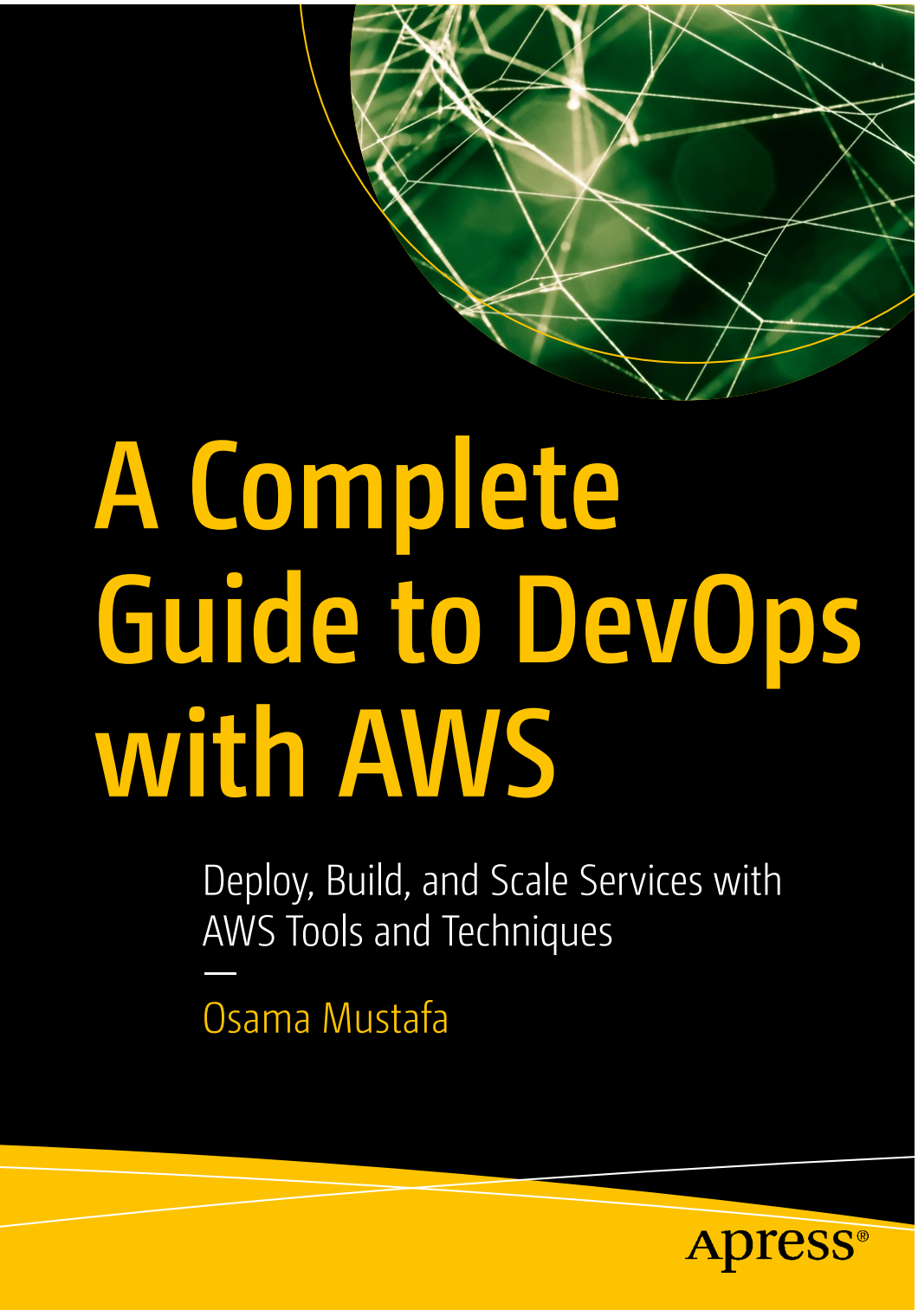

A Complete Guide to DevOps with AWS

Deploy, Build, and Scale Services with AWS Tools and Techniques

Osama Mustafa

Apress®

CHAPTER 2

Understanding DevOps Concepts

Before we start talking about Amazon AWS and DevOps, you need to have a good understanding of DevOps and why the demand of it is increasing every day.

DevOps Primer

DevOps refers to the departments of development and operations joining together. DevOps can be thought of as the extension of the Agile approach. On the technical side, it will speed up the transition from continuous (Agile) development to integration and deployment and improve the working environment for both the development and operations teams. As you can see from Figure 2-1, there is a gap between the two teams when they are not sharing anything; when implementing DevOps in a company, the gap will disappear, as you can see in Figure 2-2.

O. Mustafa, *A Complete Guide to DevOps with AWS*, https://doi.org/10.1007/978-1-4842-9303-4_2

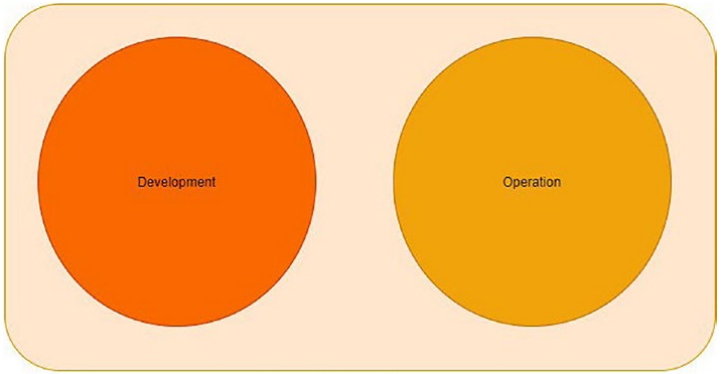

Figure 2-1. *Collaboration without DevOps*

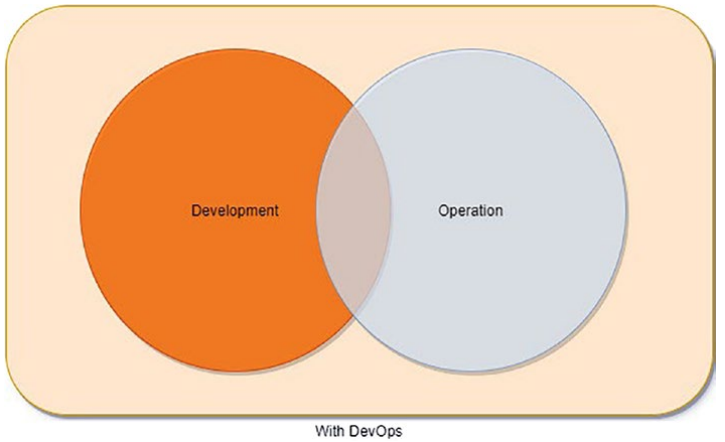

Figure 2-2. *Collaboration with DevOps*

DevOps itself is not a technology. Instead, DevOps tries to apply different methodologies within the company.

Usually, people mix up DevOps and Agile, but they are different from each other, and we will cover this in this chapter's "Agile vs. DevOps" section.

The DevOps Process

The DevOps methodology is meant to focus on and improve the software development life cycle. You can consider this process an infinite loop.

DevOps often employs specific DevOps-friendly technologies. These solutions aim to improve the software delivery process (or *pipeline*) by streamlining, shortening, and automating the different steps. These technologies also support DevOps principles, including automation, communication, and interaction, between the development and operations teams. Figure 2-3 shows the different phases of the DevOps life cycle.

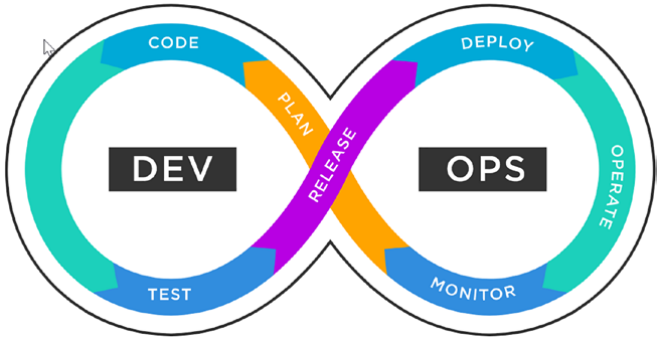

Figure 2-3. *DevOps process*

Let's break the process down and talk about the tools used in each phase:

1. *Plan*: This stage aids in identifying company values and needs. Jira and Git are examples of programs that can monitor known problems and manage projects.
2. *Code*: This stage involves generating software code and software design. GitHub, GitLab, Bitbucket, and Stash are some examples of the tools used.
3. *Build*: In this stage, you manage the software builds and versions and utilize automated tools to compile and package code for eventual production release. You use source code or package repositories to "package" the infrastructure required for product delivery. Docker, Ansible, Puppet, Chef, Gradle, Maven, and JFrog Artifactory are some examples of tools.

