of type string; a variable called elb_service_account_arn, which is also of type string; and then a variable called common_tags, which is of type map with strings as the values for the map. And I set a default for the common_tags in case someone using this module doesn't submit a list of common tags to use in the configuration. Next up, we will add our resources to the main.tf file. So let's scroll down and open up the s3.tf file.

We are going to copy all the resources except for the S3 bucket objects. Those will still be created in part of our main configuration. First, I will grab the S3 bucket resource, remove that from the s3.tf file, and paste it into the main.tf file. Next, I will grab all of the IAM resources below the bucket object resource and paste those into the main.tf file. Let's go ahead and save the s3.tf file. And now back in the main.tf file, my challenge for you is to update the references here to use the input variables for all of the resources. Go ahead and try that now, and when we come back we can review my updated solution.

In my updated solution the bucket value is going to be var,bucket_name. In the policy statement, we're now using the variable elb_service_account_arn instead of the data source, and for the references to the bucket_name, we'll use the variable bucket_name. Scrolling down to the end of the S3 bucket resource, the tags have been updated to use the var.common tags. For the IAM role, I've updated the naming to use the bucket_name as the beginning of the name for the role, and I've updated the tags to be var.common_tags.

For the role policy, I'm also using the bucket_name to name the role policy and updating the reference to the bucket_name to use the bucket_name variable.

And down in the instance profile, I updated the name to use the bucket_name for naming and also updated the tags to use var.common_tags. That's everything that's in the main. The last thing to do is to create two outputs. I'll go ahead and save the main.tf file, and let's go over to outputs and I'm going to put two comments for the bucket object and the instance profile object. My challenge to you is to add the output values here to pass the whole bucket object and the whole instance profile object back up to the parent module. Go ahead and try that now, and when we come back we can view my updated solution.

In my updated solution, we have the output web bucket, which is set to a value of aws_s3_bucket.web_bucket. Because we don't specify an attribute, it will pass the entire bucket object back as an output value. That's pretty useful. And then we do the same thing for instance profile, referring to aws_iam_instance_profile.instance_profile. We'll go ahead and save this output file, and now we need to add the module reference to our s3.tf file. I'll go ahead and select that now. Now my challenge to you is to add the module block with the proper input variables. Try it out, and when we come back you can see my updated solution.

We're creating a module with the name_label web_app_s3. The source will be the current working directory / We have to give it three input variable values, bucket_name set to the local.s3_bucket_name, elb_service_account set to the elb_service_account data source, and common_tags set to local.common_tags. Now that we've updated to use a module, we're going to have to update any references to the bucket or the instance_profile with the output values from this module. For instance, our bucket argument in the aws_s3_bucket object should be updated to module.web_app_s3.web_bucket.id. My challenge to you now is to update any other references to the bucket or the instance_profile in our configuration to use the proper output value from our S3 module. Go ahead and do that now, and then we'll review my updated configuration when we come back. In the loadbalancer file, I simply updated the access_logs argument to use the S3 bucket created by our module.

Over in instances, the iam_instance_profile has been updated to use the instance_profile output value, .name. We had to update the depends_on, because it was referencing the iam_role, but that's not available to us anymore, so instead we just make it dependent on the module itself. And then in the templatefile function we have to update the s3_bucket_name to be the web_bucket output value and the .id attribute. While we're looking at the instance configuration, one thing I want to point out is the configuration for the subnet_id. At the end of the module expression, I have removed the .id attribute, and that is because what's returned by the module is actually a list of public subnets and not the public_subnets object itself, which would have the id attribute. That's going to do it for all the updates to our configuration. Go ahead and save those updates. The next step is to get those updates deployed. We have updated our configuration to use a VPC chapter from the Terraform Registry and an S3 module that we wrote ourselves. Let's go ahead and get those changes deployed. I'll go ahead and open up the m8_commands file. And the first thing we need to do is run Terraform in it because Terraform needs to get these modules and include them in the configuration. So I'll go ahead and open up the terminal. And we'll run Terraform in it. Terraform has successfully initialized. If we scroll up, we can see under initializing modules that it downloaded the VPC module and placed it in the .terraform\modules\vpc folder. Because our module is already in the local directory, it will not try to download or copy the files for that module. Now that we have successfully initialized Terraform, the next step is to run terraform fmt. But we don't just want to format our files in the directory, we also want to make sure that the files in our S3 module are formatted properly.

And we can do that by running terraform to go into those subdirectories and properly format those files as well. And it has updated the formatting for all of our files. The next step is to run terraform validate.

Our updated configuration is valid. The next step would be to export your environment variables if you haven't already done so. And after that, we'll run Terraform plan and send the output to m8.tfplan. We'll go ahead and run that now. Since we're moving a lot of things to modules, it's also going to have to recreate a lot of our infrastructure. Let's go ahead and run terraform apply to apply all of these changes. If you happen to be running in production and you needed to move resources from the root module to a child module, that's a case where using the terraform state mv command can help you move things from the existing address to a new address that's inside the module, and that would stop Terraform from destroying the target infrastructure that you're trying to update. That's a pretty advanced topic, which we're not going to get into here. For our purposes, we're still in the development environment, so tearing down this infrastructure and recreating it is no big deal. This is going to take a while, so we'll resume when the deployment has completed successfully.

Our deployment is successful. We are now using a VPC module and an S3 module. In summary, Terraform modules are incredibly useful. You can find existing chapters on the Terraform Registry and save time and effort by not reinventing the wheel. You can also write your own modules to assist with creating common configurations in your organization and share those modules internally or publish them on the Terraform Registry. It turns out we've been using modules this whole time. The root module is simply the module you're currently working in. It has input variables, resource and data sources, and output values, just like any other module. The last thing we will cover is how to use the same configuration to spin up multiple deployments. We can leverage Terraform workspaces to manage state data for multiple environments.

One of the great things about infrastructure as code is its reusability. With a few minor tweaks, the same code can be used to deploy nearly identical infrastructure in multiple environments. Terraform has a special feature called workspaces to help with reusability. We'll start by talking to Samantha over in software development. She's ready to move this web project out of development and into production, and she wants to make sure each environment is consistent. With that objective in mind, we'll take a look at how Terraform can be used to handle multiple environments with the same configuration. We will have to consider things like input values and state data management. One way to handle state data is with Terraform workspaces, and we will put that into practice with our configuration. But first, let's have a chat with Samantha. Our configuration and deployment for the web app at ACME has evolved and improved since that base configuration was handed to us. Now ACME is ready to roll this little project into a staged environment workflow. We can think of our current deployment as the development environment. We've kept things small with tiny EC2 instances and only using 2 subnets. That is probably not going to work so well in a production environment. Samantha has approached us about adding a user acceptance testing environment and a production environment to the existing development environment. She would like us to make sure we are using the same configuration for each environment but supplying different input values depending on the environment. Fortunately, we have used variables extensively in our configuration, so it should be relatively easy to make adjustments for input values. Then we can feed those input values into our single configuration and use it to create and maintain each environment. But what about state data, credentials, and the deployment workflow? Let's examine what it means to deploy multiple environments and how to leverage Terraform workspaces. We are going to support multiple environments from a single configuration. Before I introduce Terraform workspaces to help with this goal, let's first think about some of the challenges inherent in supporting multiple environments. When you're working with multiple environments in Terraform, there's a few things to bear in mind. First of all, generally speaking, your environments are going to have more in common than they have differences between them. That's kind of the whole point of having multiple environments in a configuration. Your dev should be very close to your UAT, and your UAT should be very close to your production because anything you test and validate in UAT should be what actually ends up in production. It also means that it's useful to have abstractions within your configuration where you can apply those different values, making your code more reusable. We've done that by making extensive use of input variables. Another thing to consider in the whole process is access. You're probably not going to have access to production if you're the one deploying to the lower environments. Often there's a separation of responsibility and access between what's called the lower environments and the production environment, so it's important to keep that in mind, especially when you're thinking about dealing with access keys and secret keys. One of the ways to create multiple environments in Terraform is by using workspaces, and this is the HashiCorp recommended way of working with multiple environments. We'll see how that works in a moment. There are some decisions you need to make when it comes to having multiple environments. First is the management of your state. Where is your state data going to live and how are you going to manage the state data for multiple environments? Typically, it's not a single state file for all of your environments. Instead, you'll have your state data stored separately for each environment, or in more complicated setups, you might actually have a state file for your networking and a separate state file for each application running in an environment. That's a lot of state data to manage. Then you have to determine where you're going to store your variable values. Where are these values coming from? Are you going to store them in a file, are you going to submit them at the command line or are you going to use some tool to generate these values and submit them to Terraform? You also have to think about credentials management. You're not necessarily going to use the same credentials to deploy to production as you do to the lower environments. And in fact, in a lot of places, you use a different set of credentials for each environment. How do you manage those credentials and where are they stored? And finally, there's a balance to be struck between the complexity of your configuration and the amount of administrative overhead there is to maintain that configuration. You could go with something that is relatively simple, but requires a significant amount of admin overhead or make something that's fairly complex, but also dynamic and robust, so when you want to add or edit an environment, there's not a whole lot of administrative work to do. Let's look at two examples of how you could potentially manage multiple environments. In this example we are going to manage our environment using multiple state files and multiple configuration value files. We have our primary folder where our Terraform configuration lives, along with a common set of variables that is the same across all environments. And then we can have folders for each environment, dev, uat, and prod. When we are running our terraform plan, we can specify that we want to store the state file in one of those directories. For instance, if we're running terraform plan for development, we can say, place the state file in the dev folder and call it dev.state. Then we can specify a variable file called common.tfvars that has our common values within it. And then finally, an additional var file called dev.tfvars that's in the development folder that has our development values. Everything to do with development is stored in the development folder. We could proceed with the same for uat and with production.

That's one way you can manage your state data and your variable values. Another potential way is to use workspaces. In workspaces, you still have your primary directory where your main configuration and your Terraform tfvars files exist. Workspaces will manage the state for you. It creates a terraform.tfstate.d directory and places the state files and child directories within that main directory. When you want to create a new workspace, you simply run the command terraform workspace new, and the name of the workspace.

Terraform will create that workspace and switch you over to that context, and then you can just run terraform plan using your main configuration and the terraform.tfvars. Rather than manually managing your state, now Terraform is managing the state for you. But how do you get the individual value settings for each environment dynamically based off a workspace? Let's take a look at how we could do that in our configuration. For our three environments from Samantha, she would like us to change the following values based on environment, the VPC CIDR range, the subnet count, the instance type, and the number of instances. She wants us to use the values that you see in this table.

We can accomplish this goal by using a map for each variable and the special value terraform.workspace, which resolves to the currently selected workspace. Let me show you what I mean by making an update to our configuration.

As I just mentioned, there is a special value called terraform.workspace, which evaluates to the currently selected Terraform workspace. Let's bring up the terminal now, and I'll go ahead and enter the terraform console. We can retrieve the special value by simply typing in terraform.workspace. The current terraform.workspace value is default, and that's the only workspace we have available. The default workspace cannot be deleted, and it's selected by default when you create a new Terraform configuration.

We can make use of the terraform.workspace value throughout our configuration. For example, we could update a setting in our locals.tf file. Let's go ahead and expand the directory and open up the locals.tf file. I'll go ahead and hide the terminal to give us some more room, and if you remember from earlier, we created a name_prefix local value, and we added to the end of that value. But instead of doing that, why don't we use the name of the terraform.workspace? So I'll go ahead and delete dev off of the end and update the value to terraform.workspace. The naming prefix will reflect the environment that it's deployed in. We can also add an additional common tag called environment and set that equal to terraform.workspace. Any resource that uses the common tags will have an environment tag equal to the terraform.workspace. We can also use the value of terraform.workspace to select a value from a map in our variables. So let's open up our variables file and update the type for one of the variables that Samantha specified. As a quick reminder, those variables are the CIDR block, subnet count, instance type, and instance count. Let's update the variable vpc_cidr_block with the three CIDR block values she wants for development, uat, and production. I'll update the type to a map of strings, and I will remove the default value and make sure that we specify an appropriate map value in our terraform.tfvars file. Speaking of which, let's go ahead and open that up. In our terraform.tfvars file, we will add a value for that variable. So let's go into mode here.

We'll have terraform.tfvars open on the right and variables.tf open on the left. I'll grab the variable name from the left side and paste it , and it has to be set to a map. And we'll add three map values , one for development, one for uat, and one for production. We'll set Development to 10.0.0.0/16, UAT to 10.1.0.0/16, and Production to 10.2.0.0/16, just like Samantha wanted us to. Now how do we make use of this in our configuration? I'll go ahead and save the terraform.tfvars file and exit out of mode, and let's open up the network.tf file. Within the arguments for our VPC chapter, we now have to update the way that we are getting a value out of our variable vpc_cidr_block. And since it's now a map, all we have to do is add square brackets at the end and set the value we want to retrieve to terraform.workspace. Now Terraform will evaluate what workspace it's currently in and select that value from the map that we have stored in vpc_cidr_block. That means we have to make sure when we create our workspaces, we name them to match the keys that are in the map for the vpc_cidr_block. My challenge to you is to update the other three variables and add values into terraform.tfvars and update all the variable references in the configuration to use terraform.workspace. Go ahead and try to make those changes now, and when we come back we'll take a look at my updated configuration. Let's see how you did.

In the variables file, the vpc_subnet_count should now be a map of numbers, the instance_type should be a map of strings, and the instance_count should be a map of numbers, and all of them should have no default value.

In our terraform.tfvars file, you'll now have an entry for each of those variables with the values set for each environment. Over in our network, just to take a look at how the vpc chapter is configured, we already updated the CIDR block, for the availability zones we had to add the terraform.workspace to the vpc_subnet_count, and then also add it for the public subnets for the vpc_subnet_count and the vpc_cidr_block.

The easiest way to update everywhere is to do a simple find and replace of all the instances of those variables with the appended square brackets and terraform.workspace. Now that our configuration is updated, it's time to make use of Terraform workspaces. But before we do that, we have to talk a little bit about dealing with sensitive data.

Chapter 13 How to Manage Sensitive Data

Credentials and other sensitive data is going to be part of your Terraform configuration. The question is how to deal with that information and keep it secure. One option is to store it in a variables file that is not committed to source control. That is not especially secure, but it sure is easy. The option we've selected for our AWS credentials is to store them in environment variables, and you can use that for any variable in your configuration. It's not uncommon in a deployment pipeline to load sensitive values from a secrets management service into environment variables on the system running the deployment. That is definitely more secure. You want to make sure to mark those variables as sensitive so the values aren't displayed in clear text in your logs. The most secure way is to directly integrate a secrets management service as a data source or a resource in Terraform. When the configuration is being deployed, Terraform can dynamically retrieve the sensitive data and use it in the configuration without it ever being stored even in environment variables. In each case, sensitive data in variables should be marked as sensitive and state data should be written to a secure location. State data will contain sensitive information stored in clear text. Be sure to properly store, secure, and encrypt that data as needed. Now let's head back to the configuration and make use of workspaces. Back in the configuration. Let's go ahead and open up the commands m9, and before we do anything else, let's run terraform format. I'll go ahead and open up the terminal and run terraform format.

Now all of our files are properly formatted. Next, we'll run terraform validate to make sure our configuration is valid. Our configuration is valid.

Now if you haven't already exported your environment variables for your AWS access key and secret key, go ahead and do that now. Scrolling down, we are going to create a new workspace called development. The command is going to be terraform workspace new Development.

Terraform not only created the new workspace development, it also automatically switched us over to the Development workspace context. If we look over to the left in our directory, there is a new directory called terraform.tfstate.d. If we expand that directory, we can see there is a Development folder in there, which will hold our state once we've run a Terraform plan and apply. If we'd like to get a list of all the existing workspaces, we can run terraform workspace list, and here it shows the default workspace and our Development workspace, and the asterisk shows which workspace is currently selected.

With our Development workspace selected, let's go ahead and run terraform plan and send the output to m9dev.tfplan. I'll go ahead and copy this command and paste it down below. This is going to be a deployment, so it's going to have to create all of the resources we've defined in our configuration. It's a total of 24 resources that it is going to create.

Let's kick off the terraform apply, and obviously this is going to take awhile, so we will resume once the environment has been created.

Our deployment is successful, our development environment is up. Before we check out that public DNS link, I do want to point out that this is separate from your default deployment. If you still have that up and running, you're probably going to want to switch to the default workspace and destroy it so you're not paying for those instances. You'll also need to add a default key to the values in tfvars for the values that match what was deployed in the default environment. We've created the Development workspace and deployed our development environment. Now it's time to create our UAT workspace and deploy it to the UAT environment. First we have to create the Terraform workspace, so we'll run terraform workspace new, and the name of the workspace will be UAT. Terraform has created that new workspace and switched us over to that workspace. If we expand the terraform.tfstate.d directory, we can see there are development directory with a terraform.tfstate file in it and a UAT directory that is currently empty. Let's run a plan in our new UAT environment. We'll run terraform plan and send the output to m9uat.tfplan.

Once again, this is a wholly new environment, so it's going to have to create all of the resources for that environment. Now the number of resources is different. It's 28 now, and that's because we're using the UAT environment. Remember, if we look at the tfvars file, for our UAT environment, we are deploying two subnets, so that stays the same, we're deploying slightly bigger instance types, t2.small, but now we're deploying four of those instances to be distributed evenly across those two subnets. With that in mind, let's go ahead and run the apply. Just like the previous deployment, this is standing up all new infrastructure, so this will take a little while. Our deployment is complete.

If we take a look at the EC2 Management Console, there are now eight instances in the account. Two are from the default workspace, two are from the development workspace, and four are from the UAT workspace, and all of them are named in a way that is very obvious.

If we select one of them and take a look at the tags associated with it, we can see that environment is set to UAT. So we could search on that tag within our account to find everything associated with the UAT environment. If you would like to set up and deploy the Production workspace, I invite you to do so now. The last thing I want to show you is how to select and destroy an environment so you can tear all of these environments down. We currently have the UAT workspace selected. Let's see how we select a different workspace. The command for that is terraform workspace select, and then the name of the workspace. In this case we'll select Development, and now we can destroy our development deployment by running terraform destroy That will tear down the development environment, if you'd like to repeat that for the UAT environment, go ahead and do so, and you should definitely do that for the default workspace as well. You can also delete those workspaces with the delete command except for the default workspace, which cannot be deleted. Now we can tell Samantha we have successfully achieved her goal of multiple environments with a single configuration and different input values. In summary we leveraged a single configuration to deploy and maintain multiple environments. This is one of the big benefits of using infrastructure as code. One thing we had to consider was the variances between the environments and make sure our input variables gave us flexibility to manage those variances with input values. We also looked into how we might deal with sensitive data like credentials and secrets. Finally, we got to use Terraform workspaces to manage our environments and state data using the special value terraform.workspace to control variable values. This is the final chapter in the book, and if I had to summarize things, some of the key points that I would like to bring forward are, number 1, it is so beneficial to build your infrastructure automagically. Removing yourself from the manual build process does a lot of great things for you. Most importantly, it ensures your deployments are going to be consistent and repeatable. Doing things manually leads to mistakes. We are fallible creatures, and that's just the way it is. When you codify your infrastructure, you're creating a way to consistently and repeatedly deploy that infrastructure. You're also creating configurations that are going to be reusable for different applications, especially when you use modules to take common patterns and abstract them to a shared resource that a bunch of people can take advantage of. Removing manual configuration and creating reusable configurations allows you to significantly boost your productivity. You're no longer bogged down in these manual processes. You're not troubleshooting these weird issues that were caused by someone making a mistake, and the reusable patterns means that you don't have to deploy the same thing over and over. You can do work that is more interesting and more challenging. Ultimately, it is about making your job better or, failing that, finding yourself a better job, and that's really what this book is all about. I want to help arm you with the skills to make deploying infrastructure less of a chore. And if all else fails, I want to help give you the skills to find the job that you want if you don't have it today. If you're looking for the next step in the world of Terraform, I would recommend checking out the other books about Terraform. I'd also recommend checking out the other HashiCorp products like Vault or Packer. Vault is a platform that can help solve the question of what to do with sensitive data. And Packer helps with creating gold images for use by something like Terraform. I hope you found the content valuable, and I welcome your feedback and comments.

BOOK 9

AUTOMATION WITH TERRAFORM

ADVANCED CONCEPTS AND FUNCTIONALITY

RICHIE MILLER

Introduction

In this book, you are going discover different ways that Terraform can further integrate within your workflow and engage with other toolsets that you have in your organization. First, we are going to set the stage on what you should know already and what's coming up in the rest of this book. But the first question you have to ask yourself is, are you ready for this level of Terraform integration? In order to successfully complete this book and really get the most out of it, there's a few things you should already be familiar with, and the first of those is the Terraform command line. I'm assuming throughout this book that you're comfortable with the Terraform command line and the various commands you can issue such as plan, apply, and destroy. We are going to be getting into some of the advanced commands, such as the import command, but I do expect that you have a basic understanding of the Terraform command line in general. The next thing I would expect you to have familiarity with are variables and values. And, variables within the Terraform configuration file hold information that the configuration will use. And when I mention values, I'm talking about the different types of values that a variable can hold, such as a list, string, or an array. You should also be familiar with the concept of providers and provisioners within Terraform, providers being the thing that Terraform hooks into to provision resources, provisioners being the action by which resources are further provisioned once they are created. And finally, you should have an understanding of some of the basic resources that exist within Terraform. We're mostly going to be working in AWS, so understanding some of the AWS resources will also be important. And lastly, we're also going to be working with modules, so if you're not comfortable with the concept of modules, you may want to do a little bit of reading prior to this. Or, you could go back to the modules section of my previous book to refresh yourself about the world of modules and gain a firmer understanding of what's involved there. We will also be integrating some additional technologies in this process. The first one is Amazon Web Services. That's the public cloud provider we're going to be using for instantiating most of the resources in this book. We're also going to be using Docker to spin up containers. Specifically, we're going to spin up a container running Jenkins for CI/CD pipelines, so we're going to need Docker to spin up that Jenkins container so we can run it locally. We're also going to be taking advantage of some software called Ansible for configuration management. You don't have to be an expert in Ansible. Don't worry about that; we're going to keep it pretty high level. And then finally, we are going to be using another HashiCorp product called Consul, for both the storage of remote state and the storage of configuration data that gets pulled in as a data source by Terraform. One of the technologies we will not be using in this book is Terraform Cloud. I wanted to briefly explain what Terraform Cloud is and why we're not including it in this book. Terraform Cloud is software as a service that HashiCorp offers. It's basically their Terraform enterprise product but repackaged as a service, and it can do a bunch of interesting things, one of which is holding remote state. So if you would like to store your state data in a remote location, you can just select Terraform Cloud for that integration, and it works really well. It also allows you to remotely execute things from the command line, so rather than your Terraform commands running locally to provision resources, they are instead running in Terraform Cloud in a container or virtual machine they spin up on demand. It also allows you to store the values for your variables and your secrets, things like AWS credentials, up in Terraform Cloud. And it integrates with source control, so you can hook it into a repository running on GitHub or in GitLab, and when you make a commit to that repository, it could kick off a job from Terraform Cloud. Terraform Cloud can do all of these things, and because it can do so many different things, it really deserves its own book. For the purposes of this book, we are going to focus on what's happening inside the Terraform CLI and the way that it integrates with some other tools that you might already be running. In this book, we're going to be using our simulation with the ACME company. So if you've the previous book, you will know that you were wildly successful at deploying a new project at ACME using Terraform. You did that project on your own from an administrative standpoint, but no admin is an island and you have a team of admins you need to work with in order to provide ongoing management of infrastructure, and in fact, ACME is now asking you to use Terraform across all of the infrastructure that they're deploying in their public cloud. Also, the Terraform software lends itself to CI/CD pipeline integration, so the technology leaders at ACME would like you to integrate your Terraform configurations with their development team. And finally, they want you to automate all the things, meaning they would like to pack as much automation into each deployment as possible. So the deployment of infrastructure, the deployment of code, the testing of infrastructure, the testing of code, and the ongoing configuration of that infrastructure on day two and beyond needs to be automated as much as possible. But how are we going to cover all of those goals from ACME within the context of this book? Well, the first thing we're going to do is import existing resources. As I said, ACME wants to use Terraform for all of their infrastructure, but of course, they already have infrastructure deployed, so how do you get that infrastructure moved into Terraform? Once that infrastructure is moved into Terraform, you're probably going to want to take a look at that state file that holds all that information and perhaps move it to a remote location. That's for two reasons, one if your local desktop or laptop fails you don't want to lose that state data, and the second reason is because you might want to coordinate with other members of your team as they are making updates to the overall configuration. The next thing we're going to take a look at is data sources and templates, so we're going to see how you can use data sources to pull in configuration data and have that inform other resources, and how you can use templates to streamline your configurations. Once you've got all those things together, we're going to integrate the deployment with a CI/CD pipeline, working across multiple environments. And then finally, we are going to integrate our deployment with a configuration manager to finish the configuration and deployment of an application after Terraform has created the virtual machines, and we're going to be doing that with Ansible software. So the first question is, are you ready for the challenge? Are you ready to learn more and deep dive into the world of Terraform? I hope you're ready to go. The next step in the process is to start preparing your environment, and that's what we're going to cover in the next chapter. We're going to deploy a configuration and import some resources. There's not a bunch of prerequisites you have to set up, but I will step you through what you need to do. And we're going to get a little DevOps about this. So you will be introduced to concepts like CI/CD and source control. We are going to keep it fairly high level, and if you want to dive deeper into any of those topics, check out other books about DevOps and CI/CD. The whole purpose of this is to take your Terraform to the next level, and that's what we're going to do in this book. So get ready; because we are going to see what happens when you need to work with existing resources.

Chapter 1 How to Work with Existing Resources

There is a really good chance that when you're first deploying Terraform, there is going to be existing resources out there that you might want to bring into the management realm of Terraform and that's what this chapter is going to be dealing with. What are we going to cover in this chapter? Well, first of all, as I alluded to in the introduction, Greenfield is really the exception when it comes to introducing Terraform into an environment. There is a pretty good chance that you're going to have some existing infrastructure that needs to be fed back into Terraform so that's what we're going to be focusing on in this chapter. Another reason that you might have some infrastructure that needs to be imported is sometimes you have admins that want to help, but they don't necessarily know Terraform or how to use it or perhaps they're working on a new project and they don't know that Terraform is the standard you've adopted in the organization. So they set up all this infrastructure for a developer and now that needs to be brought under control. There are some important caveats about importing resources and one of the things I want to point out is that the import command assumes that you've already created a configuration for the resources within your Terraform config files. So if you have something running in AWS and you want to import that into Terraform, you have to add it to the configuration first. We'll talk a little bit more about that later. But first, let's take a look at the ACME environment and what's going on there. At ACME, they are using AWS for their main cloud provider. They've looked at the other clouds and they've decided, AWS is the one for us for now. They're also planning to use Terraform for all of their network deployments, and that is in a large part thanks to your good work that you did in the previous book. You deployed out a whole environment for ACME, and they said, we want to adopt this everywhere! One of the things they really love about Terraform is the consistency and security it brings to the network. Because everything's defined in code, it's very easy to see that things are set up properly, and as long as you're managing everything with Terraform, you know that what's in the code should map to what's in the environment. They also really want to set up tagging per company policies. So when new resources are deployed within AWS, they should have the standard tags that have been defined via company policy. Let's take a look at what the current environment is that you've deployed out with Terraform. The current environment is using a VPC that's located in also known as North Virginia, and it has a CIDR range of 10.0.0.0/16. As you may already know, a region in AWS is broken up into multiple availability zones. Within the VPC, we're going to be dealing with three of those availability zones, 1b, and 1c. Currently, you have subnets deployed in two of those availability zones. You have two subnets that are using 10.0.1.0/24 and 10.0.2.0.0/24, and two private subnets, 10.0.10.0/24 and 10.0.11.0/24. This is the configuration that you've already set up with Terraform. In reality, you probably don't have this stuff set up in an environment in AWS, so the first thing that we're going to do in this chapter is deploy out this environment using Terraform. What are we going to do? Well, the first thing we're going to do is examine our Terraform files. We'll get a feel for what's in our existing configuration, and then we're going to deploy that configuration to AWS, and then finally, we'll review the results of our deployment. I'm hoping that you want to play along, and that means you're going to need a few things to do that. Most obvious, you're going to need an AWS account. Fortunately, it's very easy to sign up for an AWS account, and it comes with a free tier, so you don't have to worry about it costing you any money. You'll also need the AWS CLI installed locally because we're going to use that to cache our credentials rather than putting them into the Terraform configuration. You're also going to need the Terraform CLI, so you can download that from HashiCorp's website. With that out of the way, I have one more thing to mention just broadly about the entire book. For the most part, I have done my best to keep the resources in the free tier for AWS. So hopefully, it won't cost you any money, but I can't guarantee that. So some of the resources deployed in AWS may cost some small amount of money, and I just want you to know that. You've been warned. We are in Visual Studio Code, that's my preferred code editor, you can use whatever code editor works for you. Some of these chapters do build on top of each other, but other ones are fairly standalone, and I'll call out when something is dependent on a previous chapter and when it's not. I have the Terraform configuration broken up into variables, resources, and outputs, and I also have a tterraform.tfvars file. If you have a terraform.tfvars file in the same directory as your Terraform configuration, it will automatically use that file for the values for your variables so it's very helpful to have it there. If you're following along, you won't have that file yet, you'll have the terraform.tfvars.example file.

And if we take a look at that, it basically has all of the same information that you're going to find in the terraform.tfvars, but you may want to change some of this. So all you have to do is copy this file and rename it terraform.tfvars and you can follow along. Then we have a step by step files of what we're going to be doing.

The first thing we need to do is configure an AWS profile with proper credentials and we're going to be using that profile for the remainder of the book. You're going to need an access key and secret key so go to the AWS Management console, go into IAM, and provision an access key and secret key for yourself. Then we're going to run aws configure So we're creating a profile named Go ahead and open up the command line down here and I'm going to go ahead and copy this and paste it down in the command line.

I have already entered credentials for this profile so it's showing me an obscured portion of those credentials so I have my access key, I have my secret access key, and I've set my default region to You don't have to use but it's what I'm using for these examples. And then finally, my default output format is set to nothing. Now we have our credentials stored in the shared credentials file and that's something Terraform can use for authentication when it's trying to create resources in AWS. We're going to be running some AWS CLI or AWS PowerShell commands later, and because we want to specify this deep dive profile, you're going to need to set the environment variable aws_profile to So if you're on Linux or macOS, you can run the export command. If you're on Windows running PowerShell, then you can set the environment variable to using this command. Now we're getting into deploying the current environment, but before we do that, we have to review what's in the configuration. So we have set up our AWS credentials in a profile called and now let's take a look at the Terraform configuration. We are going to start with our variables.

So within our variables, we have a few things being set here. The first one is the region, and it is set as a default of You can submit a different value in your tfvars file if you want to use a different region. I'm going to leave the default at For our subnet_count, we have that defaulted to 2, so it should deploy 2 public_subnets and 2 private_subnets. For our variable cidr_block, we're using the cidr_block 10.0.0.0/16 as our default value, and then we have 2 more variables defining the private_subnets and public_subnets. And we don't have a default value for those; we're going to supply that through our tfvars file. Let's take a look at our tfvars file.

Our tfvars file has a list for the private_subnets, and those values are the two CIDR ranges we want to use for the private_subnets. Then we do the same thing for the public_subnets, and we're setting subnet_count equal to 2, which, if that was different than the default value for subnet_count, would override what the default value is. That's everything we have in tfvars right now. Now let's take a look at resources.