4. *Test*: Continuous testing (human or automated) is performed throughout this phase to guarantee the highest possible code quality. JUnit, Codeception, Selenium, Vagrant, and TestNG are examples of test tools.
5. *Deploy*: In this stage, DevOps coordinates, plans, and automates product releases into production. Puppet, Chef, Ansible, Jenkins, Kubernetes, OpenShift, OpenStack, Docker, and Jira are some examples of the tools used.
6. *Operate*: Throughout this phase, the software is managed during the manufacturing process with Ansible, Puppet, PowerShell, Chef, or Salt.
7. *Monitor*: Identifying and gathering information about problems from a particular software release in production is part of this step. New Relic, Datadog, and Grafana are examples of tools used.

To simplify, DevOps means that an IT development team will write code that works perfectly and meets the company standard to do some job. This code will then need a review and will be deployed on one of the environments such as production. This happens without any waiting or downtime. The good thing is that this code will be tested before being deployed to any of the environments.

To achieve this purpose, businesses employ a combination of culture and technology. The Ops team and developers work on the project, and developers work on minor upgrades that go live independently of each other to align the software to the expectations. Plus, to avoid wasting time, Ops configures a CI/CD pipeline that will be run automatically once the developer uploads new code to version control that needs to be reviewed to ensure it's meeting the company's standards.

This permits a business to create a secure piece of work. Small groups can quickly and independently develop, test, and install code and release fast, safely, securely, and reliably to customers. This lets businesses maximize developer productivity, permit organizational learning, create excessive worker satisfaction, and win within the marketplace.

These are the outputs that DevOps produces. However, this is not the world we live in for the majority of us. Our system is frequently dysfunctional, resulting in dreadful outcomes short of our full potential. Testing and infosec operations occur only after a project, which is too late to address any flaws detected. Every necessary action needs too

much human labor and too many handoffs, leaving us continually waiting. Not only does this result in incredibly long lead times for getting anything done, but the quality of our work, particularly production deployments, is also troublesome and chaotic, harming our customers.

Consequently, we fall well short of our objectives, and the whole business is unhappy with IT's performance, leading to budget cuts and disgruntled staff who feel helpless to influence the process and its outcomes. What is the solution? We need to transform how we work, and DevOps tells us how.

Let's look at the main elements of a DevOps ecosystem.

DevOps Goals

The following sections provide a succession of commonly recognized definitions, elaborated on with personal viewpoints.

Shared Goal

Most businesses can't implement production changes in minutes or hours; it takes weeks or months. Nor can they deploy hundreds or thousands of changes into production daily; instead, they struggle to deploy monthly or quarterly. Production deployments are not routine, instead involving outages, chronic firefighting, and heroics.

These companies are at a substantial competitive disadvantage in an age where competitive advantage means quick time to market, excellent service standards, and constant innovation. This is primarily due to their failure to address a long-standing issue in their technology department.

So, aligning efforts around enhancing system performance and stability, lowering deployment time, and improving overall product quality will result in happy customers and proud engineers; the objective must be reiterated, clarified, and simplified until it is thoroughly understood, defended, and finally adopted by everyone.

DevOps redirects attention away from self-interest and toward that objective. It emphasizes collective accomplishments above individual accomplishments, allowing other teams to see beyond the confines of their cubicle. Have faith in them.

Collaboration (No Silos Between Teams)

The software approaches that DevOps replaced were missing the benefits that coexisting Dev, Ops, and QA teams can deliver. They did not place a premium on team communication. Thus, it wasn't until DevOps was implemented that solid support for cross-departmental cooperation fundamentally altered how departments operate as a unit.

DevOps extends the Lean and Agile software development approaches since it emphasizes operations. DevOps fosters an atmosphere where diverse teams collaborate to accomplish similar company goals. This implies that your organization's teams will no longer be separated and will no longer strive toward department-specific objectives. However, influential organizations may continue to employ various technologies to do their tasks. DevOps fosters cooperation by dismantling barriers between development, operations, and quality assurance teams and encouraging them to work collaboratively toward a common goal: increasing value for your business. This will eventually help you give more value to your customers.

Collaborative work provides several advantages. Engineers are no longer concerned with the efficiency with which their team completes tasks—because they are all liable for the final result. Also, collaborative work encourages workers from disparate departments to explore ways to enhance a product's operational procedure. This kind of cooperation cross-trains your personnel by allowing them to broaden and update their technical experience in areas not precisely inside their area of competence. As a result, it's a win-win situation for your firm and your workers.

Sometimes there is an organizational silo: One person in the company who is the bottleneck. It is hard for others to fix any bugs because there isn't enough documentation to explain this person's work.

Some people get stuck in a silo because of bad luck. Many engineers had to deal with old systems when they started working; hats off to these engineers because I know how much effort is needed to understand other people's work.

DevOps can solve these issues with concepts such as cross-functional teams. The visibility of what these developers are doing is apparent to all the other groups, and everything is uploaded under one version control repo.

Speed

One of the main advantages of DevOps is that it speeds up how quickly your business works. For example, let's assume the development team creates new feature branches. They want to deploy manually; this will probably take months, depending on the availability and the risk these features will provide to the system. Assume they deploy this code and after two days find a bug in this new release. What will happen?

Applying the DevOps methodology will solve these issues; the faster you make great software, the sooner you can see how it can help your business. The time it takes to test a product depends on whether there is a place to try it. Updates and upgrades in software usually don't take very long to test if you have a test environment. However, trying a new product takes a lot of time because the operations team must set up a test environment. DevOps makes getting new features and changes into your software easier because it does automated testing and integration, which speeds up the process.

DevOps makes your developer watch your product all the time to ensure it is running smoothly without any issues. This means it takes less time to monitor, find, and fix bugs, which speeds up your time to market. Plus, it will help you find bottlenecks in production and processes that don't add value. Because you can work on fixing them, you'll be able to make money faster.

Innovation

DevOps is the key to unlocking new ideas in software development. We've seen how DevOps can quickly help you get your software products out. Such quick software delivery frees up some of your developers' time so they can try new features or improve the effectiveness of the ones they already have. Developers can test the feasibility of these ideas by performing a proof of concept and then going on with the project as planned, with little impact on the current project.

A set of rigid rules doesn't bind the developers; a project can permanently be changed if its goals are met. It might not be possible to use an idea one of your employees comes up with for an application with that product, but it might work well with another. When people work together, they develop new ideas and quickly test them. This helps DevOps be more innovative. Thus, DevOps allows for this environment and gives your software delivery the space to work well.

Satisfaction of Customers

Customer happiness is one of those aspects of the company that may make or ruin it. One of the primary advantages of DevOps is improved customer experience and, ultimately, customer happiness.

Over time, DevOps has shown its capacity to revolutionize and accelerate the software development life cycle (SDLC) process via more flexibility, increased automation, rapid deployment, decreased time to market, and increased productivity.

Figure 2-4 illuminates how businesses attain consumer happiness through six key factors.

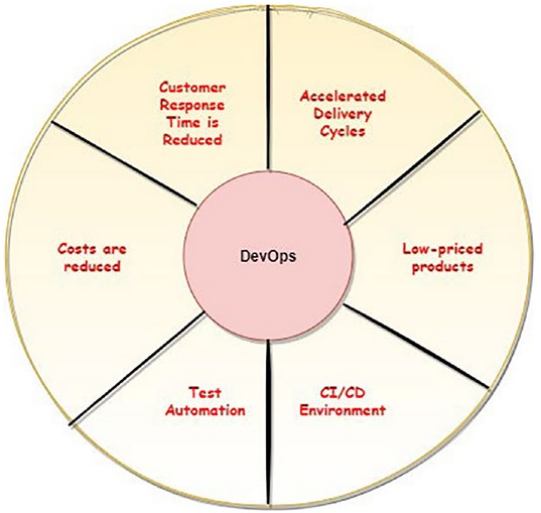

Figure 2-4. *DevOps benefits*

- *Customer response time is reduced*: Any delay in responding to client comments or complaints increases consumers' likelihood of moving to another provider or a close rival. This ultimately harms the customer base.
- *It accelerates delivery cycles*: The delayed process was caused by infrastructure problems and resulted in a long time for delivery. In the DevOps environment, releases and innovation occur quicker, with the ultimate business purpose of reaching out to customers early.