What are we actually deploying here? First, we have our provider. We are using the AWS provider because we're creating resources within AWS. I have the version set to 2.something. There is a version 3 that's coming out, or may already be out, depending on when you're reading this, and it may have some breaking changes, so I've pinned the version to the 2.x of AWS provider just to avoid any potential breakages. And we're setting the region to the value that's in our variable region. In data, we're pulling in some information about our AWS environment.

We want to get the list of availability zones so we can deploy out the VPC and some subnets. And then we are using a module to deploy our networking, and this is the VPC chapter that you can find in the public Terraform registry.

This is the one supported by AWS. And I have the version set to 2.44 because I know new versions are going to be coming out that use that 3.0 provider, and I don't want things to get messed up, so I'm pinning it to 2.44.0 because I know it works. We're setting the name of our VPC to primary. We're setting the cidr to the value stored in our cidr_block.

For availability zones, there's some interesting stuff happening here with our function. So we're using a function called slice. Slice takes a list and slices off a piece of that list for you to use. The first thing that slice needs is a list to slice information off, and we are getting a list of availability zone names, so these are the names of all the availability zones in Then we're starting with the first element, which is numbered 0 in that list, and then the last value is where to cut off the list, Remember, our subnet_count is 2, so we will get the first and second element from the availability zones list. Down in private_subnets we are passing it the two private subnet IP ranges we want to use the. Same thing for public_subnets. I have enable_nat_gateway set to false. Now in a situation, you might have this enabled, but there's two things about the NAT gateway. One, it takes a long time to spin up, and two, it costs a lot of money. So I figured I would save you money and also not delay the process by having it disabled. We are not creating a database subnet group, so that's set to false. And then finally, we are adding some tags to all the resources that are created by this module. We're setting Environment = "Production" and Team = "Network". So, that's everything in resources. We can go over to outputs, and we can see there's no outputs being generated by this whole process, so now we are ready to deploy our Terraform configuration.

Chapter 2 How to Deploy the Network Configuration

We have reviewed our Terraform configuration, and now we're ready to deploy it. So let's go ahead and pull up the command prompt again. First we're going to run terraform init, and if you'll remember, terraform init does a few things.

It downloads any chapters you're using, it downloads any provider plugins you're using, and it initializes the back end to store your state data.

Once that's complete, we can run terraform validate, and terraform validate validates the syntax that's in your Terraform configuration. You actually have to run init first because it validates the syntax based off of what's in the provider, so it needs that provider locally to evaluate the syntax and the resources from that provider.

It looks like everything is valid, so that's thumbs up, that's good. Now we're going to do terraform plan and store our plan in m3.tfplan. Now you don't have to do this, but it is best practice to run plan first and send that to an out file, which you'll then use with terraform apply.

When you run terraform apply using a tfplan file, Terraform will check and make sure that that plan is still valid. That's really important when you're running in a collaborative setting. We're going to go ahead and run terraform apply and pass it that plan file, and that's going to go out and provision the VPC in AWS. That should happen pretty quickly because, we're not creating those NAT gateways, and that's the thing that actually takes the most time.

It added 14 resources. If we want to go over to the AWS Console, we can review what has been created so far. Here we are in the AWS VPC Management Console. And , we now have our primary VPC, and it's using the CIDR range that we would expect.

If we go over to Subnets, we can filter on the VPC, and here are our four subnets that we have created. We've got our two public and our two private. So that's all deployed, and we are good to go. Now that we have our basic environment set up, a request has come in that we add a public and private subnet to the availability zone. We decided to farm this workout to Johnny, our junior admin. Now unfortunately, we didn't check with Johnny ahead of time to let him know that Terraform is the thing we're using, so he's just gone ahead and run a script to create the public and private subnets in that availability zone. But that's not good. Now we need to get those subnets under management of Terraform. And the way that we're going to do that is by using the import command. So let's talk about the import command. Unfortunately, there's no automatic import. You have to update your configuration within Terraform and then manually import the resources you want Terraform to manage. That's just how it works today. Step one in the process is to update your configuration to include the resources you want to import, and we're going to see how that works in a moment. Then, you have to match up the identifiers from your provider with the identifiers within your configuration, which that sounds a little confusing, but once you see it, it'll make sense. And then you add those new resources or the resources that you're bringing under management into your state data. Let's take a look at the import command itself to get an idea of what that flow looks like. So the import command syntax simply goes terraform import, whatever options, the ADDR, and the ID.

What's that ADDR and ID? What do those mean? so the address is the configuration resource identifier. It's the way that Terraform is identifying that resource inside its own configuration. So an example is if we're using a module, there might be a resource that is one of the subnets. And so the full address identifier for that is module.vpc.aws_subnet.public[2] because it's the third element in a list of subnets. That's the address. The ID is the specific resource identifier for that resource type, and that's something you can look up in the resource information in the Terraform docs. You can go and see what is the actual identifier for this resource if I want to import it. It's usually at the bottom of the description of a resource. For example, in the world of AWS and subnets, it's the unique identifier for that subnet. Once you have those two pieces of information, you can now run the import command, and that would be something along the lines of terraform import. You can pass it the variable file for your configuration, then that address of the resource inside your configuration, and finally, the resource identifier of the actual resource from the provider. So let's go back to our demonstration environment. We're going to create that resource using the script, and then we'll get it imported into our Terraform configuration. Back in Visual Studio Code, the next step in our process is to run the script that will manually create those two subnets. I have two scripts here, one for Linux and macOS that uses the AWS CLI, and the second one, which uses the AWS PowerShell module to create those subnets.

So you can choose whichever script makes more sense for you. Since I'm running on Windows, I'm going to use PowerShell. If you don't already have the AWS PowerShell chapter installed, assuming you're running PowerShell 7, you can install by running AWSPowerShell.NetCore. And to avoid any issues with running in an admin context, you can set the Scope to CurrentUser and it'll just install it for you. It's a lot easier than opening an admin window. And then you're simply going to run the JuniorAdminIssue.ps1 script. So let's go ahead and open up our PowerShell terminal, and we are going to run this JuniorAdminIssue script. Remember that we set the environment variable for the AWS profile to deep dive. So whether you're doing this through macOS, Linux, or you're doing this through PowerShell with Windows, it's going to know to use that profile as the credentials for creating these resources in AWS.

It ran successfully. And it actually does a little bit more than create those subnets. It actually creates the route table association between the public subnet and the public route table, as well as creating a new route table for the private subnet and creating a route table association for that private subnet. And I've helpfully dumped out the values that are the identifiers you'll need for the import command. So if we go over to the AWS console and refresh our view of the subnets, we can see there's two subnets that weren't there before.

And because it was created manually with this script, they're also not tagged appropriately with the correct naming. That's not good but we can fix this. That's no problem. We are going to do that by using the import command. So let's go back to Visual Studio and get that process going. In order to add these subnets, and route tables, and route table associations to our configuration, we have to make some changes to our tfvars file. So let's go ahead and open that first. And I already have the updated values . We're going to uncomment these other values for private_subnets, public_subnets, and subnet_count. So now we have the third private and public subnet, and our subnet_count is set to 3, and we'll go ahead and comment out the values that we had before.

So now our values have been updated to have three subnets. And the next time we run terraform plan, it's going to plan to create these resources. Let's go ahead and open up our terminal window, and we're going to run terraform plan and specify the output, m3.tfplan. The reason we're doing this is because we need to get the address for these new resources so we can use it with the import command. So we're not actually going to apply this plan.

We just needed the information in it. So, let me expand the terminal all the way up, and we can scroll up and look at some of the things it's added. And right here, we can see one of the things it's adding is chapter.vpc.aws_subnet.public[2].

This is one of the new resources that's being created. So now we know that address that's going to be used by Terraform, and we can use that in the import command instead of applying this plan. I went through the plan and grabbed all the addresses and put them in an import command one by one. So if we look at what we have here, we have terraform import, we have the variable file being specified, and then we have the address of our resource.

The other thing we need is the resource IDs that popped out when we ran the script. So let's go ahead and scroll up here, and here's the resources that we need, and they should be in the same order that the commands are. So we've got our privateRouteTable. So go ahead and take this privateRouteTable value, and I'll paste it. The route table association is a resource within Terraform, but it's not actually a separate object in the world of AWS. So the workaround is that you have to create the ID from the subnet and the route table. So the way that you do that is first you grab the subnet, and we'll paste that , and then you put a slash, and you grab the route table ID and put that . And that's the identifier in AWS that matches to the resource address that's in Terraform.

So now that we have that, we can paste in our private subnet value here. And if we scroll down a little bit, we can get to our publicRouteTableAssociation. And if you look at that, you see it's the same thing. It's the public subnet, and then slash, and the route table for the public subnets. And then finally, we're going to grab the public subnet ID and paste it . So now we have all of our import commands ready to go.

Let's go ahead and copy all of those import commands, and let me expand this out so you can see what's happening, and I'm going to go ahead and paste them in. So it's going to run the terraform import command, and it's going to say Import prepared!

Now once these have all finished, we can go back and take a look at what's actually in there. all of the import commands have completed successfully. So if we look at what happened in one of the import commands, it's basically saying I'm importing this ID with the subnet, in this case. The import is prepared, and it refreshes the state data with this new import information and lets you know that the import is successful. So basically, within the state data, it has this new resource, and it has matched it to this new ID in AWS. Let's go back to our commands list, and if we scroll down, we can see we have successfully run the import commands in ImportCommands.txt, so the last thing to do is run terraform plan and apply.

Let's see if there are any changes when we run terraform plan again. We have run terraform plan again, and there are some changes.

Nothing to be added, nothing to be destroyed, but there are a few changes, and those changes are those tags. Because the subnets were created without the proper tagging, Terraform is now going to apply those tags in the consistent way that ACME was expecting. That's exactly what we want, so we'll go ahead and run terraform apply. And obviously that scrolled by very quickly because all it's doing is updating some tags.

It doesn't have to do anything beyond that. If we go back to the AWS Console and go ahead and refresh our view of the subnets, we can now see that all of the subnets are appropriately named.

And if we select the public subnet in the 1c availability zone and go to Tags, we can see it has all the appropriate tags applied to it. So we have fixed Johnny's mistake, and we'll take him aside and show him the wonderful ways of Terraform later. In summary, this chapter was all about importing resources because Brownfield is really the norm when it comes to Terraform. You're going to walk into an environment that already has resources deployed and we saw how we can fix that for just a few subnets. In a more complicated situation, you're going to need a more robust configuration and you might not want to create and run all of those import commands manually. You might want to script out that process. Import requires config work. We saw that. We had to update our configuration before we could import things, and for now, there is no automatic way to detect the resources, not under management and bring them under management. The nice thing about it is when you do have to create that configuration, it gives you an opportunity to really learn what's in your infrastructure because you may have made assumptions about what's there or how things are set up. When you actually have to map that all out in code, you end up learning a lot about how your infrastructure is actually deployed. Coming up in the next chapter, we are going to get into managing state data. We just worked a bit with state data when it came to how we imported resources, maybe it's time to take a deeper dive into what's actually in that state data and how you manage it by yourself and in a group.

Chapter 3 Terraform State Commands & Backends

What holds Terraform together is state. State data is what maps what's inside your configuration to the resources that live in the public cloud provider or whatever you're deploying your infrastructure to. State is critical. So this chapter is going to be all about managing state in Terraform. First, we're going to explore state data, and we're going to use the command line to do that state data exploration. What can you uncover about the state and how can you interact with it without breaking it? Then we're going to look at what are the options for storing your state data? There is the default option, the local file, but it goes well beyond that, and you probably shouldn't store your state data locally if you really care about it. And finally, we'll look at the process for migrating your state data from that local file to some sort of remote back end, and the same process works if you're moving from one remote back end to another. Before we get into state data exploration, let's talk about what's going on at ACME. At ACME, you are now going to be working with a larger team. What does that mean? Well, you have your internal networking team that you're already working with, but now they're coming on board with your Terraform project. You need to collaborate with that team. You're also going to be creating infrastructure for other teams to consume. You're not just building VPCs for the fun of it, you're doing this for application teams who have applications and servers that they want to get deployed, so they need to be able to share the information about the infrastructure that you're creating. And we're going to enable this collaboration with your team and the other teams through remote states, so we're going to figure out how to get that implemented, but we don't want them to be able to muck around with our remote state, especially those application teams, they might mess up our networking so we want to restrict the level of access for those teams to just read only, you can read our state, but don't change it. So that's what's going on at ACME. If you'll remember, the current environment that we have deployed in is composed of 3 availability zones and each availability zone has a public and private subnet in it. The public subnets are starting at 10.0.0.0/24 and incrementing up by 1 for each, and the private subnets are starting at 10.0.10.0 and incrementing by 1 for those. Building out this design, we did this all locally and we have a local state file, that could present a problem for our collaboration ideal. So we need to get to a point where our state data is living somewhere else, but first, we need to understand what's going on in the state data to begin with. Let's talk brass tacks here, what is Terraform state? It's data that's stored in a JSON format, and it's very important that you don't go into wherever your state is stored and start messing around with the JSON. JSON is pretty easy to break, and it's pretty easy to mess up what's going on in that state data, because what's in that state data? Well, it's mostly resource mappings. So if you remember from the previous chapter where we dealt with importing something into state, you'll remember we needed the ID of the object inside our configuration and the address of that object in the real world. The state data is a mapping between those two items, along with some additional metadata. Some of that metadata is about the resource itself and some of that metadata is about the configuration and the current deployment. We can inspect the state data without messing it up by doing it through the CLI, and we'll step through a few of the commands in a moment. When you run certain operations, like plan, apply or even refresh, it actually goes out and refreshes the state based off the objects that the state represents. It will go out to, say, AWS and just pull your VPC to see if what it has in the state file still matches up to what's going on in the outside. It's not going to add new resources, as we saw in the import chapter, but it will update existing resources. But, where is this state data stored? Well, it's stored in back ends. There's roughly two kinds of back ends. There's standard and enhanced. The only enhanced back ends are Terraform Cloud/Enterprise and the local file enhanced back end, and what makes it enhanced is it not only stores the state data, but it also runs the Terraform CLI commands. Two other important distinctions are whether or not the back end will support locking and workspaces. We'll see what locking is a little bit later in this chapter, and there's a whole separate chapter dealing with workspaces, but it's important to note that not all back ends support these two features, and they're very useful, so be on the watch for that. Now let's take a look at what those terraform state commands are. There are a bunch of terraform state commands, and they all start with terraform state, so I'm going to skip that portion of the command and just get to the subcommand.

The first one is list, and this simply lists and enumerates all the objects that are stored in state data. This is all of your resources. There's another one called show. Show will give you details about a specific object. So a general workflow might be to run list first to get the names of all of the different objects in your state data, and then run show to get the properties about a specific object. Another one is mv, which is short for move, and move allows you to move an item or an object within the state data or to a different state data. This could be if you wanted to rename the ID of an object inside the configuration, you wanted to change the name of it somehow. Normally that would require destroying and recreating the object, but you can actually update your configuration and then use move to simply change the name. It's sort of like renaming a file. You can also use the command rm, short for remove, to remove an item from state. So if you remove something from the configuration, but you don't want that object to be destroyed, you could run rm to remove it from state so Terraform is no longer managing that item for whatever reason. Another command is pull, and this simply will display the current state data in its JSON format directly to stdout. So if you have something else that typically consumes that state data for some reason, maybe it has to go to an audit log or something, this is a command to pull that information in a graceful way. And finally, the last one is push. If, somehow, you wanted to push a local copy of your state data up to remote state, you could do that. There's not many occasions you would want to do this, but if you needed to, if the remote state was somehow corrupt and you had a good local copy of it, you could use push to overwrite the remote state. So let’s jump over to Visual Studio Code and investigate some of these different commands. In Visual Studio

Code, and I am going to expand m4 in our chapters, and I'm going to open the m4_commands.txt file. So first, we're going to try out some of our Terraform commands, so let's go ahead and open up the command prompt. And I have already deployed the VPC in the previous chapter and gone through the whole import process, so my VPC matches what we saw in the earlier diagram. So let's go into that directory, and the first command that we can run is terraform state list. This will list out all the objects or resources within our configuration.

We can see it's got things like the internet gateway, the route tables, the route table associations, our various subnets, public and private, and the VPC itself. That's what we would expect. Those are the resources we have deployed. What if we wanted to get more information about a particular item in this list? Well, that's when we get to terraform state show. Tthen the ID of the object that we're interested in. In this case, we're interested in the VPC itself, so we can get that syntax from the list and then plug it into here. Let's grab this command and it will now print out all the information it has about our VPC that's stored inside that state data.

So we can scroll up and see all the information. In case we wanted to reference one of the attributes, here's where you can also get all the attribute names about that resource in case you needed to reference them. We can also pull all of the state data by doing terraform state pull. This is going to be a lot of information. This is the entire JSON file, not very useful for you to try to parse through on your own, and that's why we have these other commands, but it is available if you wanted to pump this out to some sort of tool. You just don't want to parse through this manually. All of this state data is stored locally. If we expand m3 here, we can see there's a file, terraform.tfstate, and I'm going to open it. This is the state data.

It has some relevant information at the top, the version, the Terraform version that was used to create it, the serial number, and the lineage, and that allows it to track some information about what version of Terraform created this thing, and it won't let you use an older version, and also what the lineage is. In case you try to override it with an earlier lineage, it's going to say no, your stuff is out of date, you need to refresh your stuff before I would allow you to plan and apply something. but like I said, Terraform state is delicate. Don't go in and edit it unless you know exactly what you're doing. now we want to move this Terraform state, so let's talk about back ends.

Backends are where state data is stored, that's all they are, that's what they do. When you decide that you want to use a particular backend, it must be initialized. If you'll remember when we run terraform init at the beginning of a configuration, one of the things it says is initializing the backend, and if you haven't specified any particular backend, it's going to use the local file backend by default. Since it doesn't need any more information, it can go ahead and just configure that on its own. You don't have to put a configuration block in. If you do want to put a configuration block in, generally speaking, partial configurations are recommended. Why do I say that? We'll see what the syntax is in a moment, but basically, in order to write data to a remote location, you usually need credentials, and those credentials should not be hardcoded in the configuration. So you want to supply those at runtime, and you do that by only giving the bare minimum configuration to point it at the right location. Another thing that is important to note is that interpolation is not supported. What does that mean? It means you can't use variables or local values in your backend configuration, and the reason is Terraform hasn't even evaluated your variables and locals when it initializes that backend. It doesn't have access to that information. So the only way to give it dynamic information is to supply it at runtime. Let's look at an example backend, and hopefully this will become a little more clear.

A basic backend configuration would start like this. You have your terraform block, that's how it starts out, and then within that terraform block, you're going to have a nested block called backend. And after the backend block type, you're going to put the type of backend that you plan to use and then some information about that backend to allow Terraform to connect to it properly. Now what kinds of backends are there out there? There's a bunch of them. There's Consul, which is a HashiCorp product, AWS S3, which actually needs to be used in tandem with DynamoDB to support locking in workspaces, there's Azure Storage and Google Cloud Storage. So if you're using one of the major public clouds, you can use that as a target for your remote state. But what's ACME planning to do? They're planning to use Consul. ACME, while they've completely invested in AWS for their cloud, is not sure they want to pin themselves down using S3 as the repository for their state data. Instead, they want to work with another HashiCorp product called Consul. Consul can do a lot of different things and one of those things is work as a key value store, and since JSON data can be stored as a value with a key, it can be used as a remote backend for Terraform. So the plan at ACME is to add a path called networking in the Consul key value store, and within that networking path, add a state path where we will store all of our different state configurations as a key on that path so networking state will be the path. But, how are you going to interact with that? Well you're going to do this through your local workstation. Consul supports access control and the way that it enforces access control is through tokens. So we are going to get a token and then supply that token as part of our authentication to write and read data from Consul. We're also going to apply some policies to Consul that restricts who's able to read and write data at that networking state path. So let's go back to Visual Studio Code and we'll walk through the process of setting up Consul. We're back in Visual Studio Code, and it is time to deploy our local Consul server node. We're going to be running the Consul service locally rather than spinning up a virtual machine or a container. It's pretty simple to do, and the executable doesn't take up a whole lot of room. If you don't have the Consul executable, you can simply go to consul.io/downloads and download the correct version for your operating system. Once you've done that and added it to your path, we can go into the Consul folder and walk through the process of getting it deployed. We can take a look at what's in the consul directory here.

We have a configuration directory, and that has the in HashiCorp configuration language. Basically all the config is doing is setting up access control and granting you access to the UI. So there's an argument here, ui = true, that means enable the UI so we can look at what's going on in Consul. We're setting the bootstrap_expect to 1, so there's only one node in our Consul cluster. We're setting the data directory to /data. That's where it's going to store its key value data, as well as its other information. And we're setting the ACL property to enabled. So we're enabling ACLs for our deployment. There's a lot more to Consul than we're going to talk about in this book. Let's go back to m4_commands and we are in the consul subfolder. We're going to make that data directory, so we've done that. Now that data directory exists and we're going to launch a consul server instance with all of these different arguments, basically giving it the as its configuration and binding it to our local IP address.

When this launches, it launches in the foreground, so we're going to have to open a separate terminal to execute the remainder of our commands. Let's go ahead and open up this other terminal window. And what's the next thing we need to do? Well, we enabled ACLs on Consul, and we need to get sort of a root token to be able to do anything. So, this command, consul acl bootstrap, does exactly that. We can go ahead and grab that command down here. And let's go back into the chapter 4 directory just to make sure we're in the right place.

And we're going to go ahead and run this command. so we ran the command and it returned a bunch of information. The important thing here is that SecretID. And we are going to set an environment variable, CONSUL_HTTP_TOKEN, to that SecretID. I'm going to go ahead and grab that out of the output and copy it. And since I'm running in Windows, I'm going to paste it under the Windows direction. If you're running macOS or Linux, you can use the export command for this. Let's go ahead and copy this and paste it.

Now that environment variable is set every time I use the Consul command line, or if I use the Terraform Consul back end, it will know to look for this environment variable and use the value that's stored here. Now we need to set up some information inside Consul so it can store our remote state, and we're going to do that by using Terraform. So let's go ahead and open up this main.tf file over here. This is the additional configuration information we're going to use for Consul. So first, we have to reference the Consul provider. And because we use the local address, I've just hard coded it using the local loopback address in port 8500.

That's the default port for Consul, and the default datacenter name is dc1. Now we're going to create a few resources inside Consul. We're going to create some key paths. So , I have a resource called consul_keys, and all this does is create key paths for us. The first path we want to create is networking/configuration.

We will use that to hold configuration data in the next chapter. Then, we're going to create networking/state, and that is where our state data is going to be stored. Scrolling down a little bit more, we're going to bring in the applications team on this game at some point, so we're going to create the same paths for them, applications/configuration and applications/state.

And scrolling down a little bit more, we are going to create ACL policies for networking and applications.

The networking policy is going to grant write permissions to the key prefix "networking" and also grant us access to the session prefix. That allows us to establish a session with Consul and then have write access to that networking key prefix, so anything networking/, we have write access to. The second policy is for applications, and it's a little bit different.

For the role, we're granting write access to the key_prefix "applications", so that's basically the same. We're just doing applications instead of networking. The second one is a little more interesting. We're granting read access to the key_prefix "networking/state". That means that the applications team, if they have this policy, they'll have read access to the networking state data. That will be useful later. And then we're also granting write access to session_prefix so they can establish a session. And then scrolling down a little bit more, we are going to create two ACL tokens, one for our networking administrator, and one for our applications person. So we are going to create a token for each.

And if we scroll down a little bit more, we get to the outputs. We're going to output the accessor ID for both of those tokens, and then we can use the Consul command line to retrieve the secret ID that's associated with that accessor ID. It's a lot of information, but it's pretty straightforward. We're creating some key prefixes, we're creating some policies, and we're creating some tokens. That's basically the view of what we're doing, so let's go ahead and do that. We've reviewed our Terraform configuration for Consul, so let's go ahead and make it a reality. First, we will go into the Consul directory, since that is where are configuration is, and go ahead and run terraform init. Terraform init executed successfully.

Now we'll run terraform plan, and we'll output it to consul.tfplan, and this should go very fast because this is all running locally.

And finally, we'll run terraform apply consul.tfplan to create all these resources. And we got our two accessor IDs.

I would recommend grabbing one or both of these and pasting them somewhere else, because we're going to need both of those later. So I'm simply going to run Notepad here and paste that accessor ID for offscreen. And now let's actually get the token secret ID by pasting in that accessor ID for Mary, go ahead and paste that , and we'll grab this whole command, consul acl token read, and passing it that accessor ID, it's going to give us her secret ID. So I'd recommend grabbing that secret ID and also storing that in Notepad somewhere because we're going to use that one later. Before we do anything else, let's take a look at the Consul UI and just see what's going on there. So I just copied the root token that we've been using.

Let's go over to a browser, and I'm going to open a new tab here, and I'm going to go to 127.0.0.1:8500/ui/. And Consul should load for you. Now, we have to log in to Consul by going to the ACL and pasting in that secret ID for the root secret. And now we can see all the access controls we put in place, including the two tokens that have been issued. We can go to Policies and see the two policies we created. We can go into Key/Value and see the key/value pass for applications and networking. And if we go into networking, we can see configuration and state. But of course, there's nothing in there right now, because the next thing we need to do is migrate our existing state from the local file to this remote back end. Migrating Terraform State is remarkably simple. It is not complicated at all. The first thing you need to do is update your backend configuration. You have to tell it, as we saw in the example, where to put the data in the new backend, and once you've updated the configuration, you have to rerun Terraform in it. You're reinitializing your configuration and it will see, oh there is a new backend here, and what it's going to do when it sees that new backend configuration is ask you, do you want to migrate your existing state data to this new location? And generally speaking, the answer is going to be yes. So we simply have to confirm that state migration, it copies all the data from local to remote and you're done. It is a little anticlimactic, but that is how it works. So why don't we go back to Visual Studio Code and walk through the process. In Visual Studio Code, and we need to first update the backend configuration. So let's go up one folder, and you'll note in the m4 folder, there is a backend.tf file. We are going to copy that backend.tf file into the chapter 3 folder to update things. So go ahead and run this copy command. That file's now in m3, and let's move to the m3 folder. And over here, I'm going to expand m3, and let's take a look at what's in that backend.tf file.

This is the backend.tf file, and we have our terraform configuration block, and within there we have our nested block for the backend config. The backend type is consul, so we have backend "consul". The address for the consul server is 127.0.0.1, port 8500, and the scheme is http. HTTPS is supported, but we didn't add that to our consul config.

That's all the information we're giving it, and we might not even give it this much information if we were deploying to a production scenario and we don't know that the address is always going to stay the same. We could supply all of that information at the command line when we run terraform init. Speaking of which, now that we've copied that backend config in, we can go back to m4 commands, and what are we going to do? We are going to use secret ID to write this state data. We went through all the trouble of setting up the ACLs and getting her a token, we might as well use it. So I'm going to go ahead and copy the secret ID that I grabbed earlier, and I'm going to paste it . And all we're doing really is updating our environment variable from the root token to token. If you're running Linux or macOS, use the export command, but I'm running Windows, so I'm going to set the environment variable via PowerShell. My token has now been updated. Now we can initialize the backend config, and the command for that, it's terraform init. And if you want to specify backend config data during the initialization process, you can do that with the argument and you can either supply a file that has all the backend config settings in it, especially if you have a lot of them, or you can just do pairs and you can specify this argument a bunch of times. In our case, the only thing we need to specify is the path that we want to use here, so we're setting the path equal to networking/state/primary. primary will be the key where our JSON data will be stored as the value. We don't have to specify any authentication information because we're storing that in our environment variable. Now let's scroll up a little bit and see what it's saying. So it initialized modules. There's nothing new to do there because we've already initialized our module, and then it went to initializing backend, and it saw, hey, you're using a new consul backend. That's exciting. Do you want to move the local backend information to the consul backend? Because it didn't find any existing state data already there. And all we have to do is simply say yes, and it's going to copy that information over, and that was really quick. Since we're copying locally, it's not going to take very long to copy that state data. We can scroll up and you can see it says successfully configured the backend "consul". And it's going to automatically use this backend from here on out. If we scroll up here in the left side and we take a look at what's going on in terraform.tfstate file, there is nothing now because all of our state data has been moved, so that file is now blank. Where did that information go? It went to consul. So let's go back to the browser, and if we refresh our view here, we'll see there's now a primary key. And if we look at that key, here is all of our state data in JSON stored as the value for that key.

So this is where our state data lives from now on. So we have successfully migrated our state data from a local file to a remote consul backend.

Technically it's local, but you could do this exact same process with a consul cluster that's running in your data center or up in the cloud. In summary we established that Terraform state, it's kind of important, it's kind of a big deal. You do want to be careful with it and you want to put it in a place that's protected and accessible to other team members. We saw how we can manage our state through the CLI. So you don't want to go in and touch the JSON data. Do it through the CLI, because it knows what it's doing. Generally speaking, storing your state data remotely is preferred. It makes it safer, if your desktop or laptop fails, you're not going to lose all your state data, and you can collaborate with other people. Next, we are going to get into the wonderful world of data sources and templates. We are going to use console as a data source and not just for remote state data, and we're going to see how we can use templates to streamline our configurations a little bit.

Chapter 4 How to Use Data Sources & Templates

When you're using Terraform, you tend to focus on resources because you're excited about creating cool infrastructure, but there is another key component within your configuration and that is data sources. So that is one of the things we're going to investigate in this chapter and the other one is using templates to streamline your configuration. In this chapter, we are going to start out with talking about data sources types. There is a bunch of different data source types out there and it's important to understand the reason for using each one so we're going to get into that a bit. We're also going to look at how we can use an external source for our configuration data. Rather than using a local tfvars file or specifying those variables at runtime, what if we could reference an existing data source for that configuration data? And finally, we'll talk about templates because they are something that will help you write more efficient configurations. But, what's going on over at ACME? Well, as we saw in the previous chapter, more teams are getting on board with this Terraform thing and let's talk about who those teams are. Well we've got our information security team. What do they want out of Terraform? They want to define roles, policies, and groups consistently within AWS and also within Consul, so they are very interested in the template aspect of thing and we're going to see how that applies in a later chapter. The software development team also wants to get involved because they want to deploy their applications on top of all this awesome network infrastructure that you've been deploying, so they want to be able to read your network configuration for app deployment and that's something that we set up in the previous chapter. They do have access to our remote state data. Lastly, the change management team, they would like you to store your configuration data centrally so they can quickly reference that configuration data and go, does that match up to what you said was going to be in the change from a week ago? Good, it does. We have a record of what should be in the configuration and what's actually been deployed. It's a whole solution and change management loves it. How are we going to accomplish what change management is doing? That's what we're going to do in this chapter and we're going to do it through data sources. What are data sources? Well, you can think of them as the glue for multiple configurations and within configurations themselves. We've already seen data sources being used to get the list of availability zones for AWS, but there's much more to it than just that. Resources can be thought of as data sources for other resources inside the same configuration or within other configurations. When you create a resource, it has a bunch of attributes that can be referenced within your configuration, so you can really think about those attributes that are exported as a data source. Providers also have data sources and we saw this with availability zones, but there's plenty of other really useful data sources that exist in the provider, and if you look in the documentation, the documentation now breaks up the provider as data sources and resource types. Within those data sources, you might have something like the AWS AMI for Ubuntu or for Amazon Linux. If you needed to get that ID for the region you're currently working in, data sources are going to be your friend.