Automating repetitive jobs, eliminating routine duties, and streamlining procedures all contribute to the continuous operation of the software delivery chain. This results in a higher pace of innovation and the delivery of more products in shorter time frames. DevOps makes it simpler to achieve faster delivery cycles!

- *It allows for low-priced products*: Because of the shorter periods necessary for minor product launches, the cost aspect should not be a significant source of worry for these businesses. DevOps is precisely what makes this possible. As a result of the shorter manufacturing, operation, delivery, and feedback cycles, less money is spent on product development, reducing the total amount paid. As a result of the decreased development expenses, the customer's price is also lower. This ultimately results in the client being satisfied and interested in future versions.
- *It allows a CI/CD environment*: I will cover more about DevOps continuous integration/continuous development (CI/CD) cycles later in this chapter. CI/CD promises to stabilize the operational environment and respond quickly to production needs. CI/CD cycles reduce lead times and mean recovery time and permit more frequent releases, leading to customer satisfaction.
- *Tests can be automated*: When it comes to DevOps, automation helps to speed up the testing process from integration to staging. Automated testing promotes excellent product testing by preventing bugs from entering production early. Ultimately, this decreases the expenses associated with the launching of new products.
- *Costs are reduced*: Cost and value considerations influence DevOps' return on investment (ROI). One of the most significant advantages of the DevOps approach is the capacity to find and deliver to customers more quickly. This ultimately saves expenses, time, and resources that may be repurposed to identify new consumers and concentrate on providing more excellent value to current customers, among other things.

Before progressing and getting more technical, you should understand the difference between DevOps and Agile, which are connected.

Agile vs. DevOps

Agile is a way to manage projects and make software based on collaboration, customer feedback, and quick releases. It came from the software development industry in the early 2000s and helped development teams adapt to the market and customer needs.

Agile development is a way to break down a project into smaller parts and assemble them for testing at the end. It can be used in many ways, like Scrum, Kanban, and more.

When it comes to the stakeholders, the two concepts are different. Figure 2-5 explains who the Agile and DevOps stakeholders are.

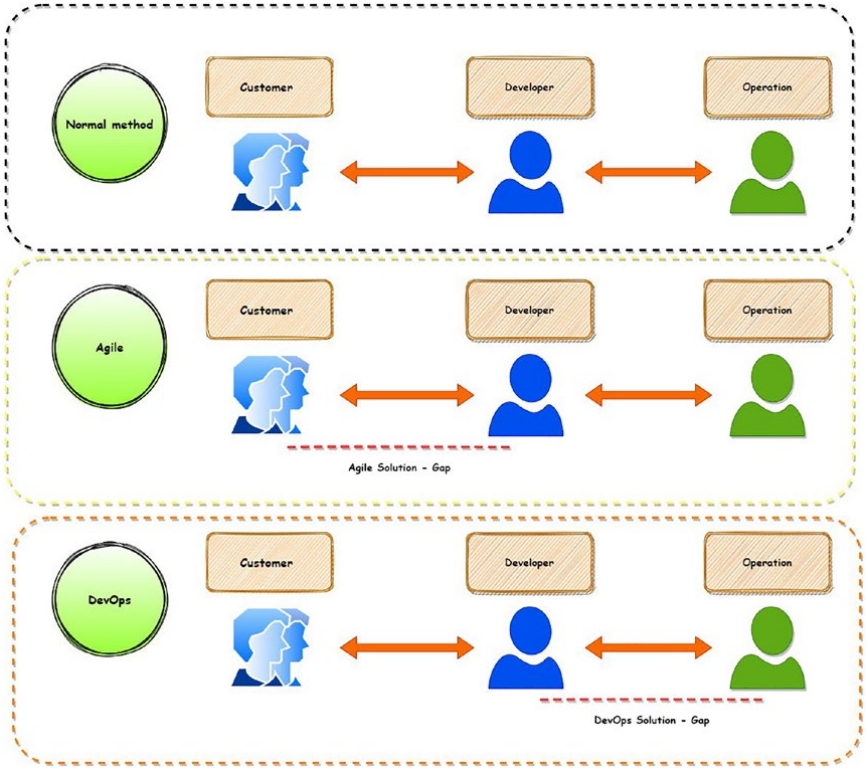

Figure 2-5. *Agile stakeholders vs. DevOps stakeholders*

Table 2-1 highlights Agile and DevOps's different factors to ensure you understand more about these two concepts.

Table 2-1. *Key Differences Between Agile and DevOps*

Factor	Agile	DevOps
Definition	Agile is a way of working that focuses on collaboration, customer feedback, and minor, quick releases.	DevOps is a way to bring development and operations teams together.
Purpose	Agile is a project management method that helps people work on big projects.	DevOps's main idea is to control all engineering processes from start to finish.
Task	In the Agile process, changes happen all the time.	DevOps focuses on testing and delivering things all the time.
Implementation	It can be done using Scrum or Kanboard.	It can be implemented using different software; we will discuss tools later in the chapter.
Feedback	Feedback is given to clients or customers.	Feedback is presented to the internal team.
Goal	There is a big gap between what the customer wants and what the development and testing teams can do to meet the goals.	There is a gap between development plus testing and Ops.
Advantage	There is a shorter development time and better defect detection.	It supports Agile's release cycle.
Tools	Jira and Kanboard are tools.	Puppet, Chef, TeamCity OpenStack, and AWS are popular DevOps tools.
Automation	There is no automation built in.	This is the primary goal of DevOps.
Speed vs. Risk	It supports change and application change.	It should make sure any risk will not affect the application or functionality.
Quality	It produces better applications that meet the needs of the people who use them.	There is a focus on automation, which reduces human errors.

There are some new DevOps terms in the previous table, so we will define them in the following sections.

- Continuous integration and continuous delivery
- Infrastructure as code
- Containerization
- Automation testing
- Version control

The person dealing with the software is the DevOps engineer; from coding and deployment through maintenance and upgrades, a DevOps engineer offers methods, tools, and approaches to balance the demands across the software development life cycle.

Continuous Integration and Continuous Delivery

Continuous integration is the process of developers merging changes as often as possible into the code base in the main branch. Developers create a build and then run automated tests against these changes. If these tests fail, the improvements aren't merged, and developers avoid any potential integration issues.

Continuous Integration's Advantages

These are the advantages:

- Early feedback and build/test automation reduce the time it takes from committing work to properly running it in production.
- Frequent deployment, automated tests, and builds are required for automated deployment.
- Time to restore service and automated pipelines allow for the speedier deployment of patches to production, lowering mean time to resolution.
- Early automated testing significantly decreases the number of problems that reach production.

Continuous Delivery

Continuous delivery is an extension of continuous integration. It automates the deployment of all code changes to a target environment (dev, QA, stage, prod, etc.) after they've been merged. Because the source of truth (your repository) is dependable given your CI process, the artifact may be created as part of CI or this process. This implies an automatic release process on top of the automated testing process. Developers may deploy their changes at any time by simply hitting a button or when CI is complete.

Continuous Delivery Advantages

These are the advantages:

- Makes the software release process more automated.
- Boosts developer efficiency.
- Enhances the code quality.
- Updates are sent more quickly.

Continuous Deployment

Continuous deployment is the last step of a sophisticated CI/CD workflow. It is an extension of continuous delivery, which automates the release of a production-ready build to a code repository. It also automates the release of an app to production. Continuous deployment depends significantly on well-designed test automation since there is no human gate at the pipeline point before production.

Roundup of Concepts

Figure 2-6 shows the process for each concept and what the difference is between the stages.

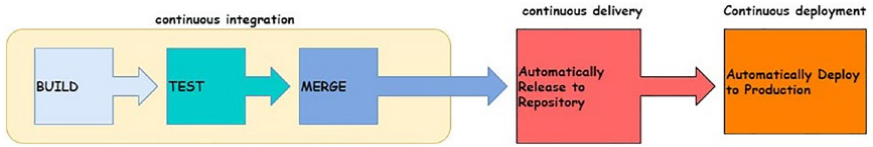

Figure 2-6. *CI/CD stages*

CI/CD Tools

The following are some standard CI/CD tools:

- Jenkins is an open-source automation server that allows developers to create, test, and deploy their applications with confidence.
- Spinnaker is a cloud-based CD platform for multicloud scenarios.
- CircleCI is a continuous integration and delivery platform that enables development teams to release code quickly and automate build, test, and deployment processes. With caching, Docker layer caching, resource classes, and other features, CircleCI can execute even the most complicated pipelines effectively.
- Concourse is an open-source CI pipeline tool that leverages YAML files for pipeline configuration and configuration-free startup; it has just released version 1.1.
- TeamCity is a general-purpose CI/CD solution that provides maximum flexibility for many workflows and development techniques. You can quickly check the status of your builds, learn what caused them, get the newest build artifacts, and do even more using the Projects Overview feature.
- Screwdriver is a build platform for large-scale continuous delivery. Because it is not bound to any computing platform, it supports an ever-growing variety of source code services, execution engines, and databases. Screwdriver is an open-source project with a well-documented API and a growing community.

Infrastructure as Code

To explain infrastructure as code, I will give you an example. Before DevOps, if the system administrator wanted to create and deploy two environments, such as development and production, with an average of 12 servers each, the developer needed to repeat the steps for both environments by making the servers, installing the operating system, setting up the network, preparing the tech stack inside the server, configuring everything, and testing it. Usually, these steps would take a month at a minimum, and don't forget the error timeline.

Infrastructure as code (IaC) will solve this issue. IaC is about managing and supplying infrastructure using code rather than doing it manually.

Configuration files containing your infrastructure requirements are produced using IaC, making changing and sharing settings easy. This also assures that an identical environment is created every time. IaC supports configuration management and helps prevent undocumented, ad hoc configuration modifications by codifying and documenting your configuration standards.

Version control is a crucial aspect of IaC, and your configuration files, like any other software source code file, should be under source control. Deploying your infrastructure as code also allows you to break it down into modular components that can be combined using automation.

Figure 2-7 illustrates the power of IaC; with a single code base, you can deploy into different environments.

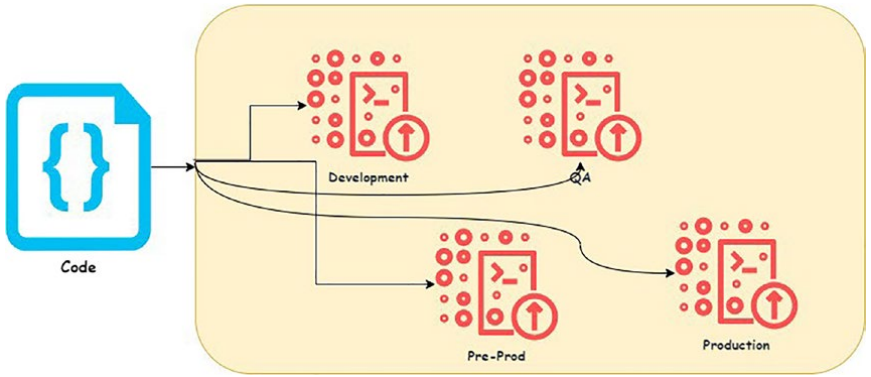

Figure 2-7. *IaC power*

System administrators don't have to manually supply and maintain servers, operating systems, storage, and other infrastructure components each time they create or deploy an application since infrastructure provisioning is automated using IaC. Coding your infrastructure establishes a template for provisioning. Although this may still be done manually, automation solutions can handle it.

How Infrastructure as Code Works

This is how IaC works:

1. The developer/system administrator writes, uploads, and uploads the code to version control.
2. After other developers review the code and ensure it meets the company standard, it will be approved.
3. Their integration with other tools, such as GitHub actions, AWS pipelines, etc., allows the version control to deploy this code automatically.
4. Depending on the tools, the code can be deployed to the cloud or on-premises to create the infrastructure.

Figure 2-8 shows a small example of how IaC works. The developer pushes the code into version control, which could be GitHub, GitLab, or Bitbucket; depending on the company, the code will be deployed automatically after the team reviews and approves the code.

The code author could be working on comments that the team gives to improve the code quality.

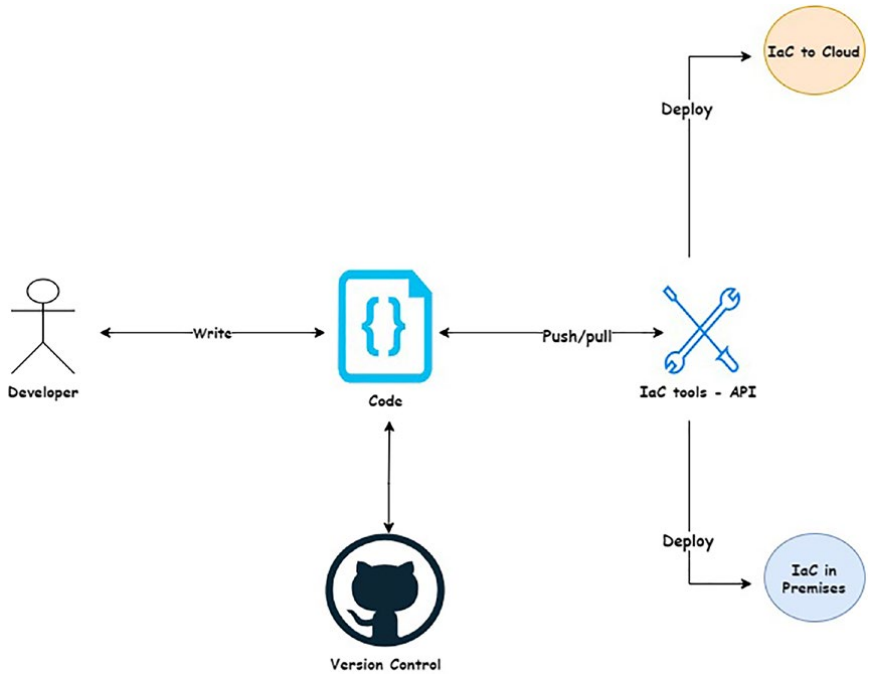

Figure 2-8. *How IaC works*

Regarding how IaC tools function, we may split them into two categories: those that follow an imperative approach and those that follow a declarative approach; you're correct if you believe the previous categories have anything to do with programming language paradigms.

The *imperative approach* specifies a set of commands or instructions the infrastructure must follow to achieve the desired outcome. By contrast, the *declarative method* "declares" the intended conclusion. The declarative approach illustrates the result instead of explicitly detailing the sequence of actions the infrastructure requires to obtain the outcome. I will explain more about these two approaches in Chapter 6.

IaC Types

There are different types of IaC, each of which can be used for a specific purpose:

- *Scripting*: The most straightforward way to implement IaC is to write scripts. Ad hoc scripts are ideal for performing basic, quick, or one-time actions. However, it's recommended to choose a more specialist option for more complicated configurations.
- *Configuration management tools*: These specialized tools, often known as *configuration as code*, manage software. Typically, they concentrate on setting up and configuring servers. Chef, Puppet, and Ansible are examples of these tools.
- *Provisioning tools*: Infrastructure creation is the focus of provisioning tools. Developers may describe accurate infrastructure components using these tools. Terraform, AWS CloudFormation, and OpenStack Heat are examples.
- *Containers and tools for templating*: I will cover containers in Chapter 9, but what you need to know about this IaC type is that you can build preloaded templates or images with all of the libraries and components required to launch an application. Containerized workloads are simple to distribute and have a fraction of the overhead of a full-size server. Docker, Vagrant, and Packer are some examples.

Standard IaC Tools

Each of the following tools has advantages and disadvantages. I will not go deep into each tool since there is a complete chapter on IaC with real-life examples and complete projects that allow you to understand more about this powerful DevOps category. Still, I will introduce these tools so you can get a brief look at them now.

Terraform

I consider Terraform one of my favorite tools; it is the most widely used open-source infrastructure automation technology. It assists with infrastructure-as-code configuration, provisioning, and management.

Terraform makes it simple to design and deploy IaC across numerous infrastructure providers using a single procedure. The needed infrastructure is defined as code using a declarative approach. Before upgrading or provisioning infrastructure, Terraform enables users to do a pre-execution check to see whether the settings fulfill the expected outcome. Customers may have their chosen architecture across numerous cloud providers through a single and uniform CLI procedure. You can swiftly create various environments with the same configuration and manage the whole life span of your desired infrastructure, eliminating human mistakes and enhancing automation in the provisioning and administration process.