There's also alternate data sources and that could be things like templates. Templates are actually considered a data source. You can also reference a website or web service that uses HTTP, so it will make a request to a site and then get a response in text or in JSON. You can also use an external data source, and this basically runs a script, and that script has to return valid JSON to the standard int. As long as it does that, Terraform doesn't really care what that external data source is, so a lot of latitude there. Another alternate data source and one that we're going to use in this chapter is Consul. You can pull key information from Consul and use that within your existing configuration. Let's take a look at a few example data sources. We'll start with the HTTP data source. An example of that would simply be first using the data keyword. So data lets you know that you're creating a data source within your configuration and the type is HTTP, and then you have to give it a name label, in this case we'll call it my_ip. Within the body of this configuration, you can specify the URL, where should it go to make this HTTP request?

The example I have here, http:// ifconfig.me, will return just your public IP address in the body of the response, so you can directly use that response to get your public IP address for your Terraform config, which is sometimes useful if you're trying to create security rules. The way that you would reference that is by doing data.http.my_ip and then the attribute is body, and this will get the body of the response, which in our case would be just our public IP address. What about the Consul data source, what does that look like? There's more than one data source within Consul.

You can get a key prefix, you can get individual keys, you can get policies, you can get tokens, there's a lot of options here. In our case, let's just look at Consul keys. So we want to get one or more keys out of Consul and the value stored in those keys. Within that data source, you'll have multiple nested blocks for each key that you want to retrieve. So let's say we had a key at the path networking/config/vpc/cidr_range. We can give it the name vpc_cidr_range, and that's how we'll refer to it within our Terraform configuration. If that value is not found, you can specify a default value for this key, and in our case we could set it to 10.0.0.0/16. If you want to use the response from this data source, you can reference this specific key by doing data.consul_keys.networking.var, so var is what says I want to reference one of the keys that was in my data source, and then which key do you want, you give it the name of that key, which in our case was vpc_cidr_range. And that's how you get the value that's stored in that key, and that is exactly what we're going to do for our configuration data for our VPC deployment. Let's look at how that setup will go. If you'll recall our Consul setup from the previous chapter, we had Consul set up and running locally, and we've added some things to the key value store within Consul. We added a networking path, and within that networking path, we added a state path, and then added primary as a key in there to hold our state data. Now we're going to add a second path within networking called config, and that is where we're going to store our configuration data. We also created an applications path, and within an applications path, we also are going to have a state path and a config path where we'll store the applications state data and config data. Now we have our cast of characters. The policies that we set up in the previous chapter gave read/write access to the networking path and gave Samantha read/write access to the applications path. It also gave Samantha read access, but only to networking/state so that she could read state data out of that networking and use it as a data source. Let's go over to Visual Studio Code, and we're going to populate the configuration data for our setup and migrate our configuration over to use the config data that we're going to store in Consul. In Visual Studio Code, let's go ahead and expand m5. How are we going to start out here? The first thing we're going to do is go into that consul subfolder. And what's in that consul subfolder? We've got three JSON files here that have configuration data in them. Before we take a look at that, we need to set up our consul token environment variable so that we can interact with Consul. And we can do that by either using the export command if you're running Linux or macOS, or in Windows and PowerShell, you can do $env: the name of the environment variable and then the secret value. I have secret ID in a Notepad document, so I'll go ahead and copy that data, and go ahead and paste it .

And now I'll copy this whole thing, and paste it. I still have Consul running from the previous chapter. , and see it's still running in the background. If you had to shut it down for some reason, you can restart it with the same Consul command that was in the chapter for commands. Now that I have my token set, I should be able to run Consul commands and have them execute successfully. What are we going to do? We're going to run two Consul commands here, we're going to run consul kv put, so we're putting a value to the KV store. We have to give it the path where we want to put that data, so we're doing networking/configuration/primary/net_info and then in order to write all the data that's inside a file, you use the @ symbol and then the name of the file. One important thing to note here, your file cannot have an underscore in the file name or the command doesn't quite work right. And we've successfully written that data to that path. We're going to do the same thing again to a path that's called and write the data that's stored in to it. We've written the data to So let's take a look at what's in primary.json first. We have JSON data that defines most of the information that we had in our variables before.

We had the cidr_block for our VPC, that's . We had a list of address ranges for our private and public subnets, and we had our subnet count. That's all now stored in JSON and we can refer to that within our configuration. The simply has a list of common tags. So now your change management or your security group could update to the list within Consul, and when you run an update for your environment, those new tags will be added. So now we need to update our configuration to take advantage of this Consul config, let's take a look at how we do that. We've written our configuration data to Consul, and we're ready to use it, but first, we have to update our Terraform configuration. And I have an updated configuration in the networking subfolder. So let's go up one level and go into the networking subfolder. That's where the configuration is. And for the moment, I'm going to hide the terminal so we can take a look at what's in this networking configuration.

I simply copied over the existing configuration that we were using in chapter 3, but made some changes to it. First change I made is in variables. So if we look at our variables here, instead of having all the network data, now I have information about where the Consul server is running and what ports and datacenter to use. That's defined in the variables. I have the proper defaults, but obviously, you could set this to wherever you happen to be running your Consul server. The next big change is in resources. So let's go ahead and look at the resource here.

I have a provider defined here for Consul that uses the variable values that were in the variables file. So that is all set and ready to go. If we scroll down a little bit, we get into the data sources, and here, I've defined a data source. I'm using the data source, consul_keys. I'm calling it networking, and then the path is the path to my configuration data, networking/configuration/primary/net_info. Now what am I going to do with this information?

Because remember, it's stored in JSON. It has a bunch of data in there. Well, if we scroll down a little bit further, I'm going to find some local values to make use of this data. That makes it easier to use locals where I was using variables before because I just have to change var to local.

So I'm basically defining the variables I had before, but in my local values. And the way that I'm doing that is first, the data is in JSON, so I have to decode that JSON. How do I do that with Terraform? Luckily, there's a function called jsondecode. So I'm using that function, and I'm giving it the information that's stored within the value for the key networking. So I'm doing data.console_keys.networking.var.networking. Because the name that I gave to refer to it was networking. And then because it's JSON data and it's a map, essentially, I can refer to the key in the map of the value I want, in this case, cidr_block, and then I do the same thing for private_subnets, public_subnets, and subnet_count. And Terraform will correctly interpret the JSON data as the Terraform data types. So when it sees private_subnets and it sees a list of private subnets, it's going to know that that is a list data type and treat it as such. We've taken our configuration data, and we've defined it as local values. If we scroll down into networking and look at the cidr argument, now instead of var.cidr_block, I'm using local.cidr_block.

In the availability zone definition, I'm using local.subnet_count, and for the private and public subnets, I'm using the local.private subnets and local.public subnets. So far, I haven't updated the tags. I'm not quite ready to use that data yet. That is all the changes that we've made for this particular configuration. So the next thing to do is go through the Terraform initialization process. We have gone through our configuration, now we are ready to use our Terraform configuration, and what I'm going to do here is run terraform init with the pointing at Consul. Run that down in my terminal window. It's going to download the plugins and the chapter and check for the backend, and because there's already configuration data in that backend, we should be good to go. If we want to verify that the proper state data is loaded, we can borrow this command that we learned in the previous chapter, terraform state list, and it'll list out all of the objects in our state data, so we know that we're good, we're connected to that remote state backend that's in Consul. Now we can run terraform plan, and if everything goes smoothly, it's going to go ahead and tell us there are no changes to be made for our configuration, because we didn't actually change anything about the configuration, we just changed where it's getting its configuration data from.

No changes. Infrastructure is up to date. So that is exactly what we were looking for. Now let's take a look at how we can use templates to streamline our configuration a little bit more.

Templates are a way to work with strings within Terraform. There's a lot of different ways to work with strings, and templates are just one of those ways to do it. It's all about the manipulation of strings. Templates are a bit of an overloaded term when it comes to Terraform. What do I mean by that? Well, templates could refer to quoted strings. It could refer to using heredoc syntax in your config. It could be referring to the template provider, or it could be referring to the template file function. When you say templates in Terraform, you probably need to be a little bit more specific. What is important about templates is that they enable interpolation of values and also the use of directives. If you're not sure what that means, let's expand on that a little bit. We'll start with template strings. Template strings, you've already been using these implicitly. When you do an interpolation, you're basically using a template. They are expressed directly in the configuration. You're not referring to an external template file for this. You're doing the templating inside your own config. You can use heredoc syntax for readability. And if you don't know what heredoc syntax is, we will see that in a moment. What does a simple interpolation look like? Well, you've probably used this already. It's when you refer to a variable or a local value or something about your resource attributes and you want to use that in a string. You refer to it by using this interpolation syntax, the dollar sign, curly braces, and then the reference to the attribute or variable that you want to use. That's simple interpolation. You can also use a conditional directive and statements to determine the value for a particular string.

Let's say that you had a prefix variable and you wanted to append that prefix to app, but only if that prefix has a value. So you could do a directive statement that starts with the percent sign and curly braces, saying if the prefix is not equal to the empty string, set the string to the prefix.app, otherwise %{ else }, set it to and then end your if statement. So that's a conditional directive. You can also do a for loop with a directive where you can loop through a number of different names, let's say you had a list of local.names, and for each name in the list, you will create a string that is So that's a way that you could use a for loop. There's a bunch of other applications for it as well, but that is one example of how you would do that, and that is heredoc syntax, that less than, less than, some string in caps, and then you end it with that same string, and that means you don't have to use any escape characters within that body of text. Another way to use templates is by doing the syntax with the template data source. What does that look like?

If we're using the template_file data source we would start with template_file and then give it a name label, in this case we'll use example, and you can actually use the count meta argument within the data source, which is pretty handy if you want to create a number of these values. And then you have to define the template you want to use and this can either be defined or it could be defined as a separate file. If you're defining you have to use the double dollar sign to indicate an interpolation of a nested variable. Where is it getting the values for those nested variables? In the variables map block. So you're specifying a map of variables here, and you could specify some string for var1, and you can also reference the current count of the count loop that you're in for this data source by setting current count equal to count.index, and then this template would produce whatever is in the variable, some_string - the current_count. So that is one example of using an template, and we're going to use that in our configuration in a moment, so you'll see exactly how this works. In order to make use of the values stored inside the template, you would refer to the rendered attribute of that template. You can also define a template in a separate file, and this is especially useful for really long bodies of text, say something like an AWS IAM policy or role, those tend to be pretty verbose and long, and they would kind of clutter up your Terraform configuration a little bit.

So how does this work? It's a little bit different. In the template configuration, you still have your data source template_file, but for the template argument you use the file function and load what's in the peer_policy.txt file. And then, just like we did before, we have to specify any variables that we want to replace values within that text. In this case we might want to use the arn of the VPC to configure a peering policy that enables the actions AcceptVpcPeeringConnections and DescribeVpcPeeringConnections.

If we look over in the body of that text, you'll see under Resource we have ${vpc_arn}. Terraform will know that it has to use the value in var.vpc_arn as an interpolation for what's in the vpc_arn in the template file, and you will get the rendered template out of that. There's also a templatefile function, so if you don't want to use a data source, you just need that string, you can use the function, templatefile, and you pass it the path to the templatefile you want to use and then a map containing the variables you want to submit as part of that template. It's a little more streamlined and it means you don't have to use the template provider. So let's go back to our configuration and we'll walk through using templates to update our Terraform config. In Visual Studio Code, we are ready to update our config data to use templates, and we're going to update it to also use those default tags that we created in Consul as config data. so how are we going to do this? The first thing we're going to do is go into the consul folder, and we're going to run one command to put some new data in an existing key. We had our networking configuration data stored in network info, we're now going to update what's stored there by running consul kv put and referencing the file

Let's go ahead and grab this command, and we'll paste it down here. We've now overwritten the data that was previously the value for that key. What is in this file? There's a lot less than the previous file. So if we look at the previous file, we had our private_subnets and our public_subnets, our cidr_block, and our subnet_count.

If we're looking at our new configuration data, we're just giving the cidr_block and the subnet_count.

How are we going to get the rest of the information? Well, if you've noticed the public and private_subnets are based off of the subnet_count and the cidr_block, so we should be able to calculate that ourselves using some functions and templates. This sounds like a good idea, so let's close this out and we're going to go into the networking2 subfolder, so we'll go up one and into networking2. And in networking2, we have another configuration that's very similar to the one that we already looked at, but again, it's a little bit different, we have this templates.tf file.

Let's take a look at what's in there. Let me go ahead and hide the terminal here, and in templates.tf, we have two template file data sources here, one for the public_cidrsubnet, and one for the private_cidrsubnet. The count for both is being set to the subnet count. So we're creating three instances of the public_cidrsubnet and three instances of the private_cidrsubnet. How are we calculating what the value should be for each? We're using an inline template for the value, and within there we're using the function cidrsubnet. What cidrsubnet does is it takes a CIDR range and then it adds whatever number comes in the second argument to that CIDR range. So if we're doing a /16, we're adding 8 to it and making it a /24. And then the last one is which subnet out of that new range do you want to use? The first time we run this the vpc_cidr is going to be 10.0.0.0/16, we're adding 8 to make it 24, and the current_count will be 0, so it's going to use 10.0.0.0/24. Where is it getting these values? It's in this vars map. In the vars map, we specify the vpc_cidr, which we're getting from our configuration data using that local.cidr_block. And the current_count is coming from the count's we're referencing the index attributes of that. So this is going to run three times, it's going to create 10.0.0.0/24 and then 10.0.1.0/24 and then 10.0.2.0/24, exactly what we want for our public_subnet. For our private_cidrsubnet, we do the same thing except we have to add 10 to the values for our subnet ranges. So if you look down in the vars map, you'll see current_count is equal to count.index +10, so now we'll get the appropriate 10.0.10.0/24. etc. So we are calculating our subnets rather than statically defining them in a configuration. And that makes things a little more dynamic if you ever needed to change your core CIDR range, you wouldn't have to redo the configuration data for all of the subnets. That's kind of useful. But how is this working with the resources? Let's go ahead and open up resources here, and let's scroll down. And under consul_keys, you'll see we're now pulling two keys, we're pulling the network info as one key called networking, and we're grabbing the common tags values as a separate key giving it the name common_tags.

If we go down into locals, we can see we're grabbing the cidr_block and subnet_count from the map that's stored in the networking data. And for common_tags, we're just getting the entire value of common_tags, which is a map itself, it's a map of common_tags.

Scrolling down to resources, if we look under private and public_subnets, we can see how we are now using the template_file data source. Under private_subnets, we're saying data.template_file.private_cidrsubnet, and then we have this * character. What does that thing do? Well, what that star does is because we use the counts, we actually have three instances of this data source, that star grabs all three.

And then we tell it we want the rendered attribute of each of those three, and it will package it up in a list for us, which is precisely what we need for and then we do the same thing for public_subnets.

And then scrolling down a little bit more under tags, we've replaced the tags that we had with local.common_tags, so we're using the tag data that's stored in Consul. So let's go ahead and get this configuration deployed. We have reviewed our configuration and we are now ready to deploy that configuration. Go ahead and pull up the terminal window, and if we scroll down a little bit, we have our terraform initialization command. We are, once again, initializing our Terraform config and we're using the same Consul backend so it should just pull that state data in.

Once again, we can confirm that we have the proper state data loaded by running Terraform state list and we should get a list of all the resources. That's looking good. And now we can run our plan and there is going to be quite a few changes here, not because we're changing anything about our network configuration, but because we're updating the tags for all of our existing resources. So a lot of things are going to scroll by here and we can just scroll up a little bit once it finishes.

So in this one resource, the little yellow tilde lets you know what's being changed about a resource and we can see it's adding a bunch of tags to the existing list of tags for that resource and that's it. It's not making any other changes to that resource and you'll see that every other resource in the list, it's simply changing the tags. Our configurations worked out properly. If we scroll up a little bit and take a look at one of the subnets, we can see that the public subnet 10.0.2.0 hasn't changed at all, so our templates worked exactly as we wanted them to.

We have abstracted some of our configuration and now our tagging data is stored in Consul. That's pretty useful. So obviously, the last thing to do is to run terraform apply config.tfplan and we'll go out and update the tags for all of those resources, and obviously, since that's all it's changing, it happens pretty quickly.

We have successfully used templates to abstract some of our configuration and used a remote data source to hold our configuration data. In summary, one of the things we talked about is how templates enable code reuse. The way that we structured our templates in our configuration means if we change the cidr address that we want to use for a VPC, we don't have to change all the subnet values. That's something that will just automatically update, and that makes our configuration a little bit more reusable. You can do the same thing with IAM policies or the user data that you submit for EC2. There's lots of different applications for templates. The next thing that we saw was that data sources can glue configurations together. They can provide the configuration data. And as we'll see in a future chapter, you can use remote state as a data source for another configuration. Finally, we saw that custom data sources are an option. You can use external HTTP. You can use Consul. There are a bunch of different data sources out there, and you're not just limited to the existing providers and resources that you're using. Next, we are going to talk about workspaces and collaboration. So we are going to bring our applications team in and get multiple environments set up to collaborate on.

Chapter 5 How to Use Workspaces & Collaboration

When you're first getting started with Terraform, you're probably deploying something in a development type environment, but as your configurations move to a production type scenario, you're probably going to want to deploy multiple instances of your configuration, as well as collaborate with other team members and other teams. That's what we're going to explore in this chapter. We're going to start with using workspaces to create different environments to deploy the same configuration. And then, we're going to get into the idea of using remote state as a collaboration tool with other teams, and the way we're going to do that is by using remote state as a data source so you can pull information about another configuration's state data and use that as a data source in your configuration. But first, let's talk about the ACME environment. As we've discussed in previous chapters, you've become somewhat of a Terraform guru at ACME. And for that reason, you're going to be sharing your experience with a larger team, allowing multiple team members to contribute to creating configurations. That doesn't just mean your infrastructure for yourself. You're going to be creating infrastructure for other teams and enabling collaboration through the use of remote state and finally restrict access for other teams. That last part is key to what we're doing in this chapter. So in this chapter, we are going to meet the application team and get them set up to use our remote state data to deploy their application. What does our current environment look like? If you'll recall, our current environment configuration looks a little something like this. We have three availability zones running in a VPC in AWS. And in each availability zone, we've created a public and private subnet. That's what we've got so far. And that was a good start to get the configuration rolling. But now we need to support multiple environments. How are we going to go about supporting multiple environments with our single configuration? The way that we're going to do that is through workspaces. So imagine what we've got here. We've got our configuration that's spitting out infrastructure into AWS. And going into that configuration are values for different environments. We're going to create a workspace for each environment and then use our configuration data for each environment to spawn a development environment first, and then we're going to spawn a QA and UAT environment, and finally, spawn our production environment. We'll be doing this all through workspaces, which begs the question, what are Terraform workspaces? I am sure you'll be a little bit familiar with Terraform workspaces, but I think it's important to review now. Terraform workspaces let you take a common Terraform configuration. However, the state data for that configuration is stored in individual instances per workspace. It's the same configuration, but as you switch to different workspaces, you're actually switching which state data is being referenced, and that allows you to support multiple environments. You can also reference which workspace you're in by using the value terraform.workspace within your configuration, and you can use terraform.workspace to make decisions about the configuration that's actually deployed. In order to support multiple instances of our configuration, we are going to use Consul to store both the state data and the configuration data. How are we going to go about that? Well, we've got our Consul key value store, and within that, we have the networking path. We're going to store the state data in the state path, and Terraform has an easy way to append the environment name to each instance of state data, so we don't have to do anything else there. What we do have to do is separate our configuration data by workspace. So we will add workspace to the path for the configuration data to create that separation. And then for the applications team, we're going to take the exact same approach, storing the state data in the applications path and then using workspace as part of the path for our configuration data. So let's go ahead and set this up using Visual Studio Code. If you don't already have Consul running, you're going to need to get Consul up and running. So, simply go back to m4/consul and fire it back up using the consul agent command that we have on line 5. That'll get you up and running. Then, you can open up a separate terminal window, and we're going to go into the m6 folder and into the consul folder. So I'm going to go ahead and open up my terminal now. And I am in the main directory, so I will go into m6 and into consul. And, now we are going to add more configuration data to our Consul installation. First thing we're going to need to do is we're making these changes. We're going to need to set our environment variable, CONSUL_HTTP_TOKEN, to a token value. If you don't already have a token value jotted down somewhere, we can go back to the Consul UI. And in the Consul UI, I have logged in using the bootstrap token that has access to everything. So I can go ahead and click on the token for and click Copy here, and now the token value is copied.

Go back to Visual Studio Code. And I'm going to go ahead and paste the value here. And because I'm running PowerShell, I'm going to store it using this command, but if you're using Linux or macOS, you can use the export command. that is now my CONSUL_HTTP_TOKEN. Now I can put the additional configuration data from my workspaces into Consul. So I'm going to use the command consul kv put and tell it the path I want to put it into. So you'll see for networking it's networking/configuration/primary/development, so that's the workspace, and then net_info is the end of that path.

And then I'm going to place in there the information that's stored in Before I do that, let's take a look at what's in So I'll go ahead and open that, and you'll see there's two values here, the cidr_block for our VPC and the number of subnets, which we've dropped down to 2. So development only gets two subnets because we don't really need more than that for our configuration.

Now that we've looked at that, we can run these three commands, which place configuration data for development, QA, and production. So I'll simply copy these three and paste them down here, and now that information has been written to Consul and it's ready to be used by our workspaces. The next step is to create those workspaces. We've added our additional configuration data to Consul. Now let's go through the process of creating a development workspace for networking. So we're going to go into the networking directory, and the next step is to initialize the backend. You only have to initialize once and that covers all the different workspaces, because each of them is going to write their state data to this path with a slightly different key at the end, which we'll see in a moment. So I'm going to go ahead and run terraform and set the path to networking/state/primary. This is the same thing that we did before. It'll download our chapters, our provider plugins, and set up that state data backend for us. Once that's complete, we are going to create our development workspace. So we'll go ahead and run terraform workspace new development. Go ahead and copy that there and paste it down here. And you can see we have now switched to the workspace development, it's a new workspace for us.

If you want to know all of the current workspaces that exist, we can do terraform workspace list, , and we see there's the default workspace, which is created when you initialize Terraform, so that is always there and you cannot delete it, and then development, which is the workspace we just created, and Terraform automatically switches you over to that workspace when you create it. so that is all set and ready to go.

The next step would be to run terraform plan, but before we do that, let's take a look at the configuration to see what has changed to take advantage of Terraform workspaces. we are going to take a look at the configuration for networking, so let's go ahead and shrink up consul and expand out networking. And I'm going to go ahead and hide the terminal for now so that we have a little more space to work, and the only change in our networking configuration will be in the resources file. Let's go ahead and open that up.

So if we scroll down, the first thing we'll see is when we're initializing our data source for the consul_keys, under the path for networking, we've got a conditional statement. So for the path value, we are checking whether the terraform.workspace value is equal to default. If it is set to default, then we are going to get the information stored at the key, networking/configuration/primary/net_info. If it's not equal to default, which it's not right now ,it's set to development, then we scroll over, we are getting the path networking/configuration/primary/terraform.workspace, which will be substituted into development /net_info. So that's how it knows to get the net info that's stored in the workspace name, assuming it's not default. So that's one configuration change. Let's take a look at another configuration change. Before, in common_tags, we were setting the environment as part of the configuration data, but now that we're using Terraform workspaces, we can use that as the value for the environment tag for our resources.

So what we're going to do under common_tags is use the merge function to merge a map where environment is set to terraform.workspace with the map that we're getting from our common_tags data. Merge will merge those two maps together into a single map, and that will be our common_tags that are applied to all of the resources. So there is another place where things change a little. Scrolling down a little bit more, we get into our resources, and we can see that the name of the VPC is now going to be whichever workspace we're in, in our case it will be development. There is one other change that's in the networking configuration and that is enable_nat_gateway. That's now set to true, and the reason is that the application configuration we'll deploy on top of this does require a nat_gateway, so that's now been set to True. So those are all the changes in the networking configuration. Let's go ahead and get this instance of the configuration deployed. We've reviewed our configuration, and we're ready to get it deployed. Let's go ahead and bring the terminal up here, and I'm going to switch over to m6_commands. The next command that we are going to run is terraform plan dev.tfplan. Let's go ahead and grab that here and copy it, and we'll run the plan down here. And this will take a few moments, but we can see since it seems to be running successfully, it was able to go out and get all of the configuration data it needed from Consul. It looks like we're good.

Let's go ahead and run terraform apply dev.tfplan. So I've copied that, and I will paste it down here. And because we are creating NAT gateways as part of this configuration, it could take up to 10 minutes to deploy. I don't want to wait around for that whole thing. So let's go over to the Consul UI and see what's going on with our state data. We are in the Consul UI. Let's go over to Key/Value, and we want to go into networking, and let's go into state and see what's going on there. We have our primary state data, which was from our original configuration. We have another one called And when we create our QA workspace, it'll be :qa, and production, so on and so forth. In case you're wondering why it's env, the original command for workspaces was actually env for environments, and then they changed it to workspaces later. We also have a new path . Let's go ahead and open that up.

We're back in Visual Studio Code, and we're just going to run through the same process again. We'll run terraform workspace new qa. Go ahead and grab that. Now we're on the QA workspace. We'll go ahead and run terraform plan qa.tfplan. Now that's running. And lastly, once that plan is done, we can run terraform apply qa.tfplan, the plan is done.

Let's go ahead and run the terraform apply command, and that's going to go ahead and deploy our QA environment. And since we're spinning up three subnets, in this case, it has to spin up three NAT gateways. So that's going to take a little while. I leave the creation of the production workspace to you as an exercise. While this is deploying, why don't we find out what the application team is planning to deploy on our infrastructure? Our application team has an application they want to get deployed on this infrastructure we've been creating. Let's see what they're planning to deploy. Well, they want to set up a set of web servers, at least one in each availability zone that's available, and they want to add a load balancer to load balance traffic across web servers. And then they'd like to stand up on MySQL RDS instance in one of the availability zones and then a instance in another availability zone. And they've told us that they only need the instance in QA and in production so we don't have to deploy that additional instance if we're running in development, that's what they'd like to set up. But, how are they going to get all this information about our network without having to keep bothering us? Well, they can do that by reading our remote state. Remote state can be a data source for other configurations and that's pretty convenient, especially in this context. Just like any data source, you use the data keyword and then the type is terraform_remote_state.

And then finally, you have to give it a name by which to refer to it. The first thing you have to specify about the remote state is what type of backend you're using, and in our case, we're using consul for the backend. Now, since we're using Consul, we also have to provide configuration information about how to connect to that instance of Consul. In the case of Consul, we have to provide at least three arguments here, one is the path to get to where the network state data is, the second is the address where you can connect to the Consul server, and then finally, the scheme determines if you're using HTTP or HTTPS. Let's go back to Visual Studio Code, and we'll get our application deployment ready. In Visual Studio Code, we successfully deployed the QA environment as well as development, so those are ready to go as targets for our application configuration.

The first thing we need to do, just as when we set up our network environments, is add configuration data to Consul that will be read. So we are going to use Samantha's token from Consul to do that. So let's go over to the Consul UI, and from the ACL, we are going to click on Samantha's token, and go ahead and copy that token.

Before we go back, let's go ahead and look at the policy we've assigned to Samantha, because it's pretty important. Scrolling down, let's open up the policy here, and just as a reminder, we have given Samantha access to write to the key prefix applications, which means she can write configuration data and state data. We've also given her read access to networking/state, so she can read the state data from the networking group, but she can't change it in any way. That's very important to be able to read Terraform remote state as a data source. Now let's go back and load our application configuration data.

We still have that token value for Samantha; let's go ahead and paste it, and we'll grab this whole command and reset our environment variable from using Samantha's token. Now that that's done, let's go ahead and go into the Consul folder that has our configuration data. And just like before, we are going to write the configuration data for each environment to a path that includes that workspace name, and we're going to be writing our common tag information. Let's go ahead and see what's in one of these configuration data JSON files. So we'll go ahead and expand this out, and we'll take a look at the development environment app settings. So we're setting instance sizes for our autoscale group, and we're setting RDS settings.

That's basically what's in this configuration data, and it will make a little more sense once we look at the configuration itself. We'll go ahead and close that up and run these four commands to populate our configuration data. I'll go ahead and paste that, and we have successfully written all of our configuration data, which tells me that our Consul token is working properly. Now let's go into the applications folder, and now we are ready to review our application configuration before we deploy it. Now it's time to review our application configuration. There is a lot of different TF files so it probably makes the most sense to start with variables.

In our variables, we've got our region that we need to set and then the various settings for the Consul variables where we're setting things like consul_address, port, and datacenter. And then there are some application variables like setting the IP range where different things can connect from.

Right now, it's set wide open and then also the username and password for RDS. Obviously in a real world situation, you would probably want to put this sensitive data in something like Vault, but for right now, we're just going to fudge it a little bit and put it in our variables. That's okay for this, but don't do this in real life.

The next thing that we should probably take a look at is our data sources. We'll go ahead and open up datasources here, and just like in our network configuration, we are getting our Consul data using the consul_keys data source. The first one is applications, and just like we did for networking, we have a conditional statement here that looks at one path if you're using the default workspace and a different path if you're using a different workspace. We're also getting our common tags from the common_tags.

The more interesting data source, if we scroll down a little bit, is the terraform_remote_state for networking. Just as we saw in the presentation, the backend is set to Consul. For our configuration, we've hardcoded the address to the 127.0.0.1 port 8500. The scheme is set to HTTP, and then there is a conditional statement for the path because, just like the configuration data, the path to the remote state is conditional on which workspace you're in. If you're in the default, it will be networking/state/primary if you're using a workspace. And then as we saw earlier, it will now be the name of the workspace. So Terraform needs to figure out which workspace is being used and use that to access the state data for networking. Scrolling back over, we get into another data source, which is the AMI we want to use, which will get us the AMI in the current region for aws_linux.

All we have to do is set most recent to true to get the most recent version, set owners equal to amazon, and then there is a number of filters that will get us the exact right AMI that we want to use. That's everything that's in datasources so let's go ahead and open up resources. Now this was based on work I found in another GitHub repository.

We are defining two providers , our AWS provider and our Consul provider. This is very similar to the network configuration. Down in our locals, we're defining all of our local values and these are all the values that we're getting from the configuration data stored in Consul. We've seen this before. This isn't a surprise.

Scrolling down a little bit more, you see the common tags. Just like we did with the networking configuration, we're getting the environment value from Terraform workspace and merging that with all the common tags we're getting from what's stored in Consul.

Scrolling down some more, we get into resources. We're going to create an aws_launch_configuration. So this is the launch configuration that will be used for the autoscale group.

The name_prefix for the instances will be terraform.workspace We have the image ID that we're getting from the data source, and the instance type is from the configuration data in Consul, and then we're applying three security groups for this instance. We're also specifying some user data that's in templates/userdata.sh. Let's see what's in that file real quick. In userdata.sh, all we're doing here is updating yum, installing nginx, turning on nginx and making sure it stays on after a reboot, that's it.

So when we go to this web server, all we'll get is the default NGINX page, which we will replace in a future chapter so don't worry about that. Let's go ahead and close that up and get back to our resources.

Scrolling down some more, we are deploying an elastic load balancer with the name and then the workspace. And for the subnets, this is the first time we're using our remote state data source. So for the subnets, it's data.terraform_remote_state.networking.outputs.public_subnets. This is an output from our networking configuration which means you can only get information from the remote state data that's exposed as an output. If it's not one of the outputs, you don't have access to it. That's very important. You have to make sure whoever is creating that other configuration exposes the proper data for you in the form of outputs. Scrolling down a little bit more, we get into the listener and the health_check, which is just looking at port 80 and we're giving it a security group for inbound traffic.