Ansible

Ansible is often the most straightforward method for automating the provisioning, setup, and administration of applications and IT infrastructure. Users can use Ansible to run playbooks to generate and manage the infrastructure resources. It can connect to servers and conduct commands through SSH without using agents. Its code is written in YAML as Ansible *playbooks*, making the configurations simple to comprehend and deploy. You can even extend Ansible's capabilities by developing your modules and plugins.

Red Hat acquired Ansible to promote simplicity. It contributes to IT modernization and aids DevOps teams in deploying applications quicker, more reliably, and with more coordination. You can easily create several identical setups with security baselines without worrying about meeting compliance standards. Ansible provides a competitive edge in business by freeing up time for the company to implement innovation and strategy and align IT with business requirements.

Chef

Chef is one of the most well-known IaC tools in the business. Chef employs a procedural-style language, in which the user must write code and define how to attain the desired state step-by-step. It is up to the user to select the best deployment method. Chef enables you to build recipes and cookbooks using its Ruby-based DSL. These recipes and cookbooks detail the processes necessary to configure your apps and utilities on existing servers to your liking.

This infrastructure management solution is designed to help you implement and model a scalable and secure infrastructure automation process in any environment. Chef allows DevOps teams to supply and deploy on-demand infrastructure quickly. Chef is a configuration management technology many businesses utilize in their continuous integration and delivery operations.

Puppet

Puppet is another open-source configuration management solution widely used to manage several application servers simultaneously. It also employs a Ruby-based DSL, like Chef, to define the intended end state of your infrastructure. Puppet differs from Chef in that it utilizes a declarative approach, in which you must first determine how you want your settings to appear, and then Puppet will figure out how to get there.

Puppet is a collection of IaC tools for delivering infrastructures rapidly and securely. It has a large community of developers who have created modules to enhance the software's capabilities. Puppet connects with almost every major cloud infrastructure as a code platform, including AWS, Azure, Google Cloud, and VMware, allowing for multicloud automation.

SaltStack

SaltStack is a Python-based open-source configuration management application that provides a simple solution for quickly provisioning, deploying, and configuring infrastructure on any platform.

SaltStack focuses on automating an organization's infrastructure, security, and network. It's a simple IaC tool that comes in handy to mitigate and resolve typical infrastructure problems. It's a safe and cost-effective IaC system that automates and orchestrates processes while reducing human work. It can automatically identify issues with event triggers and restore the appropriate state if necessary. Salt also provides SSH support, which allows for agentless mode. It contains a scheduler that allows you to define how frequently your code should be performed on the managed servers.

Vagrant

Vagrant is another excellent IaC tool built by HashiCorp. Vagrant focuses on creating computing environments with a few virtual machines rather than enormous cloud infrastructure settings with hundreds or thousands of servers spread across several cloud providers.

Vagrant is a simple yet effective tool for creating development environments. It encourages the use of unified workflows by using declarative configuration files that include all of the necessary setup information. It ensures state consistency across environments and integrates with popular configuration management technologies such as Puppet, Chef, SaltStack, Ansible, etc.

Now, let's talk about the built-in tools of each cloud provider; you need to understand that each cloud provider has its own IaC tools that allow the system administrators, SRE, or DevOps to work on it; some cloud providers provide more than one in the same cloud, for example, Amazon AWS.

AWS CloudFormation

AWS CloudFormation is an integrated IaC solution inside the AWS cloud platform that allows you to rapidly and easily deploy and manage a group of connected AWS and third-party resources using infrastructure as code. It enables you to apply all needed DevOps and GitOps best practices. By connecting CloudFormation with other essential AWS resources, you can manage the scalability of your resources and even automate additional resource management. AWS CloudFormation also lets you develop resource providers using its open-source CLI to provision and manage third-party application resources alongside native AWS resources.

CloudFormation is written in YAML or JSON format. All you have to do is create your desired infrastructure from scratch using the appropriate template language and then utilize AWS CloudFormation to provide and manage the stack and resources indicated in the template. CloudFormation also uses rollback triggers to restore infrastructure stacks to a previously deployed state if errors are observed to ensure that deployment and upgrading of infrastructure are controlled.

Azure Resource Manager

The Azure Resource Manager service allows you to install and manage Azure resources. Instead of deploying, maintaining, and tracking resources separately, an Azure-specific IaC solution enables them to be deployed, maintained, and followed in a single collective operation. Role-based access control (RBAC) is built into the resource management system, allowing users to apply access control to all resources within a resource category.

Resource Manager lets you utilize declarative templates instead of scripts to manage your infrastructure. You may reinstall your infrastructure solution several times during the application development life cycle using Azure resource management while maintaining state consistency.

Google Cloud Deployment Manager

Google Cloud Deployment Manager is a solution that automates resource creation, setup, provisioning, and administration for Google Cloud. You can create a group of Google cloud services and manage them as a single entity. You may develop models using YAML or Python, preview changes before deploying, and examine your deployments through a console user interface.

Choosing the right tool depends on your expertise and why you need to use this tool.

Tool Roundup

Table 2-2 lists several tools and briefly explains how they work; you either need to configure an agent to make them work or install the tools on the server.

Table 2-2. *Tools*

Tools	How It Works	Focus
Terraform	Agentless	Admin focused, IaC
Ansible	Agentless	Admin-focused configuration management
Jenkins	Agentless	Dev focused and can work for admin also, CI/CD
Puppet	Agent-based	Dev focused
Chef	Agent-based	Dev/admin focused
Saltstack	Agent, agentless	Admin focused

Benefits of Infrastructure as Code

IaC creates a standardized, repeatable infrastructure definition. Why is it necessary to be repeatable? Because with infrastructure as code, you can be confident that you'll always have the same infrastructure. Developing and maintaining an infrastructure specified as code is more efficient than manually maintaining it.

Changes can be tracked, versions compared, security ensured, and rules enforced using IaC. The following are the advantages of IaC:

- *Speed*: Organizations that adopt IaC spend less time on manual procedures, allowing them to accomplish more in less time. Iteration is quicker since there is no need for an IT administrator to finish manual activities.
- *Consistency*: Every stage of the process is handled and finished automatically, removing the possibility of human mistakes and sick days and weekends. Changes are made widely and quickly, allowing development teams to focus on adding value.
- *Efficiency*: Following a more consistent approach to provisioning infrastructure can save time and concentrate on nonrepetitive, higher-order jobs.
- *Less headache*: Anyone can deploy the IaC; there is no need for a system administrator to deploy it, which means there will be less overhead on each department in the company.
- *Accountability*: This one is simple and fast. You can track each configuration's changes because IaC configuration files may be versioned like any other source code file. There will be no more guesswork as to who did what and when.
- *Low cost*: Undoubtedly, one of the most significant advantages of IaC is the reduction of infrastructure administration expenses. You may drastically lower your expenditures by combining cloud computing with IaC. You'll save money because you won't have to spend money on hardware, pay people to run it, or rent physical space to hold it. However, IaC decreases your expenses differently and more subtly.

Infrastructure as Code Examples

Let's look at a quick example of a basic AWS EC2 instance setup scenario; I will cover three different tools for the same example.

Terraform

Figure 2-9 shows how the file structure will look. I will not explain the files since Chapter 6 will cover this part.

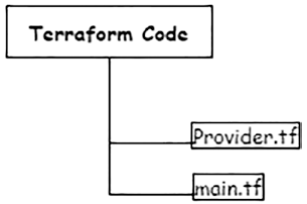

Figure 2-9. *File structure example*

This is the code in `provider.tf`:

```
terraform {
  required_providers {
    aws= {
      source  = "hashicorp/aws"
      version = "~> 3.27"
    }
  }
}

provider "aws" {
  region = "us-west-1"
}
```

This is the code in `main.tf`:

```
resource "aws_instance" "web_server" {
  ami                    = "ami-0123456"
  instance_type          = "t3.micro"
  subnet_id              = "subnet-14322"
  vpc_security_group_ids = "sg-143224"
  key_name               = "server-key"
  tags = {
```

```
    Name = "Web_Server"
  }
}
```

Ansible

For Ansible, we will use a fundamental YAML file; please note the indentation is significant in YAML.

```
- hosts: localhost
  gather_facts: False
  vars_files:
  - credentials.yml
  tasks:
  - name: Provision EC2 Instance
    ec2:
      aws_access_key: "{{aws_access_key}}"
      aws_secret_key: "{{aws_secret_key}}"
      key_name: server_key
      group: test
      instance_type: t3.micro
      image: "ami-0123456"
      wait: true
      count: 1
      region: us-west-1
      instance_tags:
        Name: Web_Server
    register: ec2
```

AWS CloudFormation

AWS CloudFormation is a service from Amazon AWS that makes it simple for developers and companies to construct a group of linked AWS and third-party resources and provide and manage them logically and predictably; you can use YAML or JSON.

```
AWSTemplateFormatVersion: "2010-09-09"
Resources:
  WebInstance:
```



```
    Type: AWS::EC2::Instance
    Properties:
      InstanceType: t3.micro
      ImageId: ami-0123456
      KeyName: server_key
      SecurityGroupIds:
      - sg-143224
      SubnetId: subnet-14322
      Tags:
      -
        Key: Name
        Value: Web_Server
```

As you can see from the previous examples, the tools lead to the same results; it's great to know different ones, but not mandatory. If you are thinking about deploying resources to cloud computing, the next step is to understand IaC because it will save time and money. Now it's time to jump into the next topic in this chapter, automation testing.

Automation Testing

Once upon a time, the only testing step in the development cycle was QA testing, which was one of the most extended steps in this cycle; the developer finished their work and uploaded it to version control, and the QA team needed to make sure the code met the company's standard depending on the testing scenario.

That took time and probably caused issues between the teams because no one likes to repeat work; DevOps solved this problem between the groups with a straightforward approach called *automation testing*.

What is automation testing? It is a software testing approach for comparing the actual result to the predicted outcome; it can be accomplished via test scripts or another automation testing tool. Automating repeated processes and other testing duties that are not easy to complete manually is what test automation is for.

Manual testing entails a single person evaluating the software's functioning in the same manner as a user would. Automated testing uses an automation tool, allowing more time on higher-value jobs such as exploratory tests, while time-consuming tests such as regression tests are automated. While updating test scripts will take time, you will improve your test coverage and scalability.

Manual testing allows the human mind to derive conclusions from a test that might otherwise be overlooked by test automation. Large projects, projects that repeatedly need testing in the same regions, and projects that have previously undergone a human testing procedure benefit from automated testing.

Advantages

Why do you need to use automation testing in the first place?

Some of the advantages of automated testing are as follows:

- Automation testing minimizes the time it takes to complete a test because automated testing is more efficient than manual testing.
- It lowers the project's cost and resource needs since a script produced once may be made to execute an unlimited number of times as long as the application remains unchanged.
- It allows you to deal with many inputs, which is impossible with manual testing.
- It helps create a continuous integration environment in which the new build is automatically tested after each code push. You can develop jobs using CI/CD technologies like Jenkins that tests when a build is deployed and emails the test results to stakeholders.
- You can save time by automating your tests. Automated tests are quick to perform and can be repeated. Put another way, you won't have to wait weeks to do the tests again—only a few hours will be enough, which will lead to different results.
    - The software development cycle is shorter.
    - Releases occur more often.
    - Changes and upgrades to the app are made more quickly.
    - Deliveries have a shorter time to market.
- Because automated testing doesn't need human participation while running, you may test your app late at night and get the findings the following morning. Software developers and QA may spend less time testing since automated tests can frequently run independently.

Basically, with automation, your engineers can focus on critical tasks. Fixing current app functionality isn't as thrilling as adding new features, as we all know.

- An immediate response provides immediate feedback. Developers get testing information instantaneously with rapid execution to respond swiftly if a problem happens. Forget about deciphering the code that was written three weeks ago.
- Test automation makes you more likely to have error-free releases with more precise tests. Automated testing is more accurate than manual testing because it requires less human interaction. The problem is that a human tester may make errors at any stage of the review process. The machine, on the other hand, will not cooperate. Because generated test cases are more exact than human testers, you may lower the likelihood of failure by removing human mistakes.
- Your app's excellent quality and performance will be ensured through automatic testing. It lets you run hundreds of automated test cases simultaneously, allowing you to test your app across different platforms and devices quickly. Choose cloud-based device farms to get the most out of test parallelism and concurrency. They can assist you with covering all of the needed OS and hardware setups.

Misconceptions

Since I mentioned the benefits of automation testing, I will discuss some misconceptions about it.

- *You will have more free time as a result of automation*: You need to understand that there is always something to automate; you can always enhance your infrastructure, and automation will allow the team to focus on other things and the quality of your code and infrastructure. Most manual testing time is spent on exploratory and functional testing, which involves manually searching for faults. Once that procedure is finished, the manual tester must perform the same actions repeatedly. That time is significantly reduced with automated testing. Instead of writing tests, automated testers code them and

improve them. However, after the test is completed, automated testing enables the reuse of tests, eliminating the need to repeat the whole procedure. Instead of spending time on the highly repetitive activities that a human tester would do, you may concentrate on broader, more fundamental problems with your building product.

- *The price of automated testing is excessive*: I don't understand why people think the automation options are expensive. You can use open-source software, and automated testing allows you to concentrate on more important topics such as client demands, functionality, and enhancements. Automated testing also lowers the cost and requirements for repeated code modifications, so the investment pays off over time. Furthermore, the software tests may be run once the source code is amended. Performing these tests is time-consuming and expensive, whereas automated tests may be done repeatedly at no extra expense. Parallel testing is another method to check how much money your automated testing makes. Similar testing allows you to perform numerous automated tests simultaneously rather than executing them one after another; this significantly reduces the time it takes to run your automated tests.
- *Manual testing isn't as good as automated testing*: This misconception makes me laugh whenever I hear it. We are not in competition here. Sometimes you need manual testing inside your company; it just depends on your business needs. Each method has its own set of benefits and drawbacks. A person sitting in front of a computer does manual testing by methodically walking through the program using SQL and log analysis, experimenting with different use and input combinations, comparing the results to the intended behavior, and documenting the findings. After the original program has been produced, automated testing is often employed. Unattended testing may execute lengthy tests that are generally avoided during manual testing. They may even be installed on several computers with various settings.

- *There is no need for humans*: Because automated testing is more precise and quicker than what people can achieve without incurring significant human error, this misunderstanding is understandable. On the other hand, complex applications are built to support a collaborative approach by incorporating tools that enable co-workers to walk through a piece of test code and provide feedback on the script. This automation does not replace the need for in-person contact or communications in software development. Instead, it increases that characteristic by giving an additional communication route. Consider it this way: email did not replace the telephone; it is only a different means of communication.

Stakeholders

Automation testing is different from any other automation; it requires expertise with QA and experience with scripting. The question in this section is, who should be part of the test automation process?

- *Developers*: They understand what kind of application the company has and the screens inside the application; therefore, integrating testing into the development process necessitates integrating development environments. It's part of the developer's job.
- *QA (manual or automated)*: For manual testers, those new to automation, recording, and replaying, utilizing the same recorded script with various input data may be helpful when finding and correcting issues in different contexts.
- *Automation experts*: These engineers will use a scripting language and interface with continuous integration platforms. The ability to scale testing might be critical for automation engineers.

Disadvantages

Automation is not always the solution for your business. The following are some scenarios where automated testing is not recommended and some of automation's drawbacks:

- *Inexperience with the automation tool*: One of the most common reasons for not using a tool and programming language to construct powerful scripts is a lack of experience with the tool and a particular programming language. These and other factors contribute to the failure of automated testing.
- *Applications that often change*: Choosing test automation for an application that undergoes frequent modifications necessitates ongoing maintenance of the test scripts, which may or may not result in the required return on investment. DevOps comes with the Agile methodology, which is not recommended.
- *Wrong test case*: The effectiveness of automated testing depends on the test cases selected for automation. Incorrectly specified tests result in a waste of resources and time spent automating.
- *Test scripts created inefficiently*: For this point, you need to understand the difference between DevOps and automation testing; both have different job responsibilities. Test scripts with insufficient or no validations might result in false-positive test results. These false-positive findings hide the underlying flaws, which might have been readily detected if manual verification or better scripting had been used.

Test Automation Process

Figure 2-10 shows the test automation cycle that can be added to the DevOps process, as I will show later in this chapter.

All operations are included in the automated testing process. We'll go over each step, from understanding requirements to automated scripting and CI/CD integration.