Scrolling down a little bit more, we get into our autoscale group. For our autoscale group, we have to tell it which subnets it's going to be using, we have to give it a name, and then we have to set the minimum and maximum size, and when it should wait for elastic load balancer capacity.

We give it the ID of the launch configuration that we created a little bit earlier and the load balancer it should be using as well. Scrolling down some more, we get down to our Database Config.

For our database config, we're creating a database subnet group so these are the subnets where RDS can create the main database server instance, as well as any copies. For that, we're setting the subnet IDs to the private subnets in our VPC.

And then we're creating the actual RDS instance itself. Most of the settings are defined by our configuration data. The last file is security_groups and I'm not going to go through this in painstaking detail.

It's basically security groups that allow communication to the instances and to our load balancer and that's all there is to it. That's our entire application config so let's go ahead and get it deployed. We've reviewed our application configuration, and now we're ready to get it deployed for development and QA. Let's go over to m6_commands and go ahead and bring up our terminal. We are in the applications folder. That's where we need to be. We're going ahead and run terraform init. And you notice that the path is a little different here, it's applications/state/primary. Let's go ahead and grab that whole command, and we'll paste it down here.

And it should successfully connect to that Consul back end and get our plugins and chapters. Now we can create our first workspace, the development workspace, for the application configuration. Go ahead and copy terraform workspace new development, and now we have created and switched over to our development workspace.

Well now run terraform plan. Go ahead and grab that and paste it down here, and this will generate the plan. It needs to successfully connect to the remote state data for networking in order to plan successfully. So since the plan ran successfully, we know it was able to connect to that remote state data for networking.

Next thing to do is run terraform apply "dev.tf plan". This is going to take a while. It needs to create multiple EC2 instances. It needs to create a load balancer, all of those security groups, and lastly, it has to create the RDS instance. Our development instance of the configuration has deployed successfully. Let's jump over to the AWS console.

We're in the AWS console. We can see our EC2 instances deployed. There's two of them because this is development, and I only want two instances running. Checking the tags, it looks like all the tags have applied appropriately, so we are good to go, and it propagated those tags from the autoscale group, which was a pretty important thing. So that is all set. If we go down to Load Balancers , we've got one load balancer. That's our Elastic Load Balancer for the development instance, and we can scroll down here and get the DNS for it. Go ahead and copy that, and we'll go ahead and drop that , and I should get the NGINX page.

There's the test page, and we can see in the address, this is the development workspace. If we want to do the same thing for the QA workspace, we can go back to Visual Studio Code, and we can simply repeat it for the QA environment running terraform workspace new qa terraform plan and terraform apply. One word of caution. We have deployed an RDS instance, we've deployed multiple EC2 instances, and we've deployed NAT gateways. Those are not free. They will cost you money, so if you're planning to step away and do something else before you move on to the next chapter, I highly recommend destroying each environment so you don't get charged for all of those resources. In summary we covered using workspaces for multiple environments from the same configuration, and that was pretty helpful. And then we got into using remote state data for collaboration. We were able to successfully use the network state data for our application configuration, and that is also pretty darn useful. So be thoughtful about what outputs you expose from your configuration, because those are something that somebody else might want to consume. Next, we're going to get into an interesting area, which is troubleshooting when things go wrong with Terraform. In most of the examples I've shown so far in the book, Terraform has worked flawlessly, but I guarantee that's not always going to be the case, so in the next chapter we are going to see how you deal with when things go wrong.

Chapter 6 How to Troubleshoot Terraform

In a perfect world, the first time you write your Terraform configuration, it would validate, plan and apply seamlessly, and you can just keep trucking on deploying your infrastructure. But, we do not live in a perfect world and Terraform is going to break. In this chapter, we are going to explore the different ways it can break. There's a lot of ways Terraform can break. We're going to go through those. We'll start by validating our configurations because the first time you write it, it's probably not going to be correct. Then, we're going to look into the other ways that things can break when you get into the plan and apply stage and we will learn how to enable verbose logging to troubleshoot some of the more difficult issues. Sometimes resources get created and they are not created quite right, so we'll look at ways that we can force the recreation of a resource using taints. And then finally, sometimes Terraform crashes, and we'll look at how that crash log is generated and what to do with it. But first, let's start by talking about what's going on at ACME. At ACME, you've been furiously working trying to help the infrastructure and applications team get their infrastructure and apps deployed, and now the application team has come to you with an update to what you've already helped them deploy. They basically want to add an S3 bucket and we'll look more at that in a moment. Now they have been running into some issues as they try to develop that update and get it deployed, so they would like your Terraform skills to troubleshoot what's going on. And finally, they found a weird issue they'd like you to take a look at, they were trying to do something and it caused Terraform to crash, so, they're going to need your help with that one as well. But what are they trying to do to their application that's giving them so much trouble? Well it's actually pretty simple. If we take a stripped down look at the application, we essentially have public subnets and private subnets. The web servers are sitting in the public subnet and the database server and its copy are working in the private subnet. The web servers are not currently logging their data to a centralized location and the app team wants to change that. They'd like to create an S3 bucket and write information from those web servers to the S3 bucket. In order for those web servers to have permissions to write to that bucket, they are going to need an IAM role associated with each instance, and that's something we can do by updating the launch configuration resource. That's what they want to do. They are running into some issues, so let's talk about the different types of errors that exist in Terraform. There are broadly five different types of errors you're likely to encounter when using Terraform. The first is a command error. So this is simply an error that happens when you're using the command line, and there are definitely ways to troubleshoot that error. Then there's syntax validation, and this is the process by which Terraform validates the HashiCorp configuration language that you've laid out for your configuration, and also some of the logic within that configuration, and we'll talk more about that in a moment. There's also provider validation. When Terraform runs its plan process, the provider has to agree with what's in that configuration. Even if the provider agrees, when we get to the apply stage, sometimes deployment errors happen, and that's where we get into things like resource taints. Again, we'll talk about that in a moment as well. Then finally, sometimes Terraform just breaks. No application is perfect. Terraform is no exception. Sometimes things happen and Terraform crashes. What can you do when that happens? First, let's talk about command errors. Command errors obviously happen at the command line when you're using the CLI, and it's usually related to bad CLI syntax or arguments. Something in what you typed out just doesn't agree with Terraform. In order to figure out what part of that didn't agree with Terraform, you can use the help argument along with the command to see what options are available to you. If that doesn't work or you need more information, you can always go read the docs. Honestly, a lot of the time, the answers are in there, and ultimately, if that all fails, you can go on Stack Overflow and pose your question there. Let's take a look at the configuration that the application team has created, and we'll walk through a command error. In Visual Studio Code, I'm going to hide the terminal window for the moment. And we'll open up m7_commands. These are the commands we'll be running here. In order to get through these exercises, you will need Consul up and running. If you don't already have Consul up and running, go ahead and open a separate terminal, go into the m4 consul folder, and run the command, console agent. Before we run any commands, let's go ahead and take a look at what's updated in the application configuration. You can see that iam_instance_profile is a new argument in there, and we are sending it to an iam_instance_profile that we're creating in a separate .tf file.

That's really the only change here. So let's go over to this new s3.tf file, and these are all the new resources that the applications team has created. Basically, we're creating a bucket prefix. What do we want the prefix of the bucket to be?

We're creating a random integer using the random provider, and the reason is our bucket needs to be globally unique. Then, we're storing the name of the bucket in a local value. Scrolling down a little bit more, we are creating an S3 bucket and setting force_destroy to true.

If you don't do that, Terraform won't be able to delete the bucket if there's any files in it. Scrolling down some more, we're creating an instance profile, and that profile needs to reference a role.

So below that profile, we're creating that role. And within that role, we are adding a policy. And we're using the heredoc syntax to put the policy directly inline.

If you've never worked with profiles and IAM roles before, this might be a little bit confusing. Basically, the action that we're granting this role is the ability to assume a role in the EC2 service. So this essentially says EC2 instances can assume this role. That's important because we're going to give this role some permissions, and that's what we do next. Scrolling down a little bit more, we're creating a role policy that gets associated with a specific role.

And in this policy, we're defining some permissions on S3. We're specifically giving it permissions to run any action against the S3 bucket that we're creating above, and that's all the resources that we are adding. We are adding a new provider, and this is our configuration in a new folder, so we need to run terraform init. And this is where the application team ran into their first issue. So let's go ahead and run that command.

When you write your Terraform configuration, you're using HashiCorp configuration language and sometimes you make mistakes and that is where validation comes in very handy. Before you run validation, you actually need to initialize Terraform first and the reason is it's not just validating the syntax of HashiCorp configuration language, it's actually validating some of the logic in the providers you're using in your configuration and we'll see how that works when we get back to Visual Studio. It basically checks both the syntax and the logic, which is huge, that can save you a lot of time from not having to troubleshoot problems. One thing it doesn't do is check state, so it's not comparing your current state against the configuration or anything that's going on in the public cloud or whichever provider you're using, it's simply checking the configuration itself, which is why you don't need a state file to do it. The validation run can happen manually, you can run terraform validate or it could be automatic as part of another command. For instance, if you run terraform plan, terraform apply, or even tried to change Terraform workspaces, all of those will trigger a validation check and throw errors if the validation doesn't come back clean. One place validation checks are especially useful is in automation. I have seen many scenarios where when someone wants to merge a pull request into the master branch, it has to pass terraform validate cleanly before that merge will be considered. So let's go back to Visual Studio Code and validate our configuration. In Visual Studio Code, let's go ahead and run the command. It's pretty simple, it's just terraform validate. So I'll go ahead and copy that, and paste it down here.

We immediately ran into a validation issue on s3.tf line 3. Well that's helpful, it tells me exactly where to look. Let's go back up to the top of our file, and in line 3, type is set to strings. The application team made a mistake there because strings is not a valid type. So let's go ahead and comment out that line and we'll uncomment the line below it, so now that is correct.

We'll go ahead and save the file and we're going to run terraform validate again. Let's see what happens. Well, we have another error.

That's get used to this, this is going to happen, it's just part of debugging Terraform, and in this case on line 65 we have an unsupported attribute. So in the first case it had nothing to do with any providers, it was something that was native to HCL and Terraform, but in this case it has something to do with the provider itself. Let's scroll down to line 65, and we can see that the role we're trying to reference for our role policy, we used the attribute ids, or I should say the application team did, and unfortunately ids is not an attribute of that resource.

Terraform knows that because we initialized, and when we initialized, it got the provider plugins and it has access to that information. So what we need to do is go ahead and comment out this line and we'll uncomment the line below it and save it.

The proper attribute is id, and actually it suggests that down in the error, maybe you meant id, which is nice. We'll go ahead and run terraform validate one more time, and now it came back clean.

That's fantastic. So now let's talk about provider validation and deployment errors. Once your configuration has been validated and you think it's ready to go, you're probably going to run terraform plan, and you may get provider validation or deployment errors. Those can happen during plan or apply. The provider validation usually happens during plan, and then deployment errors, well, those really only happen during apply. The most important thing, and the best advice I can give you is read the error message, then read it again, and then maybe read it a third time. I can't tell you how many times I have gone down a rabbit hole because I read the error message, made an assumption, went down the line of that assumption, and then later figured out that the error message was telling me something else. So maybe read it a few times and make sure you understand it. Sometimes the error message doesn't give you enough information. In that case, you want to enable verbose logging at a level that's appropriate for you to get the detail that you need. And finally, during deployment, sometimes resources will be created that were not created properly. Sometimes Terraform knows that and sometimes it doesn't. If it doesn't, you can taint the resource manually to force creation, and we'll talk more about tainting and logging in a moment. Let's go back to Visual Studio Code and we'll run terraform plan. We're back in VS Code, and we are ready to run our terraform apply. But before we do that, we have to select the correct workspace that we want to use. So we're going to go ahead and run terraform workspace select development. So we have successfully selected our development workspace. Let's go over into commands, and we'll go ahead and run terraform plan dev.tfplan. So we're trying to run that, and let's see if we run into any errors. Looks like we didn't run into any errors in the plan stage, so let's go ahead and try to run terraform apply and see what happens.

So we're running terraform apply dev.tfplan. It's going to try to create the resources that are in our updated version of the configuration. It's going to be that S3 bucket, the role, the role profile, the role policy, and it's going to update our launch configuration with that IAM role.

But we did get an error here. Let's see what the error is. Scrolling up, there's actually two errors. , the first one I understand.

It says InvalidBucketName: The specified bucket is not valid. let's go ahead and take a look at what's in s3.tf, and I'm going to shrink this terminal down just a little bit so we can look. And we're going to scroll up to where we're creating our S3 bucket.

For the S3 bucket, we use the local value, local.bucket_name, and if we look at the bucket name, what's the name? Well, it's the bucket prefix and the random integer, but with an underscore. You may not know this, but S3 bucket names can only have dashes, numbers, and lowercase characters. They cannot include an underscore, and that's why we got an error. Well, that solution is obvious. We'll go ahead and fix that right now. Go ahead and save it. We've got a dash instead of an underscore. Next, we can look at the second error. Let's see, what does it say? It says Error adding role aws_iam_role.asg.name to IAM instance profile, NoSuchEntity. The role with the name aws_iam_role.asg.name cannot be found. What does that mean? Well, it says on s3.tf line 32, there's an issue. Let's go ahead and look at that line.

So it's trying to create this instance profile. And under the role assignments, I see what happened here. The reference to the IAM role is in quotes, so it's treating it as a string and not as a reference to another object in the configuration. Obviously, all we have to do is comment this one out and uncomment that one, which is the correct syntax for it. And you note, this is not something that terraform validate picked up on because it thought, you're referencing a role with a string. That's fine. It didn't pick it up because it wasn't incorrect from a syntax standpoint. It was incorrect once we actually tried to deploy it. In theory, we have solved our two deployment errors. So let's go ahead and run terraform plan once again.

Terraform plan came up with three to add, two to change, and two to destroy, and that probably is because it needs to recreate some objects based on what we did. We changed the IAM instance profile on our launch configuration, so that has to be replaced. We changed the bucket name, so the policy will be updated that references that bucket. We changed the instance profile because it was tainted, and so that has to be replaced, and we'll talk about taints in a second. And then finally, the autoscaling group that references the launch configuration, because that launch configuration is being replaced, we have to update the setting for this launch configuration, so that is updated in place. We are all set. We'll go ahead and scroll down to the bottom, and we're going to run the terraform apply for this, and hopefully, it successfully creates all of the resources that we need. We get a new error here that says, Error IAM instance profile already exists.

It did say it was going to destroy that one because it was tainted and recreate it, so what's happening here? We might want to enable more verbose logging to troubleshoot this particular error. How do we do that? Errors that occur in Terraform can sometimes be a little hard to track down based off of the information that's in the error. So we can up the level of logging that's happening. Turning on verbose logging basically means that the actions that Terraform takes are being exposed at different levels depending on what type of verbose logging you enable. The way that you enable it is by setting the environment variable TF_LOG to one of a number of different settings. You can also write this logging out to a file by setting the environment variable TF_LOG_PATH, and that has to be the full path to a file, not just a directory. That's where it will write the data from the logging. As far as different levels of logging, the most verbose is called trace, and then there's debug, info, warn, and error. Generally speaking, you want to set it to something like info or warn because trace and debug are incredibly verbose. So unless you're really trying to dive down into the bowels of what Terraform is doing, maybe you start with warn or info and then expand as needed. One place that logging is especially useful is just like in validation, if you're using Terraform in automation, you may want a report of everything that Terraform did at a particular information level. So you can set TF_LOG as part of your automation process and set TF_LOG_PATH to store that information in a file. Let's go back to Visual Studio Code and we will enable verbose logging. We're back in Visual Studio Code, and let's go ahead and turn on logging here. Now what level of logging do we want to do? I'm going to go with info for this one, and I'm going to use the environment variable TF_LOG to do that. I'm setting it to info. If you set TF_LOG to nothing, then it will stop the logging process. So sometimes if it's too verbose, you don't need it anymore, just set it to nothing and you'll be good. Once I do that, logging is now enabled. Let's go ahead and run terraform plan again, because now my plan operation is out of date. I did try to Run and Apply. Now you can see how much more verbose this is, and everything in it was marked with what level of information it's trying to give you.

Let's go ahead and scroll up and maybe get some more information on what's going on here. And the problem we were having was with the instance profile. So here's the instance profile. Now one of the things that I know about this, because I can see it right here, is that it's got a green +, a /, and then a

What that actually means is that it's going to create a new instance of that resource and then delete this resource, which is a problem because they both have the same name, and you can't do that with an instance profile. So when it tries to replace it, it's not going to work. That's something I happen to know from troubleshooting this before, but you may not know that. So let's go ahead and run terraform apply, we'll run through the apply process, and it's going to show you in exhausting detail, every action that it took, right up until it failed.

And if you looked through all of those logs, you would see that it tried to create the new resource before deleting the existing resource. Fortunately, the way that you handle that is by adding a lifecycle block. Let me go ahead and shrink down the terminal for a minute, because it'll be easier to see this. We'll go to our resources, because I already have a block for this, and under our resources, in the launch configuration, you can see the lifecycle block there, and for that one, it's create_before_destroy set to true. It would appear that our iam_instance_profile has the same lifecycle, even though it's not explicitly stated.

So I'm going to go over to our instance profile, and I'm going to go ahead and put a lifecycle block , except I'm going to set it to false. It is not going to try to create a new instance of this resource before it deletes this existing one.

Hopefully, that is going to take care of our error. So let's go ahead and run terraform plan again. , we have our plan; let's go ahead and try to apply it. , it looks like it has failed yet again. So even with the lifecycle block, it didn't work.

It looks like we've actually run into a bug with Terraform. It should be destroying that instance before it tries to create a new one. Unfortunately, that is a bug that I'm going to log with Terraform, but in the meantime, let's go ahead and update the name of this ASG profile so it doesn't try to create one that already exists, and we'll just call this _bug. We have a name for this profile, we'll go ahead and run terraform plan again, and we'll go ahead and run terraform apply. And it successfully applied our configuration, which means it destroyed that instance profile, but you can see that's the last thing it did was destroy that instance profile.

So even though we put the lifecycle block in there and told it to destroy it before it creates it, it looks like Terraform chose to ignore that for some reason, which means there's probably a bug in that AWS provider, and I'll need to file that bug later. So we have successfully deployed our configuration. You'll remember, the whole point of this was to create an S3 bucket and get that IAM role applied. Did we successfully get that IAM role applied? Let's go over to the AWS console and check it out. , here we are in the AWS console looking at our instances, and each one should have an IAM role configured.

So we'll go ahead and look at the first one and scroll down, and there we see under IAM role, it is still blank. Why might that be? Well, it's because even though we updated our launch configuration, the autoscale group associated with that launch configuration does not automatically delete instances and recreate them, and that is how IAM roles are assigned.

How can we force the recreation of these instances? We can do that through resource taints.

Chapter 7 Resource Taints & Crash Logs

Resource taints are a way to force the recreation of a resource. It basically tells Terraform, recreate that resource for me because I know something is wrong with it. Terraform will taint things automatically if it knows they weren't created successfully. And we just saw that when we tried to create that instance profile and it had a bad attribute associated with it, Terraform tainted the object. Now, of course, we had a little trouble recreating that object because of the name, but we did see how Terraform will taint something automatically. You can also taint something manually and you identify that object by its address within Terraform. The easiest way to get that address is by running terraform state list and it'll list out all the resources in your configuration. Resources can also be untainted. So if Terraform marked something as tainted and you know it's actually fine, you can untainted that resource so Terraform won't destroy and recreate it. You also might just want to do something yourself as opposed to having Terraform do it. But how does the taint command work? Let's take a look. There's two commands here, there's taint and untainted. They're very, very similar, so let's look at taint first. The general syntax is terraform taint, any options, and then the address of that particular resource you would like to taint.

As an example, to taint a single resource, let's say it was an aws_instance, we could do terraform taint aws_instance.example and that would taint that resource. What if it's a collection of resources or a resource and a module? You can do those as well, you would just have to specify which object in the collection you needed to taint, so aws_instance.collection and then [0] would get you the first item in that collection. If it's a module, you just need the full address including the module. Use terraform state lists to figure out what the address is for that object. Untainting is very, very similar, it's just terraform untaint and the address of the resource that you want to untaint. For example, terraform untaint aws_instance.example. So those are the taint and untaint commands, let's go ahead and use them in practice. We're back in Visual Studio Code, and we know that even though we added an IAM instance role to our launch configuration, those EC2 instances aren't getting it yet. We could get the autoscale group to add new instances and delete the existing instances, but that's no fun. Why don't we do it by tainting the autoscale group itself and that will force the recreation of the instances as well. Let's go back over to m7_commands, and we'll scroll down here and go ahead and run terraform state list and that gives us a listing of all resources within our configuration. The one that we're most interested is if we scale up, it's this aws_autoscaling_group.webapp_asg.

The important thing to understand about tainting a resource is that any dependent resources are not automatically tainted as well. So you have to understand what the dependencies are in your configuration, and if you need to recreate any of those associated resources, taint those as well. In our case, the autoscaling policies and CloudWatch metric alarms are both associated with that autoscaling group so those need to be tainted and recreated as well. Let's go ahead and run through that process now.

So we're running terraform taint and then the address of each of those resources, I'll run those down here, and all of our resources have now been successfully tainted. You may notice that I still have verbose logging on, that's not very helpful. Let's go ahead and turn that off, go ahead and copy this command here, and I'm going to replace info with just blank. We won't have that verbose logging kind of getting in the way. We have successfully tainted those resources so Terraform is going to recreate them. We'll go ahead and run terraform plan here, and once that's complete, we'll go ahead and run terraform apply after that.

So it's going to add five resources and destroy five and those five are the five resources that we tainted.

Let's go ahead and run terraform apply. Terraform apply is now running, it's going to try to destroy those resources, and we get an error, AutoScalingGroup AlreadyExists.

That sounds really familiar, that sounds like the instance profile. Let's go look at what's going on with our autoscaling group. So I go over to resources and go ahead and scroll down here to our autoscaling group and oh create_before_destroy was set to true, so it was trying to create the autoscale group before it destroyed the tainted one.

It respects this lifecycle setting so I'm going to go ahead and set that to false and save it. And now, let's go ahead and run terraform plan again, and before we run terraform apply, we'll take a look at what's in the plan.

Our terraform plan has completed, and we'll go ahead and scroll up to our autoscaling group and you can see it has been tainted. And then next to that resource, you can see it says and that's how you know it's going to destroy a resource and then create the replacement, which is what we want it to do, it's also what we wanted our instance profile to do, but Terraform or the provider decided it didn't want to play along.

So now let's go ahead and run terraform apply, and because we change the lifecycle setting, it should now destroy it before it tries to create a new one. And as you can see, it is destroying the autoscale group so we know that we got the order right this time. Now it's going to take a little while to recreate these resources so I'm going to resume when it's completed creating all the resources.

It finished our apply, five resources were added so that's autoscale group and the dependent resources on that and one was destroyed. Let's go back to the EC2 console and make sure that our instances now have that IAM role. We're back in the console. Let's go ahead and refresh this and we can see we've got two instances that were terminated, those were our old instances. Let's look at this instance and , under IAM role, it has the development_asg_role.

This instance will now be able to write to the S3 bucket that was created. We've fixed most of the application teams issues, but they do have one more for us and that is a Terraform panic, so let's talk about that.

Sometimes, Terraform crashes. All applications crash. It's generated basically when Terraform panics, or at least that's the verbage they use because Terraform is written in Go, and Go panics. It doesn't crash. That's how they decided to do it. It's basically caused either by Terraform, or it can actually be caused by a bug in one of the providers associated with Terraform, maybe even the AWS provider. It's similar to trace logging in the sense that the output from the panic goes into a crash file that looks like what you would get if you enabled trace logging in verbose logging. Generally, your only rebook here is to open an issue on GitHub, unless you happen to be a Go developer and really want to get entrenched in this. There are other people that maintain these providers and people that maintain Terraform. They're going to want you to log an issue on GitHub and provide that crash log, as well as your configuration that caused that crash. Let's take a look at how a crash could get generated by the application team. The application team has been messing around with the Terraform init command, and they found this command called where you can take an existing Terraform chapter and use it as a skeleton for a new configuration. They got pretty interested in this, and then they ran into a bug. So let's go into the folder where they have an example ready for us.

Go ahead and paste that, and let's see what's in these files. So we have a folder called toplevel, and in that, we have a main.tf configuration, and that main.tf configuration invokes a chapter that's in a subfolder. Let's go ahead and look at that module, and all it has in there is an output block. It's a very simple configuration, and this is the one that they want to use with the command, and they're going to do that from this test folder here. One thing to note is I'm running a specific version of Terraform that has this bug. If you run terraform version, you'll see I'm on 12.24. This is an active bug on GitHub.

So if you have a newer version of Terraform, it may not throw this if you're following along, but let's just go ahead and run the command. The command is terraform init So we're using that toplevel folder to initialize a configuration in the test folder. Go ahead and run it, and, Terraform crashed.

So this is exactly what you'll see if Terraform crashes. And if you look in the directory over here, there's now a crash.log. We can take a look at that, and you'll see it looks very much like the logging we were seeing before.

If you have a Terraform crash, then the best thing to do is go on GitHub. It even helpfully provides the address to log an issue, and go ahead and log that issue along with all the other information that they're asking for. That's immensely helpful, and it helps other people out who are running into the same issue. That's basically what you can do if you run into a Terraform crash. In Summary errors can and will happen. In fact, we just ran into an error but we just rolled with the punches, and we troubleshot it together, and we figured out what was going on, and now I have a bug to go log. So, don't worry. Errors will happen, and it's when you do get an error, make sure you read it at least twice, and then think about it for a little bit. Try not to make any big assumptions, but actually read what it's saying to you. Also enable logs. If the error isn't giving you enough information, go ahead and enable verbose logging and see what's going on behind the scenes. Lastly, you can use taints to force the recreation of resources, but make sure you're paying attention to the lifecycle of those resources so that they are destroyed and recreated in the proper order. Next, we're going to apply some of this information that we've learned about automation to a CI/CD pipeline. We are going to integrate Terraform with Jenkins and enable automation for Terraform in a CI/CD pipeline.

Chapter 8 DevOps Terminology

When you think about the deployment of an application, laying down the infrastructure is just one part of that process. Your applications are going to be deployed upon that infrastructure sometimes using something called a continuous integration and continuous delivery pipeline. In order to reduce friction and streamline the application deployment process, it makes a lot of sense to add Terraform configurations to the pipeline and to source control. Let's get Terraform integrated with that process. In this chapter, we are going to add Terraform to a CI/CD pipeline. Before we dive into the actual integration of Terraform, let's first level set with some terminology. Once we're comfortable with our terminology, we're going to look at how automation is going to be used at ACME, and we'll walk through an example of how that deployment pipeline could be set up. We're going to review Jenkins, GitHub, and some of the supporting services behind that, and then we'll also talk about some considerations when it comes to implementing automation and CI/CD with Terraform. So let's dive into some of that crazy terminology. In the introduction, one of the things that you tend to integrate Terraform with is source control management, and there are multiple different formats that source control management can take and you're probably familiar with at least one of these formats. There is Git, team foundation version control, and subversion. Those are just a few examples of the format that source control management could take. There is also multiple platforms that take advantage of these formats. You'll probably be familiar with ones like GitHub, but there is others like BitBucket, GitLab, CodeCommit, and more. What source control management does is a few different things. A big one is enabling collaboration with a distributed team all working on the same corpus of code together. You can create your own branch and then merge your changes back into a central or main branch and you can also have other people simultaneously working on other branches and they won't conflict with the work that you're doing, or if they do, you can resolve those conflicts before they're merged into the main branch. Another thing that source control management brings is version control. As your main branch iterates or your individual branches iterate, you're going to have multiple versions, and if something goes wrong, you can always roll back to a previous version or pick out pieces of that previous version to roll forward. It's very flexible and it gives you a high degree of control of what's actually being deployed in production. Something that goes with source control are CI/CD pipelines. You've probably heard the term CI/CD and maybe you weren't even sure what the CI and CD stand for. CI stands for continuous integration and CD usually stands for continuous delivery. In a similar fashion, there are multiple platforms to support CI/CD. You've probably heard of some of these, things like Jenkins, CodePipeline, or even Bamboo. What continuous integration does, the CI, is when code is checked into a repository, continuous integration is going to run some processes against that to check that the code is valid and then potentially merge it into an existing branch, it does some of the work for you. The continuous delivery side of things is how new builds are created based off of the code you're checking in and then delivered in preparation for being deployed to a target environment. Sometimes people say continuous deployment, which is actually an additional step upon delivery, delivery just implies that you have the build artifacts ready and you can deploy that build when necessary. Part of the CI/CD process is usually some type of pipeline that does some automated testing and validation of the code that you're submitting before it makes its way to a production environment. In that regard, pipelines tend to support multiple environments and there is a flow between those environments, something that goes to development and passes development can move its way into QA or testing, which can then move its way to staging, and finally, production. So those are CI/CD pipelines. It sounds scary, but they're really not that scary, and we're going to build one in this chapter to show you exactly how not scary they really are. At ACME they have been using GitHub for their source control management, and for their CI/CD pipelines, they have chosen to go with Jenkins. It's a very common pipeline tool, and ACME feels comfortable using it. And finally, as we've seen previously, all of the configuration data for our Terraform builds will come from Consul, so that is staying consistent from our previous chapters. What does the pipeline look like in their build process? At ACME, they are using Jenkins for their CI/CD pipeline, and they're breaking it up into multiple pieces, which are usually called stages in the world of Jenkins. So we're going to have multiple stages that our pipeline advances through. The beginning of that process is code check in. When new code is checked into Jenkins, it could trigger this entire build process. The next step is a natural evolution of what normally happens with Terraform, you move to the initialization stage, so we've checked in our code, we've initialized our Terraform configuration for a specific workspace. Then after initialization, we want to validate that the configuration that was submitted via SCM is in fact good, so we're going to run terraform validate to make certain of that. If the validation passes, now we can move on to terraform plan, and we're going to run the plan command against our target environment. If our plan runs successfully, we move to the approve stage, and in our case, this is a manual approval process. The pipeline will wait for someone to check the plan and say yes, this looks good, and give it the approval, which will move it to the final stage, which is apply, when it applies the actual configuration. In our particular example, if you decline the plan, it actually destroys the environment, and that's just an easy way to tear down this environment when you're done with it. You probably would not do that in a production environment, it's really just there for convenience. So that is our automation setup that we will be doing using Jenkins. But how are we going to get Jenkins deployed? When it comes to setting up and installing Jenkins, you have a number of different options. You could just install Jenkins as a traditional app, it uses Java, you could run it on your local desktop, but that's no fun and you have to have the right version of Java running and we know that can be a pain. So what's another viable option? Well, you could deploy Jenkins in a container. The container is already going to have all the correct versions of Java and the latest version of Jenkins, and it will initialize the Jenkins configuration for you, so you just go straight to the admin logon and move on from there. A final option is to deploy Jenkins in the cloud, and this is actually useful if you're deploying to say AWS or Azure exclusively, you can give the instance that Jenkins is running on a role with permissions to deploy infrastructure and it can use that role as it's running the pipeline. So there's a bunch of different options for how you could run it. For our purposes, we are going to use Docker to deploy Jenkins in a container. So let's flip over to Visual Studio Code and see how we're going to accomplish that. In Visual Studio Code, we've got two folders , applications and networking, that have our configurations waiting for us to use, and we've got our m8_commands.txt. Let's go ahead and open that up. To successfully work through this, you're going to need Consul running in the background, and if you haven't already picked up on this, you're going to have to go into chapter four and fire up Consul with the command that's on the screen. So if you haven't done that, go ahead and do that now. The next thing we're going to do is create two tokens for Jenkins to use as it's going through its CI/CD process because just like Samantha, Jenkins is going to need access to Consul for the net configuration and the app configuration. The first thing we're going to do is set our route tokens so we can create additional tokens in Consul, so I'm going to set my environment variable CONSUL_HTTP_TOKEN to the root token, which I already have entered here. Go look at your notes if you don't have it available, but I'm assuming you've already got this part down pat.

So go ahead and open up the terminal here and whoa, that's a little high, let's scroll that down. And I'm going to copy this environment variable command, go ahead and paste it down here. So now we have that stored in our environment variable, we can run the Consul commands and it'll use that token in the background. We're going to create two tokens here. The first one creates a networking token with the description Jenkins networking. We're assigning it the policy networking which is the same policy we assigned so it will give access to the network state and the network configurations. The second one has the description Jenkins applications and it's getting the policy applications which will grant it access to the application state and configuration information, as well as access to the networking state. So let's go ahead and run these two commands. We have created our two tokens and I'm going to scroll up here so we can see what the output was and we want the secret ID from both of these.

It looks like PowerShell garbled it a little bit, but this is the secret ID for the networking token. I'll go ahead and grab that and paste it off the screen in a Notepad. I have that one and I'll go ahead and copy the secret ID for the application token. So I have both of those stowed away for when we configure our Jenkins installation to use these tokens. Now we're going to create our Jenkins container. The first step in that is to pull a Jenkins image from Docker Hub, and we want the Jenkins LTS image. That gives us the Supported version of Jenkins. So I'm going to go ahead and run docker pull jenkins/jenkins:lts. so that is running, and it is downloading all the various components of that image.

It should only take a few moments because it's not a particularly big image. Once it's completed downloading the image, we can use docker run. Docker run starts up a container based off of an image. And if we take a look at what's in this command, it's saying docker run, so start a container. We want to expose port 8080 and port 50000 from the container to the container host. Dash d lets it know that it should run in the background and not in the foreground. Scrolling over, we want to give it a volume where it should write its persistent data, and we're going to call that volume jenkins_home, and it's going to be mounted on var/jenkins_home. And if we scroll over a little bit more, we're naming this container jenkins. And then finally, the image that we're using for this container is jenkins/jenkins:lts. So I know that's a lot to take in, especially if you've never used Docker.

Let's go ahead and grab this whole command, and we're going to paste it. When the Jenkins container starts and finishes its initialization process, in the logs it puts the administrator password that we can use to further configure Jenkins. So if we wait a few moments, we can then run docker logs. And what the docker logs command does is you tell it which container you want the logs from and it directs that to standard out. So let's go ahead and paste that down. And it looks like it is still initializing. And if we scroll up a little bit, Here is the admin user code that we're going to put into the initial page for Jenkins. So let's go ahead and grab that right now. Now we're going to open a browser and start the process of configuring Jenkins to run our pipeline.

Chapter 9 How to Add Terraform Plugin

We are at a browser and in order to get to Jenkins, we're going to go http://localhost, and we'll go to localhost 8080 because that's the port that we exposed it on.

It's asking us to unlock Jenkins and we already got the initial admin password, but if you didn't have it, it gives you the location where it is stored. Let's go ahead and paste that in now, and the first thing it's going to do is ask us if we want to install suggested plugins.

Generally speaking, you do, so I'm going to go ahead and click on that and then we'll will walk through the process of installing all of these initial suggested plugins.

When it's done, we'll walk through the process of preparing this Jenkins installation to run Terraform. That plugin installation is complete, let's go ahead and create a user for ourselves. Save and Continue and then it's just letting us know we could change our instance configuration if we want to, but we're going to leave it at localhost, Save and Finish, and now Jenkins is ready for us to use.

What do we need to do now that we're in Jenkins?

The first thing that we're going to do is we're going to install the Terraform plugin because we're going to kind of need Terraform available if we want to execute a pipeline with it. So let's go back to the browser. We're back at the browser, we're going to go into Manage Jenkins, and we want to Manage Plugins, and we're going to search for the Terraform plugin under Available.

So we'll go ahead and click on the Available tab and search for Terraform, and there should be only one result here. We'll check the box here and Install without restart. This does not require a restart, it's merely installing the plugin and then we'll tell it which version of Terraform it should download and use.

So that has been successfully installed, let's go back to the main Jenkins page, we'll click on Manage Jenkins, and go into the Global Tool Configuration because the plugin is considered a global tool. And within the Global Tool Configuration, we're going to scroll all the way down to Terraform, and we want to add a Terraform installation to be used. Go ahead and click on that button, and we'll call this, very simply, terraform. If you had multiple versions , like you're going to run 11 and 12 and 13, you can create multiple Terraform installations to reference, but, let's keep it simple we're just going to use almost the most recent version of Terraform.

The most recent version is 13 because that literally just went to GA, but we've been doing this all with 12, so I'm going to stick with version 0.12 for the demonstration. Let's scroll down and get the proper version. We want the linux amd64 since the container in which Jenkins is running is a container, and that's it.

We'll go ahead and click on Save, and now that executable will be available for us to use in our pipeline configurations. That's all set. What else would we need as part of Jenkins when it comes to running a pipeline? Well, we know we're going to need those tokens for Consul access and we're going to be deploying things to AWS, so we're going to need some credentials for AWS. Generally speaking, you would create separate and new credentials for your pipeline, we're just going to reuse the deep dive credentials that we created at the beginning of this book, but obviously that is up to you. So let's go into Manage Credentials, that's the next stage here. We are now ready to configure the credentials for our Jenkins Pipeline. We're going to go ahead and go into the global section of the credentials, and we're going to add four credentials here. So I'm going to click on Add Credentials. The kind for all of these is going to be Secret text.

And there's two fields we're going to have to fill out, one is the Secret value itself, and the second one is the ID by which we can reference the secret in our pipeline configurations. Let's start by doing the consul_tokens. So if we jump back to Visual Studio Code, and scroll down a bit here, we have the names of all the credentials that we need to create. The first one is the networking_consul_token, so I'm going to go ahead and grab that so I get the naming exactly right and paste it under ID. And then for the token itself, remember I have that saved in a separate

Notepad file, so let me grab that text, and I'm going to go ahead and paste that in the Secret, and click OK.

So now we have our first one, we're going to repeat the process for the application token. We're going to grab the name from Visual Studio Code and change the kind to Secret text, paste in the ID, grab the secret for applications, and paste it under Secret. That's our two consul_token secrets. The next two things we need are the access key and secret key for our AWS credentials. So let's go ahead and grab the ID for the first one. And I'm going to add credentials here, go down, say Secret text, paste in the ID, and I have my AWS credentials over in the same Notepad document. So go ahead and paste that in, there is my access key. And lastly, we're going to do the secret key. So I'll go ahead and grab the ID of that, and go back to Jenkins, select Secret text, put the ID, and lastly, paste in the secret key. And click OK. So now all of our credentials are ready to go. The next thing to do is go ahead and set up a pipeline, but before we do that, let's take a look at some considerations when it comes to using Terraform in automation.

Chapter 10 Terraform Automation Considerations

When you're running automation in Terraform, there are some things you need to be aware of. The first is your plugins. Where are you going to get your plugins from and do you care about source controlling them? You can also specify the version of plugins you want to use, but some companies really want to store those plugins locally so they're not constantly downloading them from the internet. So that is one thing to consider, especially when you're running things in automation. Another thing you need to consider is workspace usage. Workspaces are highly encouraged when you're creating these types of pipelines. You're probably going to be using the pipeline to deploy different environments and workspaces are the perfect way of dealing with those different environments. Workspaces let you separate out state, they allow you to reference the environment within the context of a build pipeline, and then it can also impact how each configuration is rolled out. Will it be rolled out automatically or is there a manual step that has to happen because it's going to production? In terms of state control, really the only option here is remote state. Could you do it with local state on the build server? Possibly, but I really encourage against doing that sort of thing because you want that state to be available if you replace that build server or if something else needs to reference that state. So generally speaking, you're going to be using a remote state. The only question here is how do you initialize that remote state within the context of the build pipeline? Another thing to think about is output control. Terraform can be pretty chatty as we've seen and I'm sure you've noticed as we've run plan, and apply, and initialization, it produces a decent amount of output, and for the most part, that's fine, but sometimes that can mess up automation pipelines so there is an environment variable called TF_IN_AUTOMATION that can help you reduce the amount of output and the of Terraform by letting it know that there's not going to be a human being watching this output. Another thing to think about is what's the deployment pattern to follow when you're using automation? And our pattern has mostly been to initialize, plan, and apply. There are a couple of things to consider when it comes to automation. Once the plan is generated, do you want someone to manually review that plan before it's applied to the environment? In production, I'd say you probably do. In development, you might not, so you have to make the decision about how each task folds into the next one. Another thing to deal with is the fact that it's an automated process so there is no user prompt for input if you missed a variable. The last thing is error handling. How do you want to handle things if the process errors out? Is that the end of the pipeline or does it continue on, and if there is an error, how are you going to log it appropriately? Jenkins is pretty good about capturing the Consul output and logging it, but other automation platforms might not be as good at doing that and you want to store your logs somewhere else. Let's talk about some of the environment variables that Terraform offers up to help you with this automation process. Terraform supports a vast number of environment variables. Sometimes it can get a little bit confusing. Some of them are specific to automation and I wanted to highlight those now. The first one I already mentioned, it's called TF_IN_AUTOMATION, and if it's set to any value, it doesn't have to be true, but I set it to true so it's clear, if you set it to any value, it lets Terraform know it's working in an automation context, so not to create any Consul output that might mess with an automation framework. Another thing you're probably going to want to do is up the logging of Terraform to something like info or possibly even higher because if something does go wrong, you're not there to see it. You want it to be caught in the Consul logs of the pipeline that you're using. You could also, as I said before, capture that log data in a path somewhere and I would likely upend the month, date, year, and even timestamp to that log so if it's run multiple times, you're not overriding your existing logs because that's, that's bad. The next one is an environment variable called TF_INPUT. If that's set to false, Terraform will not prompt the end user for input, meaning if you run terraform destroy and it prompts you for a yes or no, Terraform is not going to do that, it's just going to cancel an error out at that point. So if it encounters any situation where it does need user input, it's actually just going to error out rather than waiting for that user input. Another useful one, and this is not specific to automation though, I do think it's helpful, is if you create an environment variable called TF_VAR_ the name of the variable in the Terraform configuration, Terraform will see that and use that value in the configuration so it's a really easy way to pass a variable value without adding it to the command line. The next one is TF_CLI_ARGS and this is another way to cut down on arguments in your command line actions. You can define arguments through TF_CLI_ARGS and those arguments will be applied to every time you run Terraform within the context of that build. So those are a bunch of environment variables that are helpful in automation and we are going to make use of those in our Jenkins file. Speaking of which, let's go back to Visual Studio Code and look at what's going to be driving our pipeline. It's time to take a look at what is in networking. So let's go ahead and expand that out. This is the same networking configuration that we've been using pretty much all along with a few very light changes. So I do want to highlight those changes and then we can look at this weird thing called Jenkinsfile, what's that all about? Let's start with the backend. The only difference here is because we're running in a Docker container and it wants to talk to consul, which is running on our main desktop, the address changes a little bit.

Instead of being localhost or 127.0.0.1, instead, the address is host.docker.internal, which will resolve to our desktop. So that's how it's going to get to console. So that is one small change here. The other big change is in our resources, when we initialize our AWS provider, I've removed the profile here, and we'll see that, we'll set two environment variables to pass our AWS credentials during the build process.

So realistically, those are the only changes to our networking config. Now let's open up this Jenkinsfile thing. The Jenkinsfile is a way to declaratively define the pipeline that you want to run, and you can store it with the code that you're going to be running in the pipeline, which is, that's kind of convenient, everything’s in one place and it's checked into source control.

So this is the pipeline definition format. It's using a format that's defined by Groovy, which is a whole other topic that we're not going to get into, but that is the format. It's pretty intuitive, basically, we're defining a pipeline, and then we're putting the pipeline information inside curly braces. Because multiple agents can exist in your Jenkins environment, we are telling it to use any available agent, and since there's only one, that's fine. Then in the next block, we're telling it what tools are going to be available to the pipeline. And remember when we installed Terraform and gave it a name? Now we're referencing that tool, installation, and name to be available to this pipeline. The next thing are any parameters that you might specify to the build pipeline on build time. And there are two things that we want to specify. One is the CONSUL_STATE_PATH and the second one is the workspace we're going to use. And we are giving a default value for both. We're giving a default value for the location of the state, and then the second one is a default value for which workspace we want to use. So by default, unless we override it when we kick off a build, it's going to deploy to development. So those are the two parameters we're defining. Scrolling down a little bit, we have an environment block, and these are environment variables. The first thing is where is this Terraform executable?

And we do that by using the tool function and then telling it which tool we want to reference, that will return the location of that specific tool, and we'll store that in TF_HOME for a little bit further down in our config. Then we are setting TF_INPUT = 0 to let Terraform know, don't expect any user input here, then we're setting TF_IN_AUTOMATION = TRUE, so it has a value, it's going to make assumptions there. The next thing is TF_VAR_consul_address. So this is an additional variable that defines what the consul address is. In our case, it's going to follow the same as the backend, it's going to be host.docker.internal. For the log level, we're going to set that to WARN, so it's going to show anything warn or error. For the HTTP_TOKEN, this is where we put in the credentials before. We're using the credentials function and we're asking for Jenkins to return the networking_consul_token value and store it in this environment variable. For our AWS access and secret key, we're doing the same thing. These two environment variables work with the AWS provider to provide this information without hard coding it or submitting it through a variable. And then the very last environment variable is updating our path variable to include TF_HOME. So the pipeline will be able to run the Terraform executable without giving it a full path to the location. so that's all of our environment information. Let's scroll down into the stages of our pipeline, and you'll remember we saw the stages in the presentation, so now we're just being more explicit about these stages. Each stage has a set of steps in it.

So there's a steps block defining the steps to take within the stage. The first stage is called network initialization. So we're going to run the Terraform init process. Within the step, we have to tell it which directory it's going to be running this executable within. So we're going to run this within chapter 8 networking because that's where our config lives. First, we're going to output the version of Terraform we're using because that's just good to have in your console logs for future reference, and then we're going to run terraform init, just like we always do, and specify a argument, and we're setting the path to whatever stored in the parameters up above for the console state path. And that's the way that you reference one of the parameters that you've defined for this pipeline. As we saw in the presentation, we want to validate our configuration after it's been initialized. So once again, we're running out of the chapter 8 networking directory, and we're simply running terraform validate.

If this fails or the stage before it fails, it's going to fail the whole build because we're not catching the error in any way. In the next stage, we are going to catch an error, and for a very specific reason. The next step is to plan, and we need to select which workspace we're going to be running this plan in. Again, we're using directory m8/networking, and then we're going to run a script. And the script tries to run terraform workspace new, and then the name of the workspace in our parameters. Now, as we know, if we've already established, say the development workspace, which we have, that's going to throw an error saying the workspace already exists. That's if it does already exist, all we want to do is select that workspace instead. So we have a catch statement below the try, and if it does error out, that means the workspace already exists, we'll run terraform workspace select, and select the workspace. Once that's done, we'll run terraform plan and then create a tfplan file. Now, what are we going to do with that file because now we're moving to another stage? We can stash that file by using the stash command, and it basically squirrels it away for whenever you want to refer to this stash file later.

So we're going to call it and say that it includes that file. so now we move into the last stage of things which is NetworkApply.

The first thing that we're going to do within NetworkApply is we're going to start a script block, and we're going to set a variable apply = false. Here we get into the meat of whether or not we're going to apply the configuration that was generated by the plan. So we're going to create an input message asking the end user to confirm the apply or cancel it. If they confirm the apply, we'll set apply = true. If they cancel it, it will throw an error, we'll set apply to false, and then within the context of the directory m8/networking, we're going to run terraform destroy The reason that I'm doing this is to make it simpler for cleaning up afterwards. So when you've done all these exercises and you want to tear all these environments down, you can just rerun the pipeline and click on Cancel, and it'll delete everything for you. That's pretty convenient. But generally speaking, we're going to want to confirm the apply, which takes us down to if apply is true, which it will be if we've confirmed applying the config, within the context of directory m8/networking we'll unstash our tfplan file and then we'll run terraform apply and that's the end of our pipeline. So those are all the stages. So now that we've reviewed our Jenkinsfile, the next step is to create this pipeline in Jenkins.

Chapter 11 How to Create Networking Pipeline

We've reviewed our networking configuration, we've gone through the Jenkins file, we're ready to set up our first pipeline. We are going to go into New Item, and what are we going to call this item? Why, let's call it In fact, that's already in my remembered items. And the type of item we want to create is a pipeline. We've given it a name, we've selected pipeline, we'll click OK. It takes us to the configuration page.

So let's scroll down here, and there's a few things that we do want to define here. Down under Pipeline, we have Definition, and we're going to use a pipeline script, but where do we have that script? We have that in source control. So let's go ahead and do Pipeline script from SCM. And now it's going to ask us where's this SCM, what type is it? Well, the SCM is Git.

So we're going to select Git here, and now it needs a little more information from us. The first is the repository URL, where is this Git repository located? Now if we go back to Visual Studio Code and go into m8_commands, I have the URL of my repo that has in it, so we can go ahead and select that as the repo. If you've forked my repository and you have your own, feel free to substitute your own URL here. Going back to Jenkins, we'll go ahead and paste that. Because this is a public repository, we don't need any credentials for it. Under Branches to build, it's asking which branch do you want to build against? Now because this is a redo of an existing book, I actually have a separate branch I've been using, which is called v2 because this is version two of the deep dive. Once I've committed everything and moved it over it will be the master branch, but for right now, I'm using v2. You might have a similar thing if you're messing around with different potential configurations and you want to use a different branch to build against. So I put in v2 there. And now if we go into Script Path, where can it find this Jenkins file that you wanted to use for the pipeline? So I'll jump back to Visual Studio Code, and under our Script path, we know that it's called Jenkinsfile. So go ahead and copy that value and paste it . And the last thing is I uncheck Lightweight checkout because that has caused some weird errors in the past. Your mileage might vary, but I know that it works for me, so I uncheck it. Let's go ahead and save this whole thing here, and now our pipeline is, well, it's ready to go. What's the last thing to do? It's to run a build for this pipeline and that is exactly what we're going to do next. At this point, we have reviewed our configurations. We have prepared our pipeline. The next thing to do is actually run a build. We're going to click on Build Now and see what happens. So I just clicked on Build Now, and we can see down at the bottom, we can click on Console Output for build 1 and see what's actually happening in real time. The first thing it has to do is get the current repository and clone it down to where it's going to be running this whole thing. So that's the first thing that it does. Remember, that's what's in the pipeline, and then it actually starts walking through our pipeline process. This is maybe a little hard to follow because it's going to be a lot of text scrolling by, so let's go back to And we can actually watch the various stages here as it goes from one to the next the next. So right now, it's in network initialization, which means that it's probably copying all the chapters and plugins that it needs to use.

Now it's moved into NetworkValidate, so hopefully, our configuration is valid, and it looks like it was. It jumped right to NetworkPlan, so now it's running tfplan and planning out its whole process. And then hopefully, it's going to stop right after this on NetworkApply and wait for us to give some input. It looks like it did. So now we're in NetworkApply. If we hover over that, it tells us it's waiting for us to say Apply Config or Abort, but we haven't looked at the plan yet. So let's scroll over, and we're going to go back into the console output and kind of see what happened here. Now if we scroll up, we can see in the output.

We have already deployed the development environment, and we didn't make any changes to it, so there shouldn't be any changes needed to be made by Terraform. So just looking at that, we know that things have worked properly. So if we scroll down, within this context, we can go ahead and click on Apply Config. And because there's no changes, it's not actually going to do anything when it applies. It'll just show us the outputs that are in the configuration again and finish with success. If we go back to deploy, we can see that all the stages have completed successfully. The first time you set up a pipeline, it's not going to be all green. There's going to be a lot of red. Don't let that get you down. When I first started setting this up to get this demonstration working, it was a sea of red. But what happens if we want to build to a different workspace, if we wanted to do this against QA? Well, you'll notice that Build has changed to Build with Parameters. And when we click on that, it allows us to update the parameters. The reason we didn't see that before is it hadn't yet pulled down the Jenkins file from our repository, so it didn't know what the parameters would be. That's a weird thing about using a Jenkins file is the first time you run it, it's just going with the defaults. Now if we wanted to set up QA, we could simply change the WORKSPACE to qa, and click on Build, and now it will kick off a new build using the QA workspace instead. So we can watch it walk through this entire thing right up to NetworkApply as well. And since we already deployed the QA environment, there should be no changes for this one as well. So it's gotten to NetworkApply, and it's pausing for acknowledgement. I'll go ahead and click on Apply Config, and it will go through the end of this process, and it’s done. It took 6 seconds.

So, it didn't deploy anything. It just made no changes, and the same console output was there. So we have now successfully gone through the networking process. The application team probably wants to get in on this too, so let's walk through setting up the application pipeline as well. We have successfully deployed our networking pipeline and now it's time to set up the application deployment pipeline. So let's go ahead and create a new item here, and what are we going to call that new item? We'll call it And, it's a Pipeline type, we'll click on OK here, and if you're following along. This probably looks really familiar to you because the process is basically the same.

We're going to be getting our Pipeline script from SCM, we're going to select Git as the source control management type. Our Repository URL we can go back to VS Code and grab it, we'll go ahead and grab it here, copy it, go back, and paste it. We don't need any credentials because it's a public repo, and I'll go ahead and change the branch to v2. And finally, I'll go down to the Script Path, and the script path for the application, if we scroll down here, is going to be /applications/Jenkinsfile, so it is a different Jenkins file that lives with the application configuration. Go ahead and paste that script path and uncheck Lightweight checkout, and click on Save, and that's it, we have created our pipeline. Before we run this, there are a few changes to the configuration, so why don't we review those changes very quickly. We've set up our application pipeline using the Jenkins file that's in the applications folder. Before we move any further, let's see what's different about the application's configuration. Once again, if we look at the backend, we can see that the address to Consul has been updated as necessary. If we go into datasources, there is a change . For our terraform_remote_state networking, we have to give a different address for Consul just like we did for the backend.

So that address has been updated as var.consul_address and that's really the only changes. And if we go into resources just like we did in the networking configuration, I've removed the profile for providers so we can just provide that through AWS access key and secret key.

One thing I do want to point out is if you wanted to use different AWS credentials for your application deployment versus your networking deployment, which you might want to do in a production environment, the easiest way to do that is just to create additional credentials within Jenkins, and then in the Jenkins file under the AWS_ACCESS_KEY and secret key environment variables, just reference those different credentials in this applications Jenkins file and you'll be all good.

Speaking of this Jenkins file for the applications, it's basically exactly the same as the networking one because all the steps are basically the same so I didn't have to make a whole lot of changes. , let's go back to Jenkins, and we'll go ahead and deploy the development and QA versions of this application configuration. We are ready to build our pipeline. We'll go ahead and click on Build Now, and this should look very familiar because it's very similar to how the networking pipeline worked. We'll go ahead and click on the console output, and we can just follow along as it goes through all the various steps of what it's doing here. So it's downloading all the plugins for the various providers, and it just ran terraform validate.

And let me just scroll up here, and you can see terraform validate there. And if we scroll down a little bit, it's buried in all these different error and warning messages. It says Success! The configuration is valid. So good news, our app configuration is valid. That didn't throw an error. Scrolling down some more, it tried to create the new development environment, but we can see a few lines down, workspace development already exists. So instead, two lines down from that, we select the development environment.

Scrolling down some more, we go ahead and run terraform plan for the application environment. And if we scroll down all the way to the bottom, you can see that we got No changes. Infrastructure is and that's because we've already deployed the development instance of our application configuration.

So we can go ahead and click on Apply Config. There's nothing for it to do. It will give us the outputs of the configuration if there are any and say Finished with SUCCESS. That was a little bit boring. Let's go ahead and go back to and ah, now we have Build with Parameters again. We can go ahead and click on that, change our WORKSPACE to qa, and click on Build, and A second build has kicked off, and this one is for the QA environment. We know that the networking QA environment is ready to go, so now our application QA environment can deploy on top of that. Now this may take a few minutes for it to run through the whole process.

So I will go back to the view of the stages, and we can watch as it progresses through the various stages here. And we can see, if we look at this, it has already made its way to ApplicationApply. So that was pretty quick. Let's go ahead into Console Output and see what's going on down there. Scrolling all the way down, we can see here is the execution plan that's been generated, so it is creating all the resources we would expect for a QA, and it is a laundry list of resources. If I were the approver, I would go through these and make sure it matches up to what I expect, especially if I'm making changes to the environment.

And if everything looks good to me, I can scroll all the way down to the bottom and click on Apply Config. And now it moves to the apply stage where it will create all the resources that were part of plan. This does take a while. It's creating an RDS instance, and it's creating all the autoscale groups, etc. You probably don't want to sit around and watch that whole process, but I leave that as an exercise to you to go through that process. And if you're really feeling like you want to, you could actually deploy the production environments for networking and applications as well. That would be pretty cool. In summary you're thinking automation is hard. There is a lot of stuff to digest, and you're not wrong. Automation is challenging. Automation is also very rewarding, when it works, it's great, and the nice thing is, once it works, it kind of just keeps working as long as you don't change anything else, so automation in that regard is very rewarding. Another thing is as you walk through automating your environment, you will find all the assumptions you made about that environment and have a better understanding of the environment as a whole. Another thing is to focus on the concepts that were introduced in this chapter around automating with Terraform and not on the specific tools. Jenkins is great, GitHub is great, but there's a lot of other options out there. The concepts stay the same even when you move to different toolsets, so keep that in mind. Next, we are going to get into the world of configuration management because there is a lot of things that Terraform is really good at. Configuration management is not one of those, so how can we hand it off to something else? Well, we could hand it off to Ansible and that's exactly what we're going to do coming up in the next chapter.

Chapter 12 How to Integrate Configuration Managers

Terraform excels at deploying infrastructure, but when its job is done, you might want a configuration management application to take over and prepare the resource for code deployment. Working in tandem, Terraform and a config manager like Chef, Ansible or Salt can get the resource configured precisely as it should be and control configuration drift over time. In this chapter, you will learn how to add a configuration manager, Ansible, in this case, to your Terraform configuration. What are we going to cover in this chapter? Well, since we're going to be dealing with a configuration manager, it would probably be useful to understand what a config manager does and how it goes about doing what it needs to do. Once we have a good grip on that, we're going to talk about different deployment patterns that exist that may or may not incorporate a configuration manager. And finally, we are going to look at our ACME environment and figure out how we can integrate a Config manager, in our case, Ansible, with the existing configs that we have in Terraform. First, let's talk a little bit about what configuration management software actually is. There's a lot of different configuration management tools out there, but they all should have some common elements among them. The first is identification. How does a configuration management piece of software identify which components will be managed? Usually, that's defined somewhere in a configuration file or an inventory file. I've got this list of servers and they're all supposed to be web servers and this is the configuration I want to apply to them. The next major thing is control. In order to properly apply a configuration to a resource, the config management software has to be able to control that resource. So the determination needs to be made, how that resource is going to be managed. The third component is accounting. I know I want a particular configuration applied to a resource, but first I need to know what the current state of that resource is before I make any changes. The accounting portion of the config management software is what's responsible for figuring out what the current state of an object is and then comparing that to the desired state. The config management software will then make the necessary changes. The last thing that is important to config management is verification. Once changes have been made, the config management software needs to validate that the changes that it made line up with the desired configuration that's stored wherever that configuration might be. There's a bunch of tools out there that do configuration management and all of them should share these components in common. Now that you have a understanding of what configuration management software should do, let's look at some of the options to consider when you're deploying configuration management. The first thing to consider is how does Terraform pass off the baton to configuration management? It's set up your infrastructure and now it's ready to say, I'm done, here configuration management software, you go ahead and do your thing. In the case of something like Chef or Salt, there's actually a provisioner that already exists in Terraform you can take advantage of to do that handoff. Other things like Puppet and Ansible, you're going to have to write some sort of shell script or do something similar to handoff from Terraform to the configuration management software. The next deployment question is do you push the configuration or do you pull the configuration? When I say push the configuration, that means there's some sort of centralized server that has the desired configuration and an inventory of nodes to apply that configuration to and it goes out and pushes against each node. Ansible is a good example of that deployment methodology. When you run Ansible, you run it on a local box, you give it a list of remote hosts, and a configuration you want applied to those hosts, and it uses something like SSH or WinRM to push that config to each node. A pull configuration is a little bit different. In a pull configuration, you have a centralized repository that has configurations in it, but each individual node is responsible for pulling that configuration down and applying it locally. Often that's done through some sort of agent. So for instance, Puppet has an agent that you install on each node and then you set that agent to run on specific intervals to pull the current configuration from a Puppet Server. The last thing to think about is sort of related to push versus pull and that's whether to go with a centralized or distributed configuration. So a centralized model would mean that there is a centralized inventory and configuration server that holds all that information, and often that works in tandem with some sort of push configuration, though it could work with either. Alternatively, you could have a decentralized or distributed configuration management deployment. In that case, the configuration could be stored in some sort of shared repository, but it's a distributed repository and there's no centralized server that is forcing configuration to happen on a certain cadence. In the case of a distributed configuration, that scales much easier because you don't have to rely on the bottleneck of a centralized server, but you are giving up some of the control of that centralized server when it comes to things like auditing and logging. Now that we have a good idea of what configuration management software does and how it does it, let's talk about some deployment options. What are some ways that Terraform and config management can work in tandem to deploy a config to a target environment? In traditional provisioning, you would typically create an image or use an existing image, and you can think of that image as either an AMI in AWS or a template in VMware. It's the base image that you work off when you deploy a new instance. That instance is just going to be a basic configuration, probably just the operating system patched up to a certain level. On top of that instance, you're going to provision the application, and you're either going to do that through Terraform, you could use the provisioners that exist in there, or you could hand off that deployment of the application to a configuration management piece of software. That would do the initial provisioning of the application and then continue to manage the configuration of that app going forward. That is a fairly traditional pipeline. I did want to bring up an alternative version of that provisioning pipeline that may or may not include a config manager, and it's called immutable deployment. Immutable deployment implies that once something is deployed, it doesn't change. It's not mutable. You don't make any changes to it. The pipeline for immutable deployments starts out very similar. You create an image, and you deploy an instance from that image. The main difference is that image already has the operating system patched, and it has the prerequisites for the application installed and the application itself. All it needs is a little bit of configuration info to know where it fits into your infrastructure, but basically, that instance is ready to go from the moment it fires up, and that's very similar to a deployment. You already have the application installed with some config information. If you're deploying this with something like Terraform, you might not need to hand it off to a config manager. When it comes time to patch or update that application or the underlying layers, you don't patch or update the instance. Instead, you update the image. That image is the immutable truth of how the deployment should be. The whole point of an immutable deployment is that you never alter the instance once it's spun up. Instead, you go back and update the image, and once the image is updated, you redeploy the instance with the update. And typically, you would do this through a rolling upgrade where you take an instance out of service, destroy it, and create a new version of the instance from the updated image. Those are two different deployment pipeline patterns. The immutable deployment is becoming more common because of containers in the way that they are deployed, but the same principles can be applied to virtual machines. Now let's take a look at how ACME is approaching their application deployment pipeline. We've been very active here at ACME getting things set up, we’ve got our infrastructure setup. We've even got our EC2 instances deployed, and what needs to happen beyond that? Well, we need to lay down the application. So far, we have our infrastructure config, and that covers both networking and application infrastructure, and we've been using Terraform to deploy that configuration to the cloud. One of the components in that infrastructure config is what's called a launch configuration. This is the portion in AWS that defines how to configure the EC2 instances when they're spun up from an autoscale group. At ACME, what they want to do is when an instance is spun up, it will use an Ansible playbook to install the application. Where is it getting this Ansible playbook from? The application developers have their application code. They're putting it together inside of an Ansible playbook, and they're pushing that up to a GitHub repository. When that EC2 instance fires up, it knows the address of that GitHub repository, and it's going to pull that playbook from the GitHub repository and run it locally on that EC2 instance. So that is our goal. That is how we're going to get an application deployed on our instances using Ansible. So let's go over to Visual Studio Code and get started.