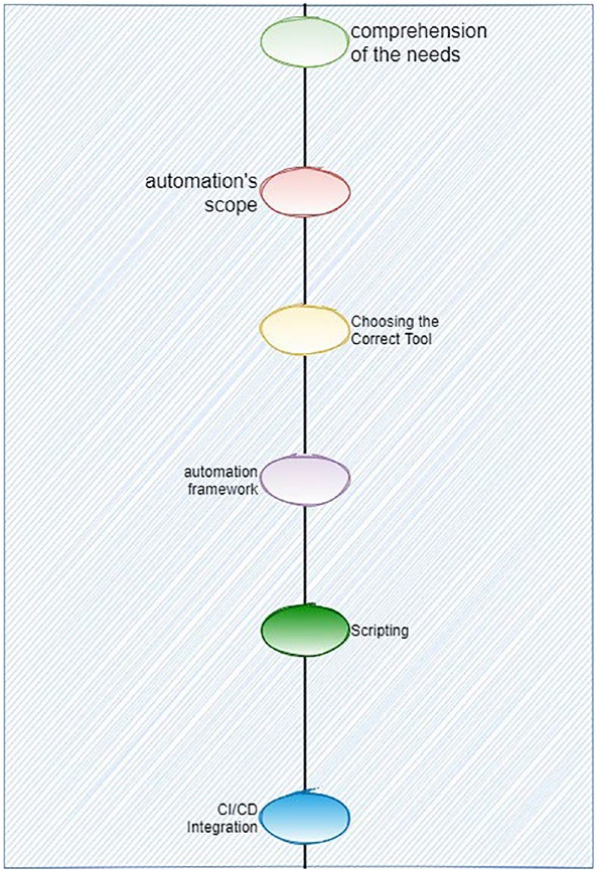

Figure 2-10. *Test automation process*

Let's break the process down:

- *Comprehension of the needs*: The first and most crucial step in test automation is fully comprehending the market. Understanding the demand will aid in defining the extent of automation and selecting the appropriate technology.

- *Automation scope*: Finding the correct test cases for automation is the first step in defining the area. All test cases under the test case classes specified in the "What to Automate?" portion of this chapter would fall under this category.
- *Choosing the correct tools*: The choice of tool is influenced by several variables, including the project's requirements, programming skills, project budget (whether to use a free or commercial tool), etc.
- *Automation framework*: We need to build an automation framework to produce comprehensive test automation suites. These frameworks aid in test scripts' reuse, maintenance, and robustness. Depending on the project requirements, we may pick from various automation frameworks; I will briefly describe these frameworks later in this chapter.
- *Scripting*: After the previous setup of the automation framework, we begin scripting the test cases that will be automated. The automation tester usually does scripting depending on the QA engineer, which will give the work to the automation tester.
- *CI/CD integration*: Although we can run the test cases on demand, CI/CD is widely used in almost every product or service-based company. This involves setting up the test suites on a CI/CD tool like Jenkins from a testing point of view. A significant advantage of combining the automation suite with the CI/CD pipeline is the ability to automatically trigger test cases for execution after deployment. The automated test suites in this configuration assess the build's stability with just one click right after deployment.

Automation Framework

After discussing the steps of automation testing, we need to learn more about the framework that this kind of DevOps category usually uses; we need to establish a set of guidelines for manual testing on any application. Examples include the structure of test cases, the prioritizing of test execution, and many strategies for improving the whole software testing process, to name a few. In the same way, we employ automation frameworks in automation testing to assist in reducing the expense of maintaining automated scripts. Improving the whole automated testing process is also a priority.

Ask yourself a few things before implementing the solution:

- What should be done to make this automation testing work?
- What is the report output?
- What should I do in case of failure?
- How can we test the data/output?
- How can we use these scenario functionalities over and over again?

To respond to the concerns mentioned, the type of framework is relevant.

- *Modular framework*: Modular frameworks are a kind of automation framework in which widely used capabilities such as database connection, login flow, reading data from an external file, and so on, are recognized and developed as methods. Instead of repeatedly writing the same line of code, we call the reusable functions whenever needed.
- *Keyword-driven framework*: You can write test cases; for example, create test cases in an Excel spreadsheet. The framework will already have methods designed for each phrase, such as automation code for the OpenBrowser and other keywords. Once the framework is set up, a nontechnical user may develop plain-text automation test scripts.
- *Data-driven framework*: The test data in a data-driven framework is stored in external files. The test case is performed numerous times with various datasets in each iteration, depending on the number of rows in the external files (fetched from the file). The term *data-driven framework* comes from the fact that data drives automation.
- *Hybrid framework*: As you can tell from the name, this is the combination of multiple frameworks. So, a variety of any two of the previous frameworks would be termed a *data-driven framework*. Mainly when we say a *hybrid* framework, it relates to a data-driven framework merged with a keyword-driven framework.

We have talked about almost everything related to automation testing; the one thing that is missing is the tools. Next I will cover some common tools I have used during my career.

Test Automation Tools

Let's start with one of the most common tools used for automation testing.

Selenium

Selenium is in demand and frequently utilized. It is one of the best QA automation tools available. It can automate across many operating systems, including Windows, Mac, and Linux, as well as browsers such as Firefox, Chrome, Internet Explorer, and headless browsers.

Selenium test scripts can be developed in Java, C#, Python, Ruby, PHP, Perl, and JavaScript, among other computer languages. Selenium's browser add-on Selenium IDE has record and playback capabilities. Selenium WebDriver is a sophisticated tool for constructing more complicated and advanced automation scripts.

Cucumber

Cucumber is a free behavior-driven development (BDD) tool. Cucumber is an open-source testing automation tool supporting languages such as Ruby, Java, Scala, Groovy, and others.

Testers, developers, and customers collaborate to write test scripts. Cucumber works only in a web environment. Gherkin is a simple English language used to write test code. Cucumber code may be run on various frameworks, including Selenium, Ruby, etc.

Watir

Pronounced "water," Watir is an open-source web automation testing tool. Watir is one of the top automation scripting tools for Windows and supports Internet Explorer. Watir + WebDriver is compatible with Firefox, Opera, and the HTML Unit headless browser.

Although Ruby is the scripting language, you may automate web applications written in any language; Watir allows you to connect to a database, read flat files, and use Excel, which is helpful for data-driven testing. You can create reusable test code that you can use in several test scripts. Watir is considered one of the best QA automation tools.

Mabl

If you don't like to code, Mabl is the right tool; it is an intelligent, low-code test automation solution for quality engineering. Agile teams can improve application quality using Mabl by including automated end-to-end testing in the development process.

Version Control

Another important DevOps category is probably used in every company nowadays, so what is it? Version control systems are software tools that aid in recording file changes by keeping track of code changes. Without version control, it's tough to automate things and create CI/CD pipelines.

Significance of a Version Control System

As we all know, a software product is produced collaboratively by a group of developers who may be based in various places. Each contributes a particular set of functionality/features. As a result, they modify the source code to contribute to the product (either by adding or removing it). A version control system is a kind of software that aids the development team in quickly communicating and managing (tracking) any changes made to the source code, such as who made a change and what it was. Every contributor who made the modifications has their own branch, and the changes aren't merged into the source code until all of them have been evaluated. Once all the changes have been given the green light, they are merged into the main source code. Version control organizes source code and boosts productivity by streamlining the development process.

Here are more impacts of version control on how a company works:

- It increases the pace of project creation by facilitating practical cooperation.
- It offers improved communication and support, increased staff productivity, expedited product delivery, and enhanced employee capabilities.
- Tracking every tiny change will reduce the risk of mistakes and disputes while the project is being developed because more than one person will review the code and ensure no errors (a third eye).

- It assists in the recovery in the event of a calamity or unforeseen circumstance.
- The most crucial point is that it tells us who, what, when, and why modifications were done.

As DevOps team members, you must know two different use cases for version control; each depends on the company. I have seen companies using both use cases simultaneously, ensuring nothing will be lost.

- A repository can be compared to a change database. It includes all of the project's modifications and historical versions.
- The personal copy of all the files in a project is a copy of the work (or checkout). You can modify this copy without harming other people's work, and when you're done, you can submit your changes to a repository.

Are these the same tool? Let's look at examples of each one of them and the pros and cons of each one of these tools.

Git

Git is one of the most popular version control systems today.

Features

- It is a model for a distributed repository.
- HTTP, FTP, and SSH are compatible with current systems and protocols.
- The history may be verified via cryptography.
- Direct object packing regularly takes place.

Pros

- Performance is both quick and efficient.
- It offers `git bash`, a fantastic command-line tool.
- Changes to the code can be monitored.

Cons

- Keyword expansion and timestamp preservation are not supported.
- If the project is big, it is tough to follow up and trace.