Just as before, you will need Consul running in the background if you're going to be walking through this with me. If I open up my terminal window and I go to my first terminal, I already have Consul running in the background, so I'm good. But if you don't, go ahead and get that going right now. Now we'll go over to terminal number 2, and we need to set our CONSUL_HTTP_TOKEN. We're going to start with a token. So I'll go ahead and grab that token from where I have it stored. Paste it, and now I'll take this whole environment variable command, copy it, and paste it down the terminal.

My CONSUL_HTTP_TOKEN is set. I can now initialize the back end for the networking, so I'm going to go ahead and go into the networking folder. And as far as our networking configuration goes, it hasn't changed at all. It's exactly as it was before. So I'm going to go ahead and initialize the back end so we can get things started. Once the initialization is done, we are going to select the development environment.

I already have my development environment deployed. If you don't and you're following along, now is when you will deploy that development environment. So I've selected the development workspace. I'll go ahead and run a plan here. Paste that down here. When the plan is finished calculating, it should say that no changes are required because I already have this network environment deployed, and nothing has changed about it since the last time I used it. , great. My infrastructure is up to date.

If you haven't deployed this or you destroyed it earlier and you're ready to redeploy it, go ahead and run the apply here, and you'll be up to speed for the next step. The next step in our process is to examine what's going on inside the application configuration, and then we'll go ahead and get that application configuration deployed. At this point, we have our networking configuration deployed in the development workspace. We're good to go with that. Now it's time to take a look at our application configuration and see what's different about it. There's not a ton that has changed, but let's go ahead and expand the applications directory, and we'll take a look at a few entries here. It probably makes the most sense to start out in datasources. We'll go ahead and open up datasources here, and we'll scroll up to the top. And, there's a new entry for a template file called userdata. Before, we were simply installing NGINX and not doing anything else. But now, we're doing a little bit more.

We are basically constructing the command to run the Ansible playbook. So we're going to hand it a number of variables that go into that Ansible playbook. Basically, we are deploying WordPress for this example, though it really could be any application that can use web servers in a database back end. We're passing it some variables to construct a rendered template from userdata.sh. We're passing it the hostname of the database, the name of the database itself, the username to access that database, and the password for that user, and then finally, where on GitHub can it find the playbook repository where it's going to be pulling the playbook from?

If we go over to the actual template that's in templates/userdata, let's take a look at that. Go ahead and expand and open it, and we have a few commands here. One thing to note is this deployment actually uses Ubuntu instead of Amazon Linux because it turns out it's actually easier to get WordPress installed properly by using Ubuntu. The point is these are the instructions that are in the userdata templates. And the first three instructions are simply to run update and then install both Git and Ansible because we're going to need those two tools. Then, we make a directory where we want to store our playbooks that we pull from the repository, and we use git clone to clone that repository to that directory.

Now that we have those files, the last thing to do is run which executes a playbook. We point it at the playbook.yaml file, and then we have to give it a list of hosts to apply this playbook, too. In our case, we're just applying the playbook to localhost. So in the hosts file that I have referenced here, /var/ansible_playbooks/hosts, it simply has localhost in there, and it tells Ansible it's running locally on that host. Scrolling over a little bit more, we have to pass it some variable values to use inside that playbook, and these are the variable values that we were passing to the template in the datasources file. So we're passing it to the db_hostname, the db_name, the db_user, and lastly, the db_password so that WordPress can go ahead and get the WordPress database all set up.

So that's everything that's in userdata.sh. Go ahead and close that and go back to datasources. If we scroll down to the data source "aws_ami" "ububtu", this is where we're grabbing the AMI ID for the Ubuntu image, depending on what region you're in.

And we're going to use this image instead of the Amazon Linux image that we were using in previous configurations. Now that we've looked at the datasources, the next thing to look at is resources. We'll go ahead and open that, and all we have to do is scroll down to our aws_launch_configuration.

This is where we're making the change to use the rendered template as opposed to just giving it the user data file by itself. So under userdata, we are now referencing the template file, userdata.rendered. There's one more change, and that's under variables because we need the URL of our Ansible playbook repository. If we scroll down to the bottom here, there's the variable defining our playbook_repository. And in terraform.tfvars, I have that variable being defined. This is where you can find the Ansible playbook repository. You can fork this if you want and replace it with your own personal repository, or you can continue to use mine. I'm not going to change it at all, so it should be safe to continue using. So those are all of the configuration changes that have been made for the application configuration. Why don't we go ahead and deploy this? We have reviewed our application configuration, we know that when the EC2 instance comes up, it's going to run an Ansible playbook with values that we are giving it from the Terraform configuration. Now, in order to run it, we have to change our CONSUL_HTTP_TOKEN to Samantha since she's the one who has access to the application config data. So I'll go ahead and open up the terminal, and I'm going to grab Samantha's token from my Notepad, and we'll go ahead and paste it . That's Samantha's token, and now I'm going to update that environment variable to Samantha's token.

Now let's go up into the application's directory, so that down there, and we'll go ahead and initialize the back end as we usually do. We'll go ahead and copy and paste. My state has had a development workspace before. So rather than creating a development workspace, I'm simply selecting one.

If you're somehow in a state where you've never created the development workspace before, just change the command from select to new and create the new workspace that you want to use.

Go ahead and copy this here and paste it down here, so I select the development environment, and then we're following the standard Terraform protocol. We're going to run terraform plan to plan out all the changes. The plan is going to take a while to calculate because there are a significant number of resources here. It looks like it has finished.

It is ready to add 18 new resources. We'll go ahead and grab the terraform apply command and paste it down here, and It is going to run through its apply process. This is going to take a while because it does have to spawn an RDS instance in all of these EC2 instances. That's going to take a significant amount of time. It looks like our application configuration has successfully deployed.

You might be wondering what's in that Ansible playbook that we just executed on those EC2 instances? Why don't we take a brief look at that Ansible playbook. In the root directory, there's a playbook.yml file, and this is kind of where it all starts. It's the one that we're pointing at with our command. If we take a look inside this file, it lists a number of hosts, so this is where the configuration should be deployed, and we're calling those hosts webservers.

And then under roles, were telling it what should be installed on those hosts. So we're installing nginx, wordpress, php, and postfix. Those are the four things that are being installed. Where do those hosts come from? If you remember in the user data, we used the command and specified a host file. Well, if we go back and open up the hosts file, this is what it's going to be looking at. We're defining a category of hosts called webservers, and we're saying that the localhost is one of those webservers, and we're letting Ansible know it doesn't have to SSH to that host, it is a local connection.

So that's what's in the host file. Going back to the playbook, we had those four roles, and the roles are defined inside the role directory. Here are our four roles: nginx, php, postfix, and wordpress. I'm not going to go through the contents of each of these because this isn't an Ansible book, we're trying to learn about Terraform and some integrations, but basically, if we take a look here, each folder has handlers, tasks, and templates in it, and the tasks really define what actions to take to get, in this case, nginx installed on the server. Go back up a couple folders, there's obviously a lot more. I would highly recommend going to Ansible's site or going through some of the Ansible books to get a better idea of how Ansible works. But in the meantime, why don't we check out our deployment of the application configuration? Now, let's jump over to the EC2 instances that we have running in AWS. I'll go ahead and click on Running instances, and I should have two instances . These are the two EC2 instances that were spawned from that launch configuration.

Because this is a fresh install of WordPress, WordPress, if you go to the main page, will redirect you to the configuration page. So in order to get this going, what we actually have to do is go to one of the instances directly. So what we're going to do is grab this public IP address down here and we'll open up a new browser window, and we have to go to http, that address, and then we have to go to and this will take us to the installation screen for WordPress.

Once you walk through the installation of WordPress and it's up and ready to go, then the load balancer will see that both instances are ready to accept traffic and you can go to the load balancer URL instead. So we have done it, we have built our networking with Terraform, we've built our application infrastructure with Terraform, and we've handed off to Ansible to deploy an application on top of all that infrastructure. In summary we covered that Terraform, it doesn't do everything, it's not the end all and be all, and it was never intended to be. Terraform is for automating the deployment of infrastructure, that's what it's for. It doesn't try to be more than that, and in fact, they've dialed back the use of provisions for that reason. It's really better to hand off to another tool. Configuration management is that other tool. When Terraform is done, if there's additional things to do, or you want to maintain configuration over a long time inside of a virtual machine, that's what config management is for, and that's why you should hand it off from Terraform to something like Ansible. Ideally, you should just pick the best tool for the job rather than trying to shoehorn a tool into a job that it was never intended for, and we've seen that in this chapter. This is the last chapter in the book, so rather than talk about what's coming up in the next chapter, let's think about what you might want to pursue next when you've finished this book. One of the things that we didn't cover, and it's relatively new to the Terraform family, is Terraform Cloud. That's their hosted version of Terraform where you can remotely run Terraform commands, you can store configuration variables, you can create workspaces, it's actually a pretty cool and robust platform and it has a free tier, so you can sign up for that now and explore what's going on in Terraform Cloud. There's also now a Terraform Certified Associate certification from HashiCorp, so you could start studying up for that, there's some pretty good study guides out there, and get your Terraform certification badge. If you're interested in examining other products in the HashiCorp family, I'd recommend checking out something like Vault, Consul, or Nomad. Combined with Terraform, those sort of form the core of everything that HashiCorp is about. If you want to venture outside of the HashiCorp family, and it makes sense to do so, you can start getting into some different DevOps tools. You could improve your source control management with something like getting used to using GitHub and Git. You could dive deeper down CI/CD and get to know some of the different pipeline products a little bit better. You could jump into the world of containers and reed a Docker or Kubernetes book, or you could get deeper into this config management world with Ansible or Puppet or Chef. The sky's the limit when it comes to DevOps tools. Now let's review a few key points from the book. Number one, Terraform is part of a larger ecosystem, it's one tool amongst many. And we've gone pretty deep on it, you may not need to go much deeper to make good use of it within your environment. Speaking of ecosystems, through our examples with ACME, you saw how a single IT administrator is actually part of a much larger ecosystem, and she had to interact with developers and other teams, and even her own team. You are also part of a larger ecosystem, and Terraform gives you tools and ways to collaborate with that larger ecosystem. Another thing that we've learned throughout this book is while we're learning a tool, we're learning Terraform really well, it's more important to focus on the concepts behind Terraform and behind configuration management and CI/CD pipelines because those concepts are going to serve you well long after any particular tool has exhausted its lifespan. And lastly, you want to be passionate about the work you're doing, and I hope I've inspired a passion for you to go out and start using Terraform to build something remarkable.

BOOK 10

TERRAFORM

CLOUD DEPLOYMENT

AUTOMATION, ORCHESTRATION, AND COLLABORATION

RICHIE MILLER

Introduction

In this first book, you are going to discover what is Terraform Cloud is aall about. What are the components of Terraform Cloud? What services does it offer? The first chapter will focus on getting to know Terraform Cloud. We will start by going over the baseline prerequisites you should have in place before embarking on this book and some of the assumptions I am making about your level of knowledge when it comes to Terraform and Infrastructure as Code. This is not a book for complete beginners. This book is focused on getting you up to speed with Terraform Cloud. Speaking of which, next, we will dig into the core concepts behind Terraform Cloud, including what services and features are available, why you might choose to migrate to Terraform Cloud, and how much this whole thing will cost you. We'll finish up the chapter with the introduction of our scenario. You are going to be a member of the ACME Cloud architecture team assisting with the adoption of Terraform Cloud. We'll learn how to use Terraform Cloud's services and features through that lens. Before you get started on this book, I wanted to layout some required prerequisites, nice to haves, and assumptions I am making about your level of experience with cloud technologies, Terraform and managing Infrastructure as Code. As I already mentioned, this is not a book for the complete cloud novice. If you've never used Terraform before, you'll find this book obtuse at best. I recommend checking out my previous books on Terraform to quickly get up to speed. I am going to assume you're already familiar with the version of Terraform, including concepts like the Terraform CLI workflow, Terraform syntax, modules, and workspaces. We will be using AWS as our target for deployment in the book. I do not expect you to be an AWS expert, but you should at least know the common services in AWS including VPC and networking, EC2 and other compute, and IAM roles and policies. You should also be familiar with some basic concepts around software development as they apply to managing Infrastructure as Code. In this case, I am talking about version control systems, Git operations like push and pull, and continuous integration and delivery. Just like AWS, I don't expect you to be an expert in these concepts, but you should at least be familiar with them. I highly encourage you to follow along with the demonstrations and exercises in this book. I've found I learned best when I am with the technology and not just watching a video or reading a book. In order to follow along, you'll need the following prerequisites in place, a terraform cloud account. We'll actually create this account together in the next chapter, so don't worry about it for now. A GitHub account for creating and storing code. A free account is perfect. You won't need any of the paid features. An AWS account for creating infrastructure. I'd recommend creating a new account for this book where you have admin access. A code editor like Visual Studio Code, although you can use whatever editor you prefer. With these assumptions and prerequisites out of the way, let's dig into what Terraform Cloud actually is.

Chapter 1 Terraform Cloud Fundamentals

What is Terraform Cloud? Terraform Cloud is a hosted service provided by HashiCorp that expands on the core functionality of Terraform open source by adding features such as storage, remote operations, process automation, and more. Terraform Cloud was born from HashiCorp's Terraform Enterprise product, which is deployed rather than being hosted. For most intents and purposes, Terraform Cloud and Enterprise are identical in nature. We will dig into some of the subtle differences as we go through the book. For now, let's take a closer look at the features and services available in Terraform Cloud. Terraform open source represents the CLI experience you are already used to. Terraform Cloud expands on that experience by offering an opinionated implementation of advanced workflows and operations. For instance, Terraform Cloud provides data storage as part of the platform. If you've had to set up your own data storage on something like Amazon S3 or Azure Storage, you know that managing and maintaining that storage is one more administrative task you don't need in your life. Terraform Cloud provides a managed location to store state data. In addition to storing state data, Terraform Cloud can also execute remote CLI operations for you using hosted agents. Once a Terraform configuration is set to use Terraform Cloud, operations like plan and apply are executed through Terraform Cloud on a hosted runner agent with the results of the operation streamed back to your terminal. You also have the option to host your own agents. HashiCorp's hosted agents run on their infrastructure, which doesn't have access to your internal network. If you're using Terraform to manage an internal VMware or OpenStack deployment, Terraform Cloud agents can be the link between Terraform Cloud and your data center. The ability to run remote operations also empowers Terraform Cloud to run operations outside of a typical workflow. Remote operations can be triggered by an event in a version control system or called through an API. Your Terraform operations can be automated using a workflow that follows a GitOps pattern through VCS or a custom pattern with the API. During the plan and apply operations, you might want to check your deployment to make sure it adheres to company policy. That is where HashiCorp Sentinel comes in. Sentinel allows you to define policy as code and evaluate your Terraform code against those policies. You can check to make sure resources are tagged properly, that SSH isn't open to the world, and that you aren't using a huge VM that will cost $10,000 an hour. That would be bad. Speaking of cost, Terraform Cloud can also look at the resources being provisioned by your code and give you a cost estimation of running that infrastructure. Although it cannot take into account things like bandwidth usage and storage utilization, it can let you know if a new deployment is going to cost you an arm and a leg. As your adoption of Terraform becomes more advanced, you may want to write and host your own chapters, as well as provide guidance to your teams on preferred chapters and providers from the Terraform public registry. The Terraform Cloud private registry enables both of these patterns. Finally, HashiCorp has developed integrations between Terraform Cloud and external providers, including ServiceNow, Splunk, and Kubernetes. Improved collaboration is one of the primary benefits of adopting Terraform Cloud. Teams within an organization can collaborate on projects together, leveraging remote state data and operations to keep individuals in sync. Management of Terraform Cloud starts at the organization level and is divided into workspaces. Each workspace represents an instance of deployed infrastructure managed by Terraform. Users and teams are assigned permissions to the organization and workspaces within the organization. A system of tags can be used to group together workspaces and check things like policy. With all of the features I've listed out, you might be wondering how much does all of this cost? Unsurprisingly, since Terraform Cloud is Software as a Service, it follows a subscription model. The good news is that there is a free tier which includes the most popular features. Currently, you can have up to five users, unlimited workspaces, remote operations and state data, private chapter registry, API access, and VCS integration. That's not bad for a free tier. The next step up from Free is the Team plan, which adds in the roles and team management mentioned in the previous section. You can create teams and assign permissions at the organization and workspace levels. Moving up from the Team plan, we have the Team & Governance plan, which adds the Sentinel and workspace cost estimation features. Finally, there is the Business plan, which includes Terraform Cloud Agents, single audit logging, and additional run concurrency. In essence, you can get started for free with up to five users, but the free tier lacks the advanced features of the paid plans. Two important caveats are worth mentioning here. First, Terraform Cloud is constantly evolving, so I recommend checking out the Plans page to see if anything new has been added to the tiers. Second, the cost of Terraform Cloud does not include the cost of running the actual infrastructure on your platform of choice. You still have to pay Amazon for that NAT gateway you provisioned and forgot to shut down. The primary difference between Terraform Cloud and Terraform Enterprise is where the software is hosted and how you're charged for it. When it comes to Terraform Cloud, the software running Terraform Cloud is hosted and managed by HashiCorp, and there are tiered plans that we just went over available on a subscription basis. Anyone can easily sign up for Terraform Cloud. Terraform Enterprise is and managed by you. You will be running the software on servers in your data center or a private cloud or wherever you want to run that software, but you are responsible for managing it. There is also only one plan available, and you'll have to talk to someone at HashiCorp to purchase Terraform Enterprise. Aside from that, the features of the two products are almost identical. What does it look like to have a business adopt Terraform Cloud? Good question. Let's take a look at a scenario.

In this book, you will be an employee of ACME, a global risk assessment company. We will use this scenario to adopt and evaluate features in Terraform Cloud. Welcome to ACME! You've recently joined the company as a cloud architect. ACME has started using Terraform for the deployment and management of

infrastructure supporting their primary risk assessment application. They are pleased with the results and would like to expand the use of Terraform to encompass other application development teams. To support a collaborative environment, ACME would like to adopt Terraform Cloud for new projects. Speaking of which, as a new member to the company, you are joining a development team focused on a new line of business application, Donovan. ACME would like you to use Terraform Cloud for the deployment of infrastructure to support the application. Once Donovan is successfully ACME would like to migrate the existing Terraform managed application, Donovan's Team, to use Terraform Cloud as well. You will support the Donovan's Team in migrating from Terraform to Terraform Cloud, including setting up teams and permissions. As you progress in your adoption of Terraform Cloud, ACME would also like to look into adopting some of the additional features including Policy as Code with Sentinel, workflows, and using the private registry. It's an exciting time to be at ACME, and you get to help them with a project. Before we move forward, let's quickly review what we covered so far. This is not a beginner book for Terraform. You should have experience with using Terraform open source before trying to adopt Terraform Cloud. Terraform Cloud is a managed service from HashiCorp meant to provide an opinionated experience that augments and enhances Terraform open source. In this book, you will be helping ACME adopt Terraform Cloud, first for the Donovan app team and then for the larger organization. Next, we are going to get started with setting up Donovan by signing up for a Terraform Cloud account and configuring the organization and workspaces.
Terraform Cloud is a hosted service that helps you manage and automate your Terraform code, but how exactly does it do that?

Before we start running our Terraform code, we first need to get signed up to use Terraform Cloud and create an organization. That is what we'll cover in this chapter. This chapter is going to lay the groundwork for the rest of the book. We are going to dig into the fundamental concepts and structures that underpin Terraform Cloud, first starting with organizations. What is an organization and how does it relate to Terraform Cloud? We will cover the settings in an organization, access, and authentication, and how users are related to an organization. Once we have a firm grip on organizations, we will learn how to interact with Terraform Cloud from the CLI. Your primary experience with Terraform has been through the CLI, and you can continue to use that experience to interact with Terraform Cloud. Without a user account and organization, you cannot deploy any code with Terraform Cloud. Let's take a closer look at both objects, starting with a user account. When you sign up for an account with Terraform Cloud, that account is associated with an email address you specified during Your user account lives outside of any particular organization, and it can be invited to be a member of multiple organizations. Authentication is handled by the Terraform Cloud service with a password and optional authentication. Optional, but you should consider it to be required. After signing up for a Terraform Cloud account, the next step is to either create an organization or accept an invitation to join an organization. What is contained within an organization? At its most basic, an organization is composed of workspaces that run instances of Terraform code, a private registry to house providers and chapters, and integrations with other services like cost estimation and Sentinel. The configuration options for an organization span many categories, and here are a few important ones to bear in mind. Teams and users allow you to invite new users to the organization and assign them to teams. Each user must be a member of at least one team. Teams can be assigned permissions at the organization and workspace levels. If you want to have access to more than just the owner's team, you'll need to upgrade from the Free plan to at least the Teams billing plan. In our example, we'll start a trial of the Teams & Governance plan, giving us access to even more features. For instance, we'll be able to apply Sentinel policy as code at both the organization and workspace levels through policy sets. Workspaces in an organization can be assigned tags, which are a way to organize and label workspaces. Workspaces live in a flat hierarchy within your organization. You can't organize them into folders or organizational units to apply policies and permissions. Tags give you a way to organize and group workspaces and perform actions like filtering your view by tag or applying a policy set with Sentinel. Since Terraform Cloud is often used programmatically, you can generate API tokens for a user, organization or team. We'll discuss the differences between those token types more in a moment. Terraform Cloud performs authentication for users. The Business level plan gives you access to use single from a source like Okta to handle authentication. We won't cover that in this book. Instead, we'll rely on usernames and passwords with the possibility of adding authentication. If you plan to use Terraform chapters that are stored on a private GIT repository, you can provide SSH keys that have access to that repository. Each set of keys will be available to workspaces for use when they run remote operations. Finally, you can configure Terraform Cloud to connect to various version control systems like GitHub or GitLab for a VCS integrated workflow on a workspace. Why don't we sign up for a Terraform Cloud account and create an organization for ACME. As you'll recall, you are helping ACME adopt Terraform Cloud, specifically for the Donovan application team. For the time being, you're going to create a user account on Terraform Cloud and create an organization for ACME. In this demo, we will sign up for a Terraform Cloud user account using the ACME.xyz email address. Then we will create an organization called If you'd like to follow along, make sure you have an email address you can use to sign up for Terraform Cloud. Let's start with setting up a Terraform cloud user account. I recommend not using any production Terraform Cloud accounts for this book.

To sign up, I went to app.terraform.io. And now I'll click on the link to create a free account. On the next page, I need to give myself a username, email, and password.

I'll go ahead and fill it all in and agree to all of the terms and conditions, and click on Create account. Once the account is created, it will send me a confirmation email, and I can go to the email account, and go ahead and click on that confirmation link, and now my account is all set. The next thing to do is create an organization.

And since this is a user account without an invitation to join an existing organization, Terraform Cloud will helpfully suggest that I create a new organization, and that is what we'll do. I'll click on Start from scratch and go ahead and enter the organization name. We can create an organization called and my user will become the owner of the account. The organization name needs to be globally unique, so don't try to use if you're following along because it will already be taken. I'll go ahead and click on Create organization, and now the organization has been created. Before we do anything else, let's go into my user account settings and enable authentication.

I recommend you do this on any new account you create. So I'll click on the account and go to User settings, click on Two Factor Authentication, and in this case I will set it up using an application and click on Enable 2FA.

I'll scan the QR code and enter the authentication code from my authenticator application and click on Verify. And now authentication has been set up for my account.

With our user account created and our organization created, it's time to explore the Terraform Cloud UI and see where all the cool settings are.

Chapter 2 How to Explore Organization Settings

It's time to take a stroll through the Terraform Cloud UI. Fortunately, it's not that complex. Believe me, I've worked on some UIs that hide stuff all over the place and you're never quite sure where to go to find a setting or a toggle switch. Terraform Cloud is straightforward by comparison. Before I dig too deep into the UI, I do want to say that Terraform Cloud is a constantly evolving product.

The UI in this book might differ slightly from what you see when you log in. I wouldn't expect any major shifts, but just be aware things might not line up Across the top, we have the where you can select which organization you want to work in, followed by three major sections, Workspaces, Registry, and Settings. We'll deal with Workspaces and the private Registry in future chapters, so let's skip straight to the Settings area. Settings are broken up into the major categories of Organization settings, Integrations, Security, and Version control.

For now, I just want to highlight things we are going to be interacting with in this book. We'll start by visiting the Plan & Billing section. Since we want to have access to cool features like Teams and Sentinel, we are going to start a free trial of the Teams & Governance plan. The trial is good for 30 days, after which it reverts back to the Free tier.

You don't even have to provide a credit card or billing info, which is pretty sweet. I'll go ahead and click on the link to get started. It already selects the Trial Plan for me, so I can simply go down and click on Start your free trial, and now I'm on the Teams & Governance plan for the next 30 days. Below Plan & Billing is the Tags area.

Tags are created and assigned at the Workspace level. You can see Tags and delete them from Organization settings, but that's about it. We are going to add Teams and Users later, so we will skip those sections. Variable sets is something we'll discuss in the Workspaces chapter. You can enable Cost estimation for all workspaces, and once you start the trial, it is enabled by default. We'll discuss cost estimation later in the book. Policies and Policy sets are managed through VCS integration. We'll cover both in the chapter on Sentinel. In the Security category under API tokens, we can create organization tokens which have permissions to manage organization settings, but not do anything with workspaces. It also mentions user tokens and team tokens, which are exactly what they sound like. A user token has the same permissions as the user it is associated with and a team token has the same permissions as the team it is associated with. Under Authentication, we can configure some timeout information and also require authentication.

That option will only be available if you've already enabled authentication for your account. I'll go ahead and enable require now. Scrolling down a bit under Version control, the only area of interest for us is the Providers area. This is where we will go to connect our organizations to a version control system like GitHub or Bitbucket. When we enable a VCS workflow on the Donovan workspace later in the book, this section will become pretty important. This has been a nice tour of the UI, but you might be wondering how you can interact with Terraform Cloud at the CLI. That's a great question, and that is what we will cover next. We need to get our CLI configured to talk to Terraform Cloud, which means we have to authenticate to Terraform Cloud. How does the CLI provide authentication to interact with Terraform Cloud? The answer is API tokens. To get a token set up for your user account, you can run the command terraform login. The command will open up a browser window to procure a token from Terraform Cloud for your user. Simply give the token a name and copy the value produced.

The CLI will wait for you to paste the token, and then it will save the token in a plaintext user file in your home directory. The full path will depend on your operating system. The hostname argument is for situations where you are using Terraform Enterprise, otherwise the command assumes you are connecting to app.terraform.io. If you would like to log out of Terraform Cloud, the command terraform logout will remove the stored credentials. Note, this does not revoke the token you used, it simply clears out the credentials locally. You can go to your user account on Terraform Cloud to delete the token generated during login. Time to go set up our CLI in preparation for the next chapter. We will log in to Terraform Cloud and generate a user token that Terraform will store locally. For the demonstrations, I will be using

Visual Studio Code as my code editor of choice. On the left pane, I have all the exercise files associated with this book. In the center pane will be any code examples, and I can go ahead and bring up the terminal window down at the bottom, which is what we'll actually be using now. I'm going to run the command terraform login without any arguments. Based on the output, you can see that the API token will be requested from app.terraform.io. I'll enter yes to proceed, and that will redirect us to a browser.

Since I'm already logged into Terraform Cloud, the browser takes me directly to a token generation dialog box. I like to name my tokens after the system they will be used on. In this case, I will name it and I'll click on Create API token. Now it will give us the user API token that we will copy and paste into the Terraform login prompt. I'll go ahead and copy it now and switch back to the terminal and go ahead and paste it into the prompt down below.

And if I expand the screen and scroll up a little bit, we can see it retrieved a token for a user. It gave us a fun little graphic that got cut off, and it gives us some helpful places where we can get started. The value itself is written out to the credentials file in my home directory, and now we are all set to work with Terraform Cloud from the CLI. In this chapter, we got to know Terraform Cloud a little better by focusing on users and organizations. A user account is required to log into Terraform Cloud. A user can be a member of one or more organizations. Organizations are made up of workspaces, a private registry, teams, and integrations. The user account that created the organization will be initially set as the owner. To work with Terraform Cloud from the CLI, you'll need to generate an API token and store it in your home directory. Fortunately, the Terraform CLI has commands to manage the process. With our organization set up, the next big thing to do is set up a workspace for Donovan and deploy our application using Terraform Cloud and the CLI workflow.

Chapter 3 Terraform Cloud Workspaces

Once you've got a user account, an organization set up on Terraform Cloud, you're probably chomping at the bit to deploy some infrastructure. To ease you into the process, we are going to start with a familiar workflow that involves the CLI. Before we can deploy code with Terraform Cloud, we first need to set up an environment for the code to run in. That environment is going to be a workspace. We will see how to create a new workspace and configure its settings. Next, we are going to get some variable values defined for our code, but how do we do that? Just like Terraform open source, Terraform Cloud has a lot of possible options, and we'll explore each one. Finally, we are going to deploy the Donovan application infrastructure using the good old CLI. We'll see how code is sent to Terraform Cloud, parsed by the workspace, and used by a remote worker all from the comfort of our terminal. First, why don't we learn a bit more about workspaces? Workspaces are the workhorse of Terraform Cloud. They are where all the action happens. Terraform Cloud workspaces build on the Terraform workspace construct. Terraform workspaces share a common set of configuration files, but each workspace keeps a separate instance of state data. Terraform Cloud takes this essential concept and adds more functionality to the mix. A workspace contains three primary components, along with a bunch of settings that we'll get to in a moment. The first I already mentioned, which is state data. Each workspace stores its state data in Terraform Cloud, meaning it is using Terraform Cloud as a remote back end. This saves you from having to set up an S3 bucket or an Azure storage account to hold your state data. The state data in a workspace can also be shared with other workspaces in the organization as a data source. That's helpful if you decide to refactor your Terraform code into separate workspaces, but you still need to pass information between them. Each workspace also has its own set of variable values and environment variables. You can still submit variable values using the CLI or through specially named files, but it is nice to have variable values stored securely in the workspace itself. Workspaces can also make use of variable sets that are defined at the organization level. The last component is the real power of Terraform Cloud workspaces, remote operations and logs. Instead of a Terraform operation like plan or apply being run on your local machine, Terraform Cloud runs it on a hosted agent and saves the results and logs to the workspace. This gives you a standardized working environment for all Terraform runs and, more importantly, a single place to go and inspect logs when something inevitably goes wrong. When you create a new workspace, Terraform Cloud will ask what type of workflow will be used by the workspace. You will be presented with three options, CLI, VCS, and API, The CLI option is the simplest and closest to the way you would interact with Terraform open source. The Terraform code will include a cloud block defining the workspace as a remote back end and place to run operations. You can control the Terraform workflow from the CLI with standard Terraform commands like plan and apply. The VCS, or version control system, workflow is the most common option, but it also requires that you host your Terraform code in a version control system repository. Events on the repository will trigger workflows on Terraform Cloud. For instance, a commit to the default branch could kick off plan and apply operations in Terraform Cloud. If you need more customized automation and workflows than what is available in Terraform Cloud, you can use an API workflow to integrate Terraform Cloud into a larger automation pipeline. For instance, if you are already using Jenkins or Azure DevOps Pipelines to automate your Infrastructure as Code deployment, you can hook Terraform Cloud in to handle Terraform actions. Let's put this information to use by creating a workspace for the Donovan application team. But before we do that, let's first talk quickly about naming workspaces. A workspace name is very important because it should tell you what that workspace is used for. The name can be composed of alphanumeric characters, dashes, and underscores. The recommended naming convention from HashiCorp is the component, region, and environment. Depending on the size of your organization, you might want to preface it with the line of business or the application name. Workspace names exist inside an organization, so they do not have to be globally unique, only unique within the organization. You can include more information about the workspace using tags. The actual name should include what is unique about the workspace to distinguish it from other workspaces in the organization. For the Donovan application, we can use the naming This lets us know that the workspace is the Donovan application infrastructure running in and it is the development environment for their application. It's time to go create a workspace, and we will be creating a workspace for Donovan. This is our first foray into Terraform Cloud, so we will go with the CLI workflow option since it is the simplest and closest to our current process. Later in the book, we will migrate to a VCS workflow. After the workspace is set up, we will explore some of the settings and make sure our workspace is ready for us to deploy our Donovan application infrastructure. Creating a workspace is a simple affair. In our organization, we are on the Workspaces tab, and we can create a workspace by clicking on New workspace. I'll go ahead and do that now.

The first thing it will ask you is to choose your workflow, and this will determine the rest of the steps in creating the new workspace. We are going to select the workflow, and now it will prompt for a workspace name.

We will give it the name and we can give it a description of Development environment for Donovan application in Finally, we will click on Create workspace, and our workspace is now ready to go. Let's take a look at some of the settings while we're here. In addition to the three primary components and workflow styles, Terraform Cloud workspaces have a ton of settings that you can tweak. Why don't we look at some of the more common settings you might want to configure? Clicking on Settings under the General category, we've got some important settings that you may want to change. Scrolling down, we get to Execution Mode.

Although workspaces will run your operations remotely by default, you can turn that off by setting the execution mode to Local. This essentially makes the workspace function as a remote back end for state data. If you're not ready to take the plunge into remote operations, this is a way to ease into things. When a remote plan runs successfully, you may want to automatically run and apply as well. This is turned off by default. I recommend being cautious about turning it on for any production environments. You can set the Terraform version for a workspace, and that is the version the hosted agents will use for remote operations.

No longer do you have to wonder which version of Terraform you need to stand up your infrastructure or make sure your whole team is using the same version.

Workspaces make it easy to standardize and upgrade when the time comes. Scrolling back up, we'll move over to the Notifications category. When something happens with the workspace, you can trigger a notification to be sent to an email address, Slack channel, or webhook.

This is great if you want to be notified when a new plan needs to be approved or when an apply went horribly wrong. Going into the Settings again, we'll look at Team Access. Each workspace can be configured with team access permissions.

The Team Access category allows you to define which teams have access to the workspace and what level of access they have. There are levels of access like read, plan, write, and admin, or you can construct your own custom permission sets. Lastly, I want to point to the Destruction and Deletion area.

This settings area is dedicated to running a Terraform destroy or deleting the entire workspace. You can still destroy the infrastructure at the CLI using terraform destroy, but if you switch to a VCS workflow, this is the only way to trigger a terraform destroy. Before we deploy our code through the Donovan workspace, we will need to provide credentials to AWS and values for the variables. Why don't we check out how we do that? When you deployed Terraform code in the past, you probably defined values for your variables using a tfvars file, command line arguments or even environment variables, but that was when you were running things locally. Now that Terraform Cloud is running things remotely, you need a way to submit values. Let's take a look at your options. You can still specify values for your variables using local information on your workstation. You can either pass variable values through the command line using the or arguments, or you can set environment variables that start with TF_VAR, and then the variable name. Passing values using the file named terraform.tfvars will no longer work, but if you add .auto.tfvars to the end of a file name, then values defined in that file will be submitted to Terraform Cloud. At the workspace level, there is a dedicated variables area where you can define both variable values and environment variables. Both value types can be set to sensitive, ensuring that the value is not shown in the UI and not printed in the log data. This would be a good place to store your AWS credentials. Variable sets are defined at the organization level and can be shared with some or all workspaces in the organization. A variable set defines one or more Terraform or environment variables and their desired value. For variables that are common across multiple configurations, variable sets are a good option. You could potentially have the AWS account credentials for each environment defined in a variable set and presented to each workspace that needs them. With all these ways to set a value for a variable, you might be wondering what happens if a variable is defined in more than one place. A general rule to follow is closest to the configuration wins. Command line overrides workspace, workspace overrides variable sets, and everything overrides the auto.tfvars files. For more information, I recommend checking out the official Terraform Cloud docs for workspace variables. They include a whole table of precedence. Now that we know how to define variable values, let's get our Donovan configuration prepped. We will start by reviewing the Donovan code to see what variables are defined. Then we'll use that information to define our variable values using both a local.auto.tfvars file and the variable settings. In Visual Studio Code, we will take a look at the Donovan application infrastructure code by opening up the m4 directory, and within that, we have our configuration. Let's take a look at the variables.tf file that defines the variables will need for our configuration. Looking in variables, we have a variable named prefix that's required, a variable named project that is required, with the remainder of the variables set as optional, each with its own default setting.

We want to set the prefix, which is a naming prefix for all the resources that will be created, and we want to set the project name for the application project. And right now we're defining those in terraform.tfvars, as well as the environment name. So we have a prefix of dd, a project named diamond_dogs, and an environment equal to development. Using terraform.tfvars no longer works when you move to Terraform Cloud, But what we can do is make a copy of this file and rename it terraform.auto.tfvars, and it will be submitted as part of our plan and apply. We have our variable values set for the prefix, the project, and the environment. The other thing we need to do is set our AWS credentials, and we're going to do that over in the variable settings. Let's go over to the browser and do that now.

Chapter 4 How to Configure Workspace Variable Values

To configure the variables for a workspace, we simply go to the tab labeled Variables. So I'll do that now, and we will add some variable values for the workspace. I'll go ahead and click Add variable. And the variable type we want to add is actually an environment variable to submit our credentials for AWS.

We're going to need to define two environment variables, so I'll select Environment variable, and the key for this environment variable is going to be AWS_ACCESS_KEY_ID. Now let me go ahead and grab the value, and I'll paste it in the Value section and click on Sensitive to mark this as a sensitive environment variable, and then click on Save variable. And then I'll repeat the same process for the AWS_SECRET_ACCESS_KEY, which is the other environment variable we need to successfully authenticate to AWS using the AWS provider for Terraform. I'll go ahead and grab the actual value for the key, paste it in, and mark this one as Sensitive as well, and save the variable. One thing to note is once it's marked as Sensitive, we can write a new value to it, but we cannot look at the current value.

If I click on it and say Edit, it will allow me to change the value, but it will not show me the current value. I'll go ahead and click on Cancel. And now all the variable values are in place to deploy our infrastructure. The last thing to do is get our code ready to deploy to this workspace. How do we let Terraform know to use a Terraform Cloud workspace for code deployment? How does it know which workspace to use? And what commands do we use when working with Terraform Cloud? We'll start by checking out the cloud configuration block. Prior to Terraform 1.1, you would link up a Terraform Cloud workspace by adding a backend block to your terraform configuration block. The backend type was remote and the hostname for Terraform Cloud was app.terraform.io. You would also need to include the organization and the workspaces you wanted to use with the workspaces block. You could specify a single workspace with the name argument or a set of workspaces with the prefix argument. That prefix would be applied to each workspace you created. In Terraform 1.1, HashiCorp introduced a new configuration block type, the cloud block, which is nested in the terraform configuration block. Because Terraform Cloud is much more than a remote back end for state data, HashiCorp felt they should give it a different construct. The syntax of the cloud block is extremely similar with the hostname, organization, and workspace arguments. The hostname in this case is optional since Terraform will assume you are using Terraform Cloud. The preferred method is to use the cloud configuration block, although you may see the backend remote block in older configurations. Both the backend remote block and the cloud block support a single workspace or multiple workspaces. The primary difference is how the Terraform workspace commands interacted with Terraform Cloud. Since the cloud block is the preferred method, let's look at how you would configure a block to reference a single or multiple workspaces. The nested workspaces block takes one of two arguments. The name argument restricts the code to a single named workspace. If the workspace doesn't yet exist in the organization, it will be created when you run Terraform in it, but it will not have any settings or variable values defined for that workspace. If you want to use the same code with a different workspace, you will need to make a copy of that code and change the workspace name. The other option is the tags argument. When you initialize a Terraform configuration with the tags argument, Terraform will prompt you to select an existing workspace with matching tags or create a new workspace. You can use the standard Terraform workspace commands to manage workspaces with those tags. This option allows you to use the same configuration files with multiple workspaces without copying and pasting code. For either of the two options, you will still need to run terraform init from the command line after you add the cloud configuration block. For the named workspace option, the init process will either select an existing workspace by that name or create a new one.

For the tags workspace option, Terraform will allow you to select an existing workspace with matching tags or create a new one. When you use the tagging option, you can still use all the standard Terraform workspace commands to interact with Terraform Cloud. The workspace list will list all the existing workspaces with matching tags. The Terraform workspace new command will create a new workspace with the tags defined in the cloud configuration block. Terraform workspace select will allow you to switch between existing workspaces, and Terraform workspace delete will delete a workspace from Terraform Cloud. You might want to be careful with that one. One major improvement with the introduction of the cloud block in Terraform

1.1 was support for the terraform.workspace value. Previously, Terraform Cloud would always use the default workspace on the remote runner. The workspace name is used by the remote runner, making it available for use in your code. After initialization is complete, you should be ready to run, plan, and apply. Let's take a look at that process. When you are getting ready to deploy your Terraform configuration, the step after init is usually terraform plan. In Terraform Cloud parlance, this is known as a speculative plan. Terraform will zip up your configuration and settings and send it to Terraform Cloud. Terraform Cloud will take the file, combine it with any variable values and other settings, and send it to a remote runner. The runner will execute the terraform plan and stream the output back to your terminal. It also saves the terminal output and logs to the workspace for future reference. Terraform apply is virtually the same process, except now Terraform Cloud will run a terraform plan and prompt you to approve the plan changes before moving to the apply phase. You can skip the approval step by using the flag, just like you would locally. Let's go ahead and see this in action. We will start by adding a cloud block to our code with the proper organization and workspace. Then we will run through the standard Terraform process of initialization, planning an application to provision the Donovan environment. If we go back to the Terraform Cloud UI and take a look at our workspace, on the Overview tab for the workspace, it actually gives us some example code we can copy and paste. I'll go ahead and hit the Copy button now, and let's go back over to Visual Studio Code. And I'm going to replace the contents here with what I just copied and save the file, and now our configuration is all set and ready to go.

The next step is to initialize our configuration. So I'll go ahead and bring up the terminal, and I'm going to expand the view a little bit here so we can see all the output. I'll go into the m4 directory and run terraform init. Once it has completed initialization, we can take a look at the output that it gives us. We can see it says Initializing Terraform Cloud, then it downloads the provider plugins, and finally creates that lock file for the providers and chapters that you're using in the configuration. It tells us that Terraform Cloud has successfully been initialized, and tells us that the next thing to do is run terraform plan, which is exactly what we're going to do. Let's try running terraform plan. This plan is actually running on a remote runner in Terraform Cloud, but it's streaming the terminal output from that runner back to our terminal here so we can see what's happening with our plan. Once the plan command is complete, it tells us it's going to add nine new resources, and it gives us a helpful cost estimation of how much it will cost a month to run this application, which is a huge $8.63. Let's scroll up and review some of the other output from our terraform plan command.

Just below Preparing the remoteplan..., it tells us that if we want to view the run in a browser, we can visit a URL. Let's go ahead and click on that URL. And going over to the browser, it takes us directly to the workspace and the Runs tab, and it shows us the plan that was created. If we scroll down a little bit, in this run we can see who triggered the run, what type of run it is, it's a speculative plan from the CLI, we can view the output from the remote runner, and we can see the cost estimation. Below that, it lets us know that this is a speculative plan, so we cannot apply the plan. If we wanted to apply the plan, we should run terraform apply. Everything looks good to me, so let's go ahead and run terraform apply from the CLI. I'm going to go ahead and run terraform apply from here, and just like before, this is running remotely, and if we wanted to we could click on the link that it gives us to see the run in in the browser. Why don't we jump over to the browser and view the run as it goes. Back in the browser, let's scroll up and go to Runs, and we can see that in our run list there is a run that is in the planning phase. This is the one that we just kicked off. We can go ahead and click on that run and it will give us more information about the run. We can see that the plan phase has finished and the cost estimation has finished. If we expand the plan, we can once again see what it's planning to create.

And then it's saying that apply is pending. If we go back to the terminal, it's also waiting at the terminal for us to approve the changes. Whether we apply it from the browser or apply it at the terminal, either one is fine.

Let's go ahead and type in yes at the terminal. This will approve the apply, and if we go back to the browser, apply has now moved into the Apply running phase, and it is going through the process of building out the infrastructure that's in the Donovan application.

That's not a very complicated application, it's simply spinning up a VPC and an EC2 instance. If we give it a few moments, as part of the output, it will give us a URL we can visit to confirm that our application came up. Our apply is complete. It successfully created the nine resources and gave us the outputs that we need, including the URL to our new Donovan application.

If we go over to the terminal, the same information is also available in the terminal. So from a CLI perspective, not a lot has changed. We had to add a block to use the remote workspace and we were able to use all the standard commands for the CLI. If you want to tear down this infrastructure now, you can do that by running terraform destroy, and it will run a plan and then prompt you to approve that destroy. I'll leave that as an exercise for you. In summary, workspaces are the workhorse of Terraform Cloud. They are where the magic happens. There are three workflow options for workspaces, CLI, VCS, and API. VCS is the most common, while API is the most complex. The CLI workflow, however, is the simplest and most familiar. With improvements made in Terraform 1.1, the CLI experience still allows you freedom to manage variable values and workspaces while providing you the benefits of operations and logs. Next, we are going to shake things up by moving to the VCS workflow on our Donovan dev workspace. We'll see how to migrate from one workflow to another, configure options for the VCS workflow, and how to use the CLI for planning and testing.

Chapter 5 VCS and API Workflows

The CLI workflow in Terraform Cloud is certainly the most familiar, but it misses out on the automation and collaboration that exists in the more advanced workflows. Terraform is Infrastructure as Code, after all. We might as well apply some software development practices to it like source control and Git triggered operations. To kick off this chapter, we will first do a little refresher on what workflow options exist in Terraform Cloud, and then we will take a closer look at the API and VCS workflows. Since we will be adopting a VCS workflow, we will drill down into the details of what a VCS workflow includes, how you might migrate from a CLI workflow, and what you can still do at the command line after migration. Of course, our Donovan application is not meant for a single workspace. We need to support development, staging, and production. We'll see how the VCS workflow supports that scenario through a combination of branches and Git operations. In the previous chapter, we covered the CLI workflow extensively as a way to ease you into using Terraform Cloud. Now it's time to investigate the other two workflows. If you'll recall from the introduction of workspaces, there are three workflow options available when you create a new workspace. The CLI workflow is the simplest to set up and feels familiar to folks who are used to running Terraform open source. The VCS workflow is the most common selection in Terraform Cloud because it adds the ability to link workspaces to source control and automate operations. The API workflow is the most difficult to implement, but it is also the most flexible. Why don't we learn a little bit more about using the API workflow. The main reason to adopt an API workflow is that the options available in the CLI or the VCS workflow don't meet your needs. The API workflow requires you to interact with Terraform Cloud using its API. Libraries are available for Go, Python, Ruby, and .NET. When you work with the API, you will need to handle authentication using the appropriate API token for your actions, whether that's a user token, team token, or organization token. Rather than running a CLI command or using an integrated webhook, the API workflow requires you to trigger remote operations through an external system. Typically, this would be your continuous integration and continuous delivery system. When you trigger a run with the API workflow, you will need to bundle up your Terraform configuration in a specific format and submit it to Terraform Cloud for evaluation. The API workflow is complex to set up, but it is also incredibly flexible. We won't be setting up a demo in this book, but HashiCorp has some reference scripts in their documentation if you want to experiment. Instead, we are going to focus on adopting the most common workflow, the VCS workflow. Before we dig into what the VCS workflow is, you might be wondering why would I even move away from my current CLI workflow, it's simple and familiar. What are the benefits of using a version control system? If you are the only person working on a Terraform configuration, there might be no reason to move to VCS, but as soon as your configuration expands to include other team members or you want to automate your process, you might find some distinct benefits to VCS. For instance, the code for your Terraform configuration is now stored in source control, which means there is a centralized location everyone can go to to access the latest version of that configuration. There is one source of truth for your deployments much easier than passing around files. Your team can collaborate on the code through mechanisms on your VCS provider. Code review, suggested improvements, and best practices can all be applied through things like commit hooks, issues, and pull requests. You now also have access to a formal approval process for deploying new versions of your configuration and promoting the new version across multiple environments, and because version control maintains previous versions and allows you to track changes, it's a lot easier to roll back a change if it breaks something. So while the VCS workflow might seem like overkill in your solo CLI endeavor, the additional effort is well worth it if you're in a collaborative environment or you want to automate your deployment process. Let's check out what's in the VCS workflow that enables all these benefits. VCS stands for version control system, so it shouldn't be terribly surprising that the VCS workflow requires that you put your Terraform code in a repository hosted on a version control system. Terraform can natively connect to several VCS providers, including GitHub, GitLab, Bitbucket, and Azure DevOps. If you are using the private version of any of those products, you will probably need to supply Terraform Cloud with SSH keys for access. If your VCS is not in this list, you can always go the API workflow route. When you create a VCS connection and associate it with a workspace, operations on the workspace will be triggered by events in the repository. You can control which branch and directory is monitored for changes. For instance, you may have your Terraform code sitting in a subdirectory of your repository, and you only want to trigger an evaluation when the content of that subdirectory changes. The VCS workflow will ignore any back end or cloud blocks defined in the code. That's because the VCS workflow is triggered by a webhook on the selected repository. So Terraform already knows it's using Terraform Cloud and which workspace. The cloud block we added in the previous chapter does not need to be removed, and in fact, we can use it to test a plan at the CLI. Speaking of which, you might be wondering what happens to the CLI interaction after we've migrated to the VCS workflow. The most significant change is that you are no longer able to trigger an apply operation on the workspace. All apply operations must happen through the VCS workflow or at the Terraform Cloud UI. You can, however, still execute speculative plans to test out changes to your code before you commit them to source control. What happens when you do commit to source control? How does that trigger an apply operation? Well, let's assume we've got a simple repository set up in GitHub with a single default branch. That branch is associated with the workspace. Locally, you will make changes to your code and then commit and push those changes to the repository in GitHub. Terraform Cloud will see a commit on the default branch and assume that you want to update your infrastructure. It will trigger a terraform apply operation on a remote runner and stream the output back to the UI and the VCS service. The apply operation first runs a plan and then waits for someone to approve the plan. From the UI, you can review the plan and approve the changes. If you're feeling extra confident, you can configure the workspace apply method to auto apply on a successful plan, but I wouldn't recommend that for production workspaces, at least not right away. When you are working on your code locally, you might want to see what the plan changes are without having to commit to source control and log in to the UI to view the results. Fortunately, you can still run a speculative plan through the CLI. You will need to have the correct workspace selected and then simply run a terraform plan from the command line. Terraform Cloud will kick off a speculative plan on a remote runner and stream the output back to your terminal. Since this is a speculative plan, it cannot be saved and applied to the workspace, but at least you'll see what changes would have been made based on your code updates. If you like what you see, you can commit the code and push it to GitHub, which will start the apply process we just reviewed.

Chapter 6 How to Create Repository Branches

Workspaces allow you to use the same configuration for multiple deployments by storing separate variables and state data. We are now going to investigate how the VCS workflow supports multiple workspaces from a single repository. We've already seen the workflow for a single workspace, but how do you work with multiple workspaces and a single repository? The mechanism used by Terraform Cloud is branches or directories. In this example, we will focus on using branches. We're going to use some software development techniques here, so bear with me. On the top of the screen we have our VCS provider, and on the bottom we have Terraform Cloud. When a developer wants to introduce a change or a feature to their software, typically, they will create a new branch from the default branch and develop the code in the feature branch. Once they are happy with the state of their change, they will ask to merge that update into a testing or development branch through a mechanism called a pull request, sometimes called a merge request depending on your VCS. Terraform Cloud can monitor for pull requests on a target branch, like development, and run a speculative plan when a PR is created or updated with new commits. The person who is responsible for reviewing and approving the pull request can look at the speculative plan to see if the change looks good. Once a PR is approved and the code is merged, the result is a new commit on the target branch triggering an apply operation in

Terraform Cloud. After the development environment update is applied, there might be further testing or updates before the code gets rolled to the next environment. The next step from the VCS side is for a pull request to be submitted against the staging branch, triggering a speculative execution in the staging workspace, followed by a merge of the code to staging and applying the changes to the staging workspace. Finally, after whatever testing is necessary in staging, the same process is repeated for the production branch, which is often the default branch for the repository, a pull request and a speculative plan followed by a merge and apply on the production workspace. This might seem a little esoteric, so why don't we put it into practice for the Donovan? Our goal in this demonstration is to set up the workflow we just saw. To get started, we will create branches for development and staging in the repository and update the development workflow to track the development branch. Then we will create two new workspaces for staging and production and configure each workspace to use the VCS workflow and follow their designated branch in the same repository. We'll also create a variable set to be shared across the two workspaces and add our AWS credentials to the variable set, avoiding the need to create them for each workspace. Finally, we'll make an update in development and walk it through the full update workflow from development to staging to production. Our first step is to create the branches that we will use in our repository. So I'll go ahead and click the here, and we're going to create a new branch called development. And then we'll create another branch called staging, and we'll use the main branch as our production branch. Now let's go over to the development workspace, and we'll go up to Settings and click on Version Control. And in the Version Control settings, we'll update the tracked branch to be development and then click Update VCS settings. In this chapter, we may have moved out of your Terraform comfort zone and into the world of version control and software development, but don't worry, it's not that scary, and Terraform Cloud makes it fairly easy. The VCS workflow is controlled by events in your version control system and can be associated with a specific branch and directory. Although we lose the ability to run an apply action from the CLI once we adopt a VCS workflow, we still have the ability to run speculative plans and test changes to the code. Since the VCS workflow can be associated with the branch, we can leverage a single repository with multiple branches to support multiple workspaces. We also saw how we can use variable sets to simplify defining common variables across workspaces. If you find that the VCS workflow doesn't provide enough flexibility for your organization, you can always adopt an API workflow. Although it is more complicated to implement, it is also the most flexible option of all the workflows. Now that we have our code stored in version control, we can take advantage of another feature in Terraform Cloud, Sentinel, and applying policy as code. We will be able to check for security issues and best practices in our code before it is applied to the target environment.

Chapter 7 HashiCorp Sentinel Fundamentals

Our Terraform configuration deployment has been automated through the wonder of the VCS workflow, and that opens the opportunity to add other elements to the automation process. Enter HashiCorp Sentinel, a policy as code solution we can leverage in Terraform Cloud to validate best practices and requirements for security, compliance, and engineering. In this chapter, we'll apply policy as code using Sentinel. Before we start using Sentinel, we should probably get to know a little bit about what Sentinel is and how it is constructed. Once we have that under our belts, we can get down to brass tacks, seeing how Sentinel policies are applied to Terraform configurations and how it informs Terraform Cloud operations. To apply the knowledge we have gained about Sentinel and policy as code, we will review some requirements from ACME and apply them to our Donovan workspaces. But first, let's begin with an overview of Sentinel. HashiCorp's Sentinel is a framework for implementing governance policies as code in the same way that Terraform implements infrastructure as code. Sentinel is embedded in HashiCorp's enterprise products like Consul, Vault, and, of course, Terraform Cloud and Terraform Enterprise. Since the policies are defined with code, they can be managed in a VCS repository and developed using common software development workflows. The policies use Sentinel's native policy language, which is a balance between configuration languages like YAML and JSON and languages like

Python. The intention is to make it flexible enough to accomplish its goal while still approachable to The policies used by Sentinel are as fine grained as you want and based on one or more conditions. Information used to make a policy decision can come from Terraform itself or from an external source. Any violations found can result in an advisory, a soft rejection, or a hard full stop rejection. Let's dig into the core components of Sentinel and see how they are leveraged in the Terraform Cloud context. In the context of Terraform Cloud, Sentinel policies are grouped into policy sets. The policies and the policy sets are stored on a VCS repository and linked to Terraform Cloud at the organization level. You select the repository, branch, and path to the policy set document. You can manage your Sentinel policies with the same workflow we saw in the previous chapter. Policy sets in Terraform Cloud are applied at the workspace level. Each policy set can be applied globally to all workspaces or to a select set of workspaces. The evaluation of a policy set occurs during a Terraform run after the plan and cost estimation phases, but before the apply phase. If the plan phase fails, the policy set will not be evaluated. Placing the policy evaluation phase after plan and cost gives it access to the plan data, cost information, and any information stored in the workspace, like tags or state data. Sentinel can also pull information from external sources as well, but let's focus on Terraform Cloud. A policy can evaluate as passing or failing. If all of the policies pass, the apply action will be available for execution. When it fails, the policy definition in the policy set determines what action Sentinel should take. An advisory action simply adds a warning, but allows the process to continue. A soft mandatory requires explicit approval before the apply stage can start. A hard mandatory prevents the apply stage until the underlying policy violation is remediated. You probably don't want to run through the entire Terraform plan operation just to see if your policies will pass or fail, especially when you are testing new policies or tweaking existing ones. That's why the plan phase of a workspace run has an option to download Sentinel mock data. The mock data includes the configuration, plan data, and state data, meaning it may have sensitive data contained within and should be treated with care. Sentinel includes a CLI that allows you to test and run policies. The mock data can be imported using the Sentinel CLI and used to test those policies. Mock data is produced as long as the plan phase passes, so you could use it as a troubleshooting tool if the Sentinel policy phase fails. Let's take a look at an example policy and policy set to get a feel for how they are structured. Using Sentinel and the Sentinel language could be an entire book on its own. We are simply going to cover the basic syntax to get you started. If you're interested in going deeper into writing Sentinel policies, HashiCorp has some great examples posted on GitHub and good documentation for both Sentinel itself and using Sentinel with Terraform Cloud. We'll start with a basic policy, one that checks to make sure only the proper instance sizes are being used by a configuration. Policies will have the extension, .sentinel. First, we need information about what is going to be deployed, and we can get that from the plan data. The import keyword allows you to import data structures and functions into your policy.

In this case, we are importing the plan information provided by the plan phase of the Terraform run. Next, we can get all the EC2 instances from that plan data using the filter function. We are looking at the resource_changes information stored in the plan data and looking for any changes that are of type aws_instance and mode managed. The list is being stored in a variable called ec2_instances. we can test to make sure the instance sizes we want are being created or updated. Each policy must have a main role that resolves to true or false. This is the final determination of whether a policy passes or fails. For our main role, we are looking at all entries in our ec2_instances variable and checks to see if the new value, aka, instance.change.after.instance_type, is in the list of t2.micro or t2.small. If any of the instance types are not in that list, then the rule evaluates to false, and the policy fails.

The enforcement level for a policy is defined by the policy set. Let's take a look at one now. Policy sets are a collection of policies and chapters defined in an HCL file. Policies can be sourced from the same directory as the policy set file or from a relative or remote location. In this first policy, the name is and the enforcement level is set to meaning the policy has to pass for the Terraform run to move to the apply phase. Since we don't list a source for the policy file, Sentinel will look for a file called in the same directory as the policy set. The second policy is called and this time, we are listing a source. The policy file is located in a separate GitHub repository. The enforcement level is advisory, meaning that if the policy fails, you will be notified, but the apply phase will still be available. I mentioned modules briefly, so why don't we expand on that? Sentinel supports the use of modules, which are analogous to a library in other programming languages. Modules bring additional functionality to policies, instead of having to write a common function directly in each policy document. This helps you practice the principle of dry or don't repeat yourself software development and take advantage of other's efforts. In the sentinel.hcl policy set, we are bringing in the module, As the name implies, this chapter has a set of functions that make it easier to work with tfplan data. Over in the file, we can import the module, and give it the name plan. Now, instead of using the longer filter function from earlier to get all EC2 instances, we can invoke the find_resources function from plan and feed it the resource type of AWS instance. That's pretty convenient. Armed with this new knowledge, let's see how ACME plans to use Sentinel in Terraform Cloud. The security and compliance teams at ACME have some requirements they would love to see applied to various workspaces. Starting at the organization level, in every workspace, they would like to see the project and billable tags applied to all resources that support tags. They also do not want SSH open to the world since that seems like a bad thing. At the development environment level, they would like to make sure that only smaller instance types like t2.micro or t2.small are being used. They are also trying to follow HashiCorp's recommendation of not using provisioners by prohibiting the use of local and provisioners. This prohibition will start at the development level and eventually roll up to the rest of the workspaces. To fulfill ACME' requests, we are going to need to create two policy sets, one for all workspaces and a second for the development workspaces only. Within each policy set, we'll need to reference a couple of policies that satisfy their requirements. For the demonstration, we are going to start by reviewing the contents of the Sentinel policies we'll use for all workspaces and just the development workspaces. After the review, we will create and apply the global policy set for all workspaces and test the policy set by running a plan operation on one of the workspaces. It's going to fail the first time, so we'll remediate the issue and see that it passes on the second try. Next, we will create and apply the development policy to our Donovan development workspace and test the policy by running a plan operation. Once again, things will fail, but this time we can override it for the moment. Let's head over to the demo environment to get started. Here are the exercise files in Visual Studio Code, and we are working with the files in M6, so I'll go ahead and expand that now, and we have our policies grouped into the dev and global directories. Let's look at the global policies first. We have two policies , and

Taking a look at the we can see that we are first importing as planned, so we have access to all those extended functions, that's pretty convenient, and we're setting two mandatory tags that should be applied to all AWS instances. They are Project and Billable.

Next, we will get all of the EC2 instances using that handy function, find_resources of type aws_instance, and then we're going to use another handy function, filter_attribute_not_contains_list feeding it all of the EC2 instances we just got looking in the field tags_all for our mandatory tags and saying that those mandatory tags should exist, that's what the true portion of that is.

And then finally, for our main rule, we're looking at the length of the messages property for violating EC2 instances. If it's 0, then there were no violations found. If it's anything else, then we need to fail this because it didn't find a tag on one of those AWS instances.

Taking a look at this one's a little more complicated, so we don't necessarily need to go through the whole thing but essentially, we're going to, again, import some tfplan information, import the and then we're going to look for the forbidden CIDR, 0.0.0.0/0, which is open to the world, and the forbidden_port 22, which is what SSH uses.

And then it's going to scan through all the security group ingress rules and see if any of those security group rules have a CIDR block of 0.0.0.0/0 and the forbidden_port. If any of them do, and we scroll all the way down to the bottom, like I said, this is a little more complicated than the other one, and actually grab this directly from the example files published by HashiCorp. If there are any violations, then all the way at the bottom, we have our main rule and it basically looks if validated is set to true.

If validated is true, then everything passes. If it's not, then there was a violation in one of the security group rules and it needs to fail. Looking in our sentinel.hcl policy set, we're importing the chapter because we're going to use it in our policies.

We're adding the policy which is set to the enforcement level of advisory, and the policy setting that to enforcement level of

Now let's take a look at the development policies. We've reviewed the global policies. Now it's time to take a look at the dev policies. I'll go ahead and expand the dev folder, and we've got two policies, and

Let's take a look at those prohibited provisioners. Once again, we're going to import some functions. This time it's the that we're importing, and we're creating a prohibited_list for provisioners of and

Then we're going to use this function, find_all_provisioners, that looks through the config to find any provisioners. And then it's going to look and see if any of those provisioners are in the prohibited_list, and if it is, then in the main role we are going to fail this policy if either of those prohibited provisioners are being used in the configuration.

Looking over in the this time we are using the we're setting the allowed_types to t2.micro and t2.small. Then we're going to get all of the EC2 instances and check to see if any of the instance types in those EC2 instances are not in the allowed_types. If that's true, then the main rule will evaluate as false and the policy will not pass. Looking at our sentinel.hcl file, we're importing two different chapters here, the and the and then we have two policies. The is set to and the is set to Now that we've looked at our policies and policy sets, it's time to add those policy sets to our Terraform Cloud organization.We are first going to add our global policy set. To do that, we need to go into the Settings for the organization. And in there, we will click on Policy sets. First, we need to connect a new policy set, so go ahead and click on that, and this will take us to a page where we can connect to a version control provider.

I do want to change the name to If I go down to Policy Set Source, because the policy set is not in the root of the repository, we're going to have to give it a path. We are using the default branch in our repository, so nothing needs to change there, and we want this applied to all workspaces, so I can leave the Scope of Policies selected as Policies enforced on all workspaces. Go ahead and click the Connect policy set, and now that policy set has been created and applied globally to all workspaces.

Let's go ahead and kick off a plan for our development workspace and see what happens. I'll go into Workspaces and go into the development workspace, and we'll go ahead and kick off a new plan. We'll do Start new plan here, and we'll say new global policy set. I'll go ahead and start the plan, and we can monitor the run from here.

This will go through the plan phase first, then the cost estimation, and then we have a new phase . It's the policy check. That comes after plan and cost estimation. So let's wait until that kicks off. our policy check has passed.

And if we look a little bit closer, we can see that Policies: 1 passed and 1 advisory failed. Before we look at that, I do want to point out in the plan portion of the phase, there's an option to Download Sentinel Mocks. So if you need that Sentinel mock data to test your policies, you can get it from there. Let's go ahead and expand the policy check, and we can see that the advisory failed is our If we look at the View JSON Data and scroll down a little bit, there is a print message. We can scroll over and see that it is missing the required item, Billable, from the list Project and Billable.

The Billable tag is not in there. So why don't we update the code in our development branch to include that Billable tag? Our tags policy failed with an advisory warning, which means we need to add this additional billable tag to our configuration. First, make sure that you've selected the development branch to check out from the list of branches in your Donovan repository. Now we're going to go into variables and create a new variable called billable. We'll call the variable billable and set the type equal to string.

We'll set the description equal to (Required) Billable code for project. We need to add this as a tag in our main.tf file. I'll go ahead and open that, and there are our default tags for the AWS provider. So I can simply add a new tag called Billable and set it to var.billable. I'll go ahead and save that and save the variables. And then lastly, let's go into terraform.auto.tfvars, and we're going to add a value for billable.

We have a value for billable. I'll go ahead and save that, and we'll also add that into our terraform.tfvars file just so that they are consistent. Now that I've made all of these changes, the next thing to do would be to commit these changes and push them. So I'll go into the source control portion, I'll go ahead and say update tags as the commit message, commit the change to source control, and push it up to the development branch. Going back to Terraform Cloud, because there were no changes to the infrastructure, there's no apply to run here.

Let's go back to the current runs, and a new run has started for update tags because we made a commit to the development branch. Wow, that's pretty convenient, isn't it? Let's go into update tags and look at the process. Plan is currently running. It has some changes to make, which is updating the tags associated with all these different resources. The plan has finished. Let's scroll down, and we can see that the policy check has passed with 2 passing and 0 failed. That sounds great, so we'll go ahead and confirm and apply this configuration, and now we'll go ahead and apply those updated tags to our resources. In summary, Sentinel could really be its own book. We've barely scratched the surface of what it's capable of. Let's review a few key points. First, Sentinel is a separate HashiCorp product that allows you to define policies as code. The Sentinel policies and policy sets for Terraform Cloud are stored and managed in a version control system. Policy sets can be applied globally to all workspaces or selectively. Policy sets are evaluated during the planning process, after plan and cost estimation, but before apply. Depending on the enforcement level, you may or may not be able to move forward to the apply phase. So far, the Donovan adventure has been a resounding success. Now it's time to expand your efforts to include other teams. Next, we will set up some teams in Terraform Cloud and see how we can use Teams to define permissions on workspaces.

Chapter 8 How to Operate Terraform Cloud for Teams

Terraform Cloud is all about collaborating with teams, but so far, we've only been collaborating with ourselves. The time has come to bring in other users to help out at ACME. The central organizing unit for users on Terraform Cloud is teams, and that is what we'll focus on in this chapter. Permissions in Terraform clouds are managed through teams. Users are granted those permissions by being a member of one or more teams. We'll start the chapter with a quick refresher on the relationship between users, organizations, and teams. You might be wondering what kind of permissions are available and how they are applied. There are two categories of permissions. First, we'll look at the permissions at the organization level, and we'll learn about the special owners team that has permissions not available to any other teams. Second, we will take a look at the permissions available at the workspace level and how to associate a team with a workspace and apply permissions. Then we will put it all into practice by looking at the updated requirements from ACME As a quick reminder of the relationship between users and organizations, a user in Terraform Cloud exists independently of any particular organization. A user can be a member of one or more organizations with different permissions in each organization. Inside of an organization, each user must be a member of at least one team. When you invite a user to be part of your organization, you also select a team they will be a member of. If you are creating a new organization, then you are automatically a member of the owners team for that organization. We'll talk more about the owners team a bit later in the chapter. A user can be a member of more than one team in the organization, and each team will have a set of permissions assigned to it. There are two types of permissions, permissions that include the ability to manage workspaces and policies, and permissions that control what a team can do in the context of a workspace. A team can have workspace permissions assigned on multiple workspaces, and the permissions assigned in each workspace do not have to be the same. A developers team could have write access on the development workspaces and on the production workspaces. A team can have both organization and workspace permissions associated with it. Some permissions imply permissions at the workspace level. For instance, the organization permission, manage workspaces, grants admin permissions on all the workspaces. Why don't we zoom in on what permissions are available at the organization level. A team can be granted permissions. As of this recording, there are five permissions you can grant. Manage policies grants the team the ability to create, edit, and delete Sentinel policy sets, and it also implicitly grants the read runs permission on all workspaces. Managed policy overrides grants the team permissions to override soft mandatory policy violations. It does not give the team permissions to alter assigned policies. So if a policy fails with hard mandatory applied, they cannot override that. Manage workspaces grants the team permissions to create and administer all workspaces in the organization. As I mentioned before, this implicitly gives them admin rights on all workspaces.

Manage private registry grants the team permissions to manage public providers and modules associated with the private registry and manage any private modules within the private registry. Manage VCS settings grants the team permissions to manage the VCS providers configured for an organization and any SSH keys for use with those VCS providers. There's one more category of permissions that I need to address, and that is permissions implicitly granted from outside Terraform Cloud. As we've seen with the VCS workflow for both workspaces and policy sets, anyone with proper permissions to the VCS repositories can initiate a change or run on Terraform Cloud, even if they don't have direct permissions. The same thing goes for any API integrations with external systems. Be mindful of how and when you delegate access from external systems. You might have noticed that things like changing billing or creating and managing teams are not in this list of permissions. Those and some other permissions are reserved for the owners group. In fact, let's quickly review what only the owners team can do. The owners team is a super special team created along with the organization. When a user creates an organization, they are assigned to the owners team. If the organization is on the Free tier billing plan, the only team available will be the owners team. Owners have permissions that are not available to other teams in the organization, including the following: the ability to invite new users to the organization and manage teams, including creation, membership, and deletion; the ability to manage all organization settings you see in the user interface, including setting the billing plan for an organization and managing the payment information; the owners team can also manage the lifecycle of Terraform Cloud managed agents running on your internal environment; and finally, owners can create an organization API token. This token can do just about anything relating to the organization, but it cannot perform workspace actions like running a plan, altering a config, or overriding a policy check. You can only have one active organization token at a time, and it should be treated with extreme caution. The general guidance is to use an organization token for initial setup and then replace it with a team token that has more restricted permissions. Speaking of restricted permissions, let's turn our attention from the organization to the workspace.

Chapter 9 Workspace Permissions

The most common use for teams in Terraform Cloud is to grant permissions on one or more workspaces. The permissions you defined for a team in one workspace can differ from what you assign in other workspaces. Terraform Cloud breaks workspace permissions into four different categories. Let's take a look at what's included in each of those categories. The four categories of workspace permissions are run, Sentinel, variables, and state. The run permissions control access to runs in the workspace and have the levels of read, queue, and apply. That means a team could have permissions to queue a run, but not actually apply the resulting plan. There is also a permission to lock or unlock the workspace, which will prevent runs from taking place while the workspace is locked. The Sentinel category is very simple. It controls whether or not the team can download Sentinel mock data from the workspace runs. There can be sensitive data in those mocks, so bear that in mind when granting access. The variables category determines whether the team can view workspace variables only or also edit those values. Values marked as secret will not be visible regardless of permissions since those can only be changed and not viewed. Lastly, the state category has three types of access, read outputs only, read the state data directly, and read and write the state data. Generally, Terraform Cloud manages state data alteration. So you really only need write access if you plan to execute your runs locally instead of using the remote runners. That's a lot of permissions to configure. Fortunately, Terraform Cloud makes it a bit easier for you by providing four permission sets to use. Each set builds on the permissions included in the previous one. Let's start with the read permissions set. Read provides read access to all information in the workspace, but you cannot edit anything. Plan adds the ability to create runs, but not the ability to apply them. Write can read and write data like variable values, approve runs, and control workspace locking. Lastly, admin has permissions to configure all settings in the workspace, including team access, execution mode, VCS configuration, and deletion of the workspace itself. Basically, if it's in the Settings menu of the UI, admins can alter it. Right now, there's no way to save a custom permission set as something like a role. So if you want to consistently apply a permission set outside of these fixed options, I'd recommend doing so programmatically with the API or Terraform Cloud provider. Now that we're familiar with the teams and permissions model in Terraform Cloud, let's see what ACME is looking to do with their organization. ACME has a few requests for you to fulfill in the Terraform Cloud organization. First, they want you to create an admin team that has all available permissions. Then they'd like you to invite a new user to the organization and assign them to this newly created admin team. Next, we'll move on to teams. ACME wants you to create the devs and managers teams that will be assigned workspace level permissions. The Donovan Devs team will receive the plan fixed permission set, allowing them to run speculative plans from the CLI, but still forcing them to go through the VCS workflow to apply changes. The managers will get the read fixed permission set to review the status of a workspace without being able to change any settings. These permissions should be applied consistently across all existing workspaces. Time to get some teams and permissions set up. We'll start by creating the admin team with the appropriate permissions. Then we'll invite new users Tricia McMillan to be a member of the admin team in our organization. Next up, we will create the Donovan devs and managers teams, and then we will finish out by assigning permissions to the workspaces. Let's head over to the demonstration environment and get started. From the Terraform Cloud UI, we're going to go into the settings for the organization and select Teams. This is where you create new teams.

And we're going to create a new team called org_admins. I'll go ahead and click on Create team, and now it will give us the opportunity to select what level of organization access this team will have. Remember that we want to select all available permissions, so we'll go ahead and tick all of these boxes here.

And after you have selected all the boxes, you have to click on the button below to update the team organization access. That's what actually saves the permissions. Scrolling down a bit, we have the ability to control the visibility of this team within the organization. In this case, we're going to set it to Visible for everyone and click the button to Update team visibility.

If we scroll down a little bit more, we can see there is the team API token. If you generate an API token for the team, that has the same effective permissions as anybody who is a member of the team.

Scrolling down a bit more, we can see we have the ability to add a new team member. This is for users who already exist in the organization. And we have the ability to delete this team if we're done with it. We have created our organization admins team, and now it's time to invite a new user. Go ahead and click on the Users here to start the process of inviting a new user, and then I will click on Invite a user. It's going to ask for an email address and what team to add this to. Remember, every user has to be a member of at least one team. I'll go ahead and put in the email address here, and then click the and select org_admins as the team to add her to, and click on Invite user. This will send an email to Tricia McMillan, inviting her to join the organization. I have her email open in the other tab, so let's switch to that.

We'll copy the URL for this invitation. And now we have to open up a separate private browse, because we're already logged in this browser. So I'll go ahead and bring up a private window, and we'll paste the invitation into here, and now we can create a new user on Terraform Cloud. If already had a user account on Terraform Cloud, it would simply add the organization to her existing user account. We'll go ahead and put in a username for her, and I'll paste in a password for her and agree to the terms of use and the privacy policy, and click on Create account.

Now we can see the invitation to join the organization, I'll accept the invitation. And because we have our organization set up to require MFA, we first have to go into the user settings for and set up authentication. I'll do that right now, and type in the authentication code.

Now that we have enabled authentication, we're allowed to select the organization. We'll accept the invitation again, and now we are connected into the organization and a member of the org_admins team. Now it's time to create the Donovan devs team and the managers team. We're going to create two teams for workspace level permissions. And even though we created a team called org_admins and added to it, she doesn't have the necessary permissions to create new teams. I'll click on Create team to create the team. We're not going to be adding any organization level access, so I'll scroll down and make this team visible to everyone, and click on Update team visibility to update that setting. And for now we're not going to add any team members, so we're all done with this team. Let's go back to Teams and create our managers team, and I'll click on Create team for that. We have successfully created the managers team, and I'll set the visibility for that also to be visible to all members of the organization. Our two teams have been created. The next step is to add them to Workspaces and assign them relevant permissions. Let's start by going to Workspaces and clicking on the development workspace. And within this workspace, we're going to go into Settings and click on Team Access to configure the team access for this workspace.

Next, we'll click on Add team and permissions, and we're going to select the devs team. And if we wanted to assign them custom permissions, we could toggle this radio button and it will give us full access to all the different permissions settings. But we're going to go with a fixed permissions set, so we'll scroll down and assign permissions for the plan fixed permissions set. Now we'll add the managers team. I'll select that team, and they will be granted read permissions to this workspace. We have now successfully created the managers and devs teams and assigned them the appropriate permissions on the development Donovan workspace. I leave it as an exercise to you to go into the other workspaces and assign the appropriate level of permissions to the teams. In summary we finally have some teams in our organization. If you'd like to experiment some more, I encourage you to invite some additional users and try out different permissions. In this chapter, we learned that all users in an organization must be a member of at least one team. The owners team is a very special team with permissions that aren't available to anyone else. The creator of an organization is a member of the owners team by default. We learned that there are permissions at the organization level and the workspace level. Some permissions create implicit permissions on the workspaces. Terraform Cloud offers fixed workspace permission sets to simplify administration, but you do have the ability to create your own custom permission sets. Next, we are going to onboard the Donovan’s Team to Terraform Cloud. They have already been using Terraform open source locally, so we need to get them set up as a team in Terraform Cloud and migrate their existing deployment.

Chapter 10 How to Migrate to Terraform Cloud

When we were setting up the Donovan, we were starting from scratch. They did not have any infrastructure deployed yet, but that is not going to be the case for most organizations that are already using Terraform open source. Instead, you will need to migrate their existing deployments over to Terraform Cloud. That is what we'll be looking at in this chapter. First, we are going to take a view at the migration process for Terraform Cloud. The good news is that it isn't terribly complicated. However, when any new technology is introduced, folks in your organization will have questions and concerns. We'll review some of the common concerns, and I'll do my best to address them. Then, we will meet the Donovan's Teams team at ACME and help them migrate to Terraform Cloud. First, let's talk about the migration process. If you're currently using Terraform you're already using a back end to store your state data. That state data is what we're migrating when we move to Terraform Cloud. You may have added a block to your configuration that points to a remote location to store state data. For instance, an Amazon S3 bucket. The state data is written out to that location. If you're using Terraform workspaces, each workspace will have a separate set of state data in the same location. Beyond the configuration and state data. You also have the actual infrastructure created by your Terraform code. When you migrate your state data from the current back end to Terraform Cloud, all you need to do is remove the block and replace it with a cloud block. If you're using the local back end without a block, then you can simply add the cloud block to your Terraform code. After changing your code, running terraform init will kick off the migration process. Terraform will see that the back end has changed and walk you through some prompts to complete the migration. You'll notice that while the code and the state data are changed, the deployed infrastructure remains exactly the same. This process is not disruptive to the functioning of your application infrastructure. Let's take a look at the actual steps involved when migrating to Terraform Cloud. The first step I already alluded to, and that is updating the configuration to use Terraform Cloud. The exact settings you configure in the cloud block will depend on how you are using Terraform today; particularly, whether you are currently using or plan to use multiple workspaces with the same configuration. The next step is to run terraform init to kick off the actual migration process. Terraform can see the current configuration in its local configuration data and will use that in combination with the new back end to perform the migration. You will be prompted to provide input depending on the workspaces set up in Terraform Cloud. After the migration, your work is not quite done. Assuming you're going to use the remote agents to execute runs, you'll probably need to set up some variable values in the workspace. You might also want to change the workflow type for the workspace or adjust other settings. Finally, you can run a speculative plan to verify that no changes will be made to your infrastructure. A successful plan with no changes shows that the migration was successful and your deployed infrastructure will not be impacted by the change. There are some common questions and concerns folks have when migrating to Terraform Cloud. So let's take a look at those. The single most common concern I hear is that by migrating to Terraform Cloud you are now locked in and cannot migrate off. Not to worry, you can always change to a different back end and migrate your state off Terraform Cloud should you decide it isn't right for you. Another concern is that you won't be able to migrate from your existing back end to Terraform Cloud or that it will require some tricky command line operations. If you're on a supported type today, you should have no problem migrating to Terraform Cloud. A lot of organizations are using multiple Terraform workspaces against a single configuration, and they might worry that workspaces won't migrate properly. In older versions of Terraform that use the remote back end instead of the cloud block, there could be some confusion around workspace naming and the use of prefixes. The tags argument in the cloud block helps alleviate those concerns. All your workspaces can be migrated without issue. When workspaces are provisioned on Terraform Cloud, you'll need to pick a workflow type, but how do you do that? That decision will largely depend on what your team is comfortable with and your future plans for automation. As we've already seen, the CLI workflow is the most familiar option, but the VCS workflow brings significant benefits for automation and code management. A closely related question is regarding automation. Many organizations already have in place a CI/CD pipeline that orchestrates their Terraform deployments, and they want to know if they can keep using that pipeline. Of course you can. You could either keep the pipeline the same by leveraging the CLI workflow with local execution or you could update the pipeline to leverage the additional functionality in the VCS or API workflows. Why don't we see what ACME is looking to do and how it will inform their choice of a workflow. The Donovan Team at ACME has been using Terraform open source and a single default workspace to manage their application infrastructure for quite some time now. They've heard about your success at using Terraform Cloud and would like to join the party. The code for the Donovan Team application and the state data are stored locally on a deployment machine at ACME HQ. The Donovan’s Team logs into the deployment machine and performs updates through the Terraform CLI. They would like to maintain that process for the moment. Your goal is to migrate the Donovan’s Team Terraform configuration to Terraform Cloud without disrupting their infrastructure or their current code deployment process. That means migrating the state data from the local machine at ACME HQ to Terraform Cloud by adding a cloud block and running CLI commands. For the workspace, we will leverage the CLI workflow for now since it most closely matches their current deployment process. The Donovan’s Team will need access to kickoff runs on the Donovan’s Team workspace, so we will need to create a Donovan’s Team dev's team and grant them write access to the workspace. Lastly, the Donovan’s Team would like to include the Donovan application URL in their app and keep it current if that URL changes. We can accomplish this by granting the Donovan’s Team development workspace access to the Donovan development workspace state data. Leveraging the remote state data source, we can set up a run trigger from the Donovan development workspace to the Donovan’s Team development workspace to pull that application URL. Since we don't already have a Donovan’s Team application deployed, we'll start the demonstration by deploying the code from our files. Then we will go through the migration process to move the state data to Terraform Cloud. Once the migration is complete, we will configure the workspace with proper variable values and team access permissions and validate the migration was successful by kicking off a planning run. Finally, we will add the remote_state data source to the Donovan's Team application, update the settings in both workspaces, and verify that the remote_state data access and run trigger work properly. In Visual Studio Code, I have the Exercise files open on the left. That is what we're going to be deploying to AWS. In the terminal below, I am already in the directory, so all I have to do is run terraform init. To initialize the configuration, this configuration is making use of a chapter that's stored locally and one that is from the Terraform public registry, which you can see in the initializing modules portion of the output. That will be relevant when we move to the next chapter in the book.

The next step to deploy the application is to run terraform apply, and I will add to skip the approval prompt. This is going to use the AWS credentials that I have stored locally from my AWS CLI profile. If you don't have your AWS CLI profile set up, you can store your credentials in environment variables in the same way that we did for Donovan and then deploy the application. Our deployment has completed successfully.

Our application has been deployed successfully, so the next step is to migrate our application to Terraform Cloud. We have successfully deployed our application, and the state for it is stored locally. The next step is to update the backend.tf file to .2 Terraform Cloud. So I'll go ahead and open backend.tf, and I'll start by removing the comment blocks here. All the comment blocks are out, and now we need to fill out the organization, in our case, it is And for the workspaces, instead of specifying a single workspace, let's use the tags argument instead. So I'll start with the argument tags, and I'll give it a value of two tags. The first one will be apps, so it will have the tag apps, and the second will be team. This means any workspaces created with this configuration will have these two tags applied to them in Terraform Cloud. I'll go ahead and save the back end, and then the next step is to run terraform init. You can see it says Initializing Terraform Cloud, so it found our cloud block. And now it is asking us if we want to migrate to Terraform Cloud. Do we want to migrate our existing state? We do, so I'll go ahead and enter yes. Next, it sees that we're using the default workspace locally.

But in Terraform Cloud, each workspace has to have an explicit name. So it wants us to create a workspace that will replace the existing default workspace. Since we do have a naming standard, let's go ahead and use it. We'll give the workspace the name because that matches our existing naming convention. I'll go ahead and hit Enter, and it will create that workspace over in Terraform Cloud and migrate our state data to that workspace. Now we can see it has successfully initialized Terraform Cloud.

The last thing that we need to do locally is to delete the terraform.tfstate file. If we don't do that, the next time we try to run a terraform plan or terraform apply, it's going to say, I see this local tfstate file. I'm not sure what to do, so I'm not going to do anything. So I'll go ahead and delete that file and the backup file. In this chapter, I hope you saw just how easy it is to migrate to Terraform Cloud, at least it is simple from a technical standpoint. As is so often the case, the real challenge is dealing with people and process rather than technology. To that end, it is critical to plan the migration process to match team expectations and properly train teams on using Terraform Cloud. Ideally you can pick a workflow to start out with that easily integrates into their existing processes with a plan to potentially standardize on a different workflow at some later date. Once you've migrated to Terraform Cloud, your workspace can be linked with others for data and runs. That creates an opportunity for more sophisticated automation and decoupled architectures. In the Donovan’s Team configuration, there are some chapters that might be useful to the rest of ACME. We will take a look at how to move those chapters to the ACME private registry and reference them in our code.

Chapter 11 How to Use Private Registry

One of the main benefits of using Infrastructure as Code is the ability to create reusable bits of code to deploy common infrastructure patterns. Terraform delivers this functionality in the form of modules. The public Terraform Registry allows people to create and upload modules and providers to share with others. Terraform Cloud includes a private version of the registry you can use in your organization. That's what we're going to dig into in this chapter. We will kick off our exploration of the private registry with an overview of what the private registry is, how it compares to the public registry, and the benefits that come from using a private registry in your organization. Then we will get into the details of how to actually use the private registry, including how to add providers and modules and how to update your existing configurations to use the private registry. Finally, we'll check in with ACME who has some plans to use the private registry to host modules and providers for internal use at the organization. But first, let's dig into the private registry overall. The private registry is a feature of Terraform Cloud available for any type of billing plan. It is essentially a private instance of the public Terraform registry dedicated to your organization. While the URL of the private registry is publicly available, access to the private registry on Terraform Cloud is restricted to members of the organization only. If you aren't a member of a team in the organization, you cannot access the registry. If you're running Terraform Enterprise, then there is the ability to share private modules from one organization to another inside the same Terraform Enterprise deployment. That functionality is not currently available to Terraform Cloud. Since the private registry is very similar to the public registry, it supports things like versioning of modules and providers and search functionality to discover modules and providers that have been added to the registry. You can browse documentation, look at examples, and even follow the link to source code if you have access. You can add three types of items to the private registry. The first two are public providers and chapters. Adding one of these items creates a pointer in the private registry that is linked to the public instance. You might wonder why you would add publicly available items to a private registry. I will address that in a moment. The third item type is a private module. Private modules are sourced from a VCS repository in the same way that you would add a chapter to the public registry and are only available from the private registry. One of the reasons to add public providers and modules to the private registry is to control which modules and providers are being used in your organization. You can enforce this restriction with Sentinel policies that require all modules to exist in the private registry, whether it's a pointer to a public module or a private module, preventing folks in the organization from using unapproved modules. Also contained in the private registry is a design configuration utility. It's meant to be a simple drag and drop experience for building Terraform configurations from the modules and providers stored in the private registry. We're not going to get into it in this book, but I thought I would mention it for completeness. Now that you know what the private registry can do, you might be wondering what the benefits are of using it. HashiCorp does provide guidance on hosting your own registry if you really want to. The Registry API spec is publicly available, and there are a few reference implementations, but that means you'd have to manage it yourself. The private registry in Terraform Cloud is a managed service that you can leverage for your organization. Adding both public and private items to the registry creates a centralized location with a curated selection of modules and providers that developers can visit to discover preferred items, read documentation, and view examples. Even if you don't plan to restrict what developers can use, having a central location for discovery is fairly convenient. If you're using internally developed modules today, there's a good chance they're stored in a Terraform configuration or hanging out on a file share somewhere. The private registry helps you move those modules into a VCS repository and add versioning for consumers of the modules. That's definitely a big help when you want to roll out a new version of a module that might contain breaking changes. the private registry is private in terms of access. If you have written modules that you don't want to be available in the wild, but you still would like to offer them to your internal teams, then the private registry is the perfect solution. The alternative is to host your own registry, but I'd recommend against doing that if you don't have to. We've talked a lot about adding items to the registry, so how exactly do you do that, and how do you reference those items in your code? Well, there are two distinct things we can dig into when it comes to using the private registry: adding new items and then using those items in your code. Since you can't use the items until they're added to the registry, let's start by looking at how you add items in the first place. You can add any providers or modules that exist in the public Terraform registry to the private registry. The private registry houses a pointer for the item referencing the public instance. The process is very simple. First, you select to add a provider or module from the Terraform Cloud UI and search for it in the public registry. Then you click the button to add it to the private registry, and that's it. That's the entire process. You can also add items via the API if you prefer not to use the Terraform Cloud UI. Adding private modules is a little more involved, and the process is remarkably similar to adding a module to the public registry. First, you need to create a VCS repository where you will store your module and name it using HashiCorp's official guidance, which is terraform, dash, the main provider used in the module, dash, the main purpose or resource in the module. You can store multiple modules in a single repository, but that's an advanced topic we won't get into. Once you've added your module code to the repository, you need to create a release tag with semantic versioning like v1.0.0 or 3.1.4. The private registry uses the release tags to manage available versions of the module. Once the module is added to the registry, it will automatically pick up on new versions of the module when it sees a new release tag added. The last thing to do is add the module to the private registry through a VCS connection. An important note here is that the private registry requires an VCS connection. So if you've used the GitHub app to connect to GitHub from Terraform Cloud, you'll need to create a new GitHub connection with OAuth. With our items added to the private registr, it's time to use them. Using the public providers and modules that have been added to your private registry doesn't require you to change anything in your code. You will reference them exactly the same way as you have before. For instance, the typical way to reference a public module is to configure the source and the version. The value for the source argument will be something like the repository name it came from, slash the primary resource type or purpose of the module, slash the primary provider used by the module.

In the case of a private module, the only real difference is the source argument. The updated form will be slash the primary resource type or purpose of the module, slash the primary provider type. Now don't worry about memorizing any of this.

The private registry will automatically generate a usage block you can copy and add to your code. We now know how to add items to the private registry and use them. In summary, a private registry is created for each organization on Terraform Cloud, and access is limited to members of that organization. That is what makes it private, after all. The private registry stores pointers to public providers and modules from the public Terraform registry and private modules associated with VCS repositories. The registry supports versioning for all item types and is searchable by organization members. If you would like to limit users to only items in the private registry, you may do so with Sentinel policies. That does it for this book. But, I'd like to take a moment to summarize the book as a whole and recommend some possible next steps and resources for you as you continue your Terraform journey. Terraform Cloud is a managed service offered by

HashiCorp that adds functionality not available in the version of Terraform. Terraform Cloud is made up of users, organizations, and workspaces in those organizations. The main benefits of using Terraform Cloud are the increased ability to collaborate with teams and implement automation for infrastructure deployment. One of the key integrations of Terraform Cloud is Sentinel, a policy as code engine that allows you to define policies and policy sets that will be applied to workspaces running your Terraform code. Terraform Cloud also includes a private registry where you can host public providers and modules and private modules you manage internally. That fairly sums up Terraform Cloud without going into too much detail. Now what could you do next in the world of Terraform? There are several other Terraform books available. Chances are you've already read the first two books or know the content, but there are also books that go deeper into the world of Terraform and look at implementing it on AWS and Azure. I'd also recommend checking out the learn site by HashiCorp. They have a whole series of practical labs you can follow to learn more about any Terraform topic. While you're there, you could broaden your horizons by checking out other popular HashiCorp products like Vault and Packer. I hope you found it educational and entertaining. Good luck and go build something great!

Conclusion

Congratulations on completing this book! I am sure you have plenty on your belt, but please don’t forget to leave an honest review. Furthermore, if you think this information was helpful to you, please share anyone who you think would be interested of IT as well.

About Richie Miller

Richie Miller has always loved teaching people Technology. He graduated with a degree in radio production with a minor in theatre in order to be a better communicator. While teaching at the Miami Radio and Television Broadcasting Academy, Richie was able to do voiceover work at a technical training company specializing in live online classes in Microsoft, Cisco, and CompTia technologies. Over the years, he became one of the top virtual instructors at several training companies, while also speaking at many tech and training conferences. Richie specializes in Project Management and ITIL these days, while also doing his best to be a good husband and father.