SVN

Features

- Metadata is versioned in a free-form manner.
- Branching is not affected by file size and is a low-cost procedure.
- Versioning is applied to directories.

Pros

- It's simple to set up and manage.
- When compared to Git, it has superior Windows support.
- It has the advantage of excellent GUI tools such as TortoiseSVN.

Cons

- The modification time of files is not saved.
- Signed revisions are not supported.

Bitbucket

Features

- Code review and comments are included in pull requests.
- Bitbucket Pipelines is a service that provides continuous delivery.
- Two-step verification and two-step verification are necessary.

Pros

- It allows you to work on the repository as a team.
- It's tightly integrated with Atlassian products like Jira and Confluence.

Cons

- It no longer supports HTTPS-based authentication and now only supports SSH.
- When there are several code conflicts, merging code might be complicated.

GitLab

Features

- Git is used for version control and repository management.
- Issue management, issue tracking, and bulletin boards are all available.
- The Review Apps tool and the Code Review feature are helpful.
- GitLab CI/CD is a CI/CD tool.

Pros

- It supports Kubernetes.
- It has an integrated CI platform with support for both default and custom runners.
- It includes helpful features such as built-in CI/CD assistance, which allows us to install and test our apps quickly.
- There are fewer code conflicts since it enables local checkout and several developers to work simultaneously on the same software file.

Cons

- The documentation could be improved.
- It is expensive.
- GitLab needs a dependable support center that is simple to find and use.

Sure, there are more tools than this, but I will not mention all of them here. I tried to cover the common ones, and I will now discuss different types of version control, because as a DevOps engineer, you need to know about each one of them.

Version Control System Types

When you are part of a development team, the team members can work on the same pieces of code during a project. As a consequence, changes made to one section of the source code may be incompatible with alterations made to the same section by a different developer working on the project simultaneously.

Let's discuss the version control types to understand more about version control.

Local Version Control Systems

This is one of the most basic types, and it uses a database to track file changes. One of the most widely used VCS tools is RCS. Patch sets (differences between files) are stored in a particular format on a disk. It can then re-create what any file looked like by adding all the fixes.

This kind of version control offers backup and testing benefits.

Centralized Version Control Systems

Centralized version control systems (CVCSs) have a single repository, with each user having a working copy (see Figure 2-11). You must commit to updating the repository with your modifications. Others may be able to view your changes if you update. To make your improvements visible to others, you'll need two things.

- You will commit.
- Others will update or request a change.

The CVCS benefits vary from one company to another, but the general one is that it is easy to set up. In addition, it provides work transparency and allows the team to follow up. On the other hand, if one server goes down, the developer can't save the work, and the remote commit is usually slow.

Distributed Version Control Systems

Distributed version control systems (DVCSs) have many repositories (see Figure 2-11). Each user has a working copy and repository. Simply committing your modifications does not provide anyone access to them. This is because the commit will only reflect those changes in your local repository, and you'll need to push them to the central repository to see them. Similarly, until you have previously pulled other people's

modifications into your repository, you do not receive other people's changes when you update.

Four elements are necessary to make your modifications apparent to others.

- You commit.
- You push.
- They pull.
- They update or request a change.

This type of version control is the most common one; it's like a two-in-one tool because of local version control. The whole history is always accessible thanks to local contributions; there is no need to access a remote server. This saves time, mainly when dealing with SSH keys, and is an excellent approach for people who work remotely or offshore.

The centralized repo stores everything in one place depending on the working directory and the file structure that the company has, and it's considered one of the simplest forms of version control; on the other hand, the distributed version controls where the code will be mirrored on every developer's computer.

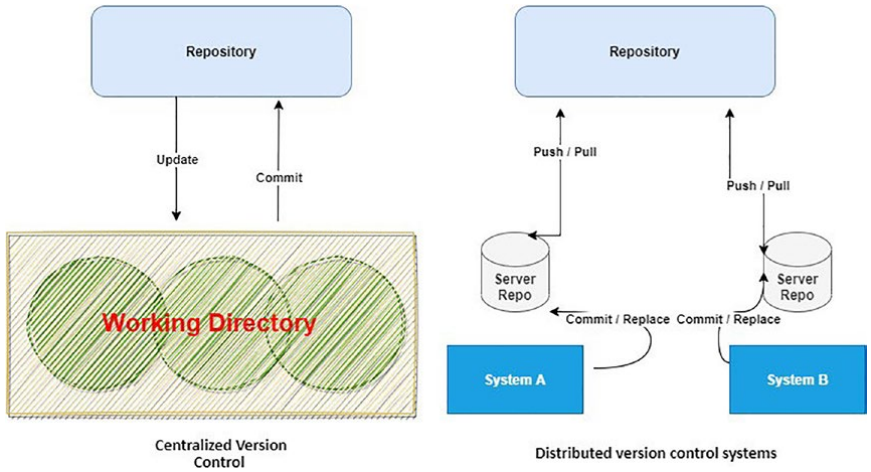

Figure 2-11. *How a centralized repo works*

Summary

In this chapter, we covered the most topics for the DevOps category and the tools that can be used; depending on each category, there are tons of tools that can be used. To summarize, whether you are a novice or an expert looking to learn more, DevOps is a mindset and a profession that allows you to be creative with no boundaries; there is always something you can improve and expand. Knowing all of the tools isn't required, but understanding how they function and how to utilize them will be a strength when implementing DevOps.

The following chapters will explain more about DevOps. For example, I didn't discuss containerization in this chapter, which is part of DevOps as well. Let's not waste any time getting to the technical part.

CHAPTER 3

AWS Services for Continuous Integration

In the previous chapter, we reviewed DevOps and AWS concepts. In this chapter, I will cover DevOps continuous integration, particularly the following core topics:

- What is continuous integration?
- Continuous integration concepts
- Amazon AWS services for CI
- Examples of these CI services

By embracing DevOps methods, AWS delivers configurable services that allow enterprises to create and deliver products quicker and more reliably.

Provisioning and managing infrastructure, deploying application code, automating software release procedures, and monitoring the performance of your application and infrastructure are all made easier with these services.

You can deploy AWS resources using different methodologies; one of these methods is infrastructure as code (IaC), which we discussed earlier. Others are Terraform, CloudFormation, and AWS Cloud Development Kit, which help a company track the resources.

AWS CodeBuild, AWS CodeDeploy, AWS CodePipeline, and AWS CodeCommit are some services that aid continuous deployment, and we'll cover each in this chapter.

Continuous Integration and Continuous Deployment

Continuous integration (CI) and continuous delivery (CD), commonly joined as CI/CD, are a culture, set of operating principles, and practices used by application development teams to deliver code changes more frequently and reliably.

O. Mustafa, *A Complete Guide to DevOps with AWS*, https://doi.org/10.1007/978-1-4842-9303-4_3

For DevOps teams, CI/CD is a best practice. In Agile methods, it's also a best practice. CI/CD allows software development teams to focus on satisfying business needs while maintaining code quality and security by automating integration and delivery.

Continuous integration developers integrate their modifications back to the main branch as often as feasible. The developer's modifications are verified by building a build and running automated tests against it. You avoid integration issues if you wait until release day to integrate changes into the release branch.

When new commits are incorporated into the main branch, continuous integration focuses on testing automation to ensure the application is not damaged.

This is considered a programming methodology and technique set that regularly encourages development teams to commit tiny code changes to a Git repository. Because most current applications include writing code on various platforms and tools, groups want a standardized method for integrating and validating changes. Continuous integration allows developers to efficiently create, package, and test their applications.

Developers are more likely to commit code changes more often when they have a consistent integration procedure, which leads to improved cooperation and code quality.

Continuous delivery is an extension of continuous integration. It distributes all code changes to a testing/production environment immediately after the build step, which means you have an automatic release process and automated testing. You can deploy your application at any moment by just pressing a button.

In principle, continuous delivery allows you to distribute daily, weekly, biweekly, or monthly, depending on your company's needs. Suppose you genuinely want to reap the advantages of continuous delivery.

In such instances, you should deploy to production as quickly as possible to guarantee that little batches of code are published that are simpler to debug in the case of a problem.

The environment-specific parameters that must be bundled with each delivery are stored using CI/CD technologies. After that, CI/CD automation performs any required service calls to web servers, databases, and other services that must be restarted. Following deployment, it may also run additional tasks.

CI/CD necessitates *continuous testing*, as the goal is to produce high-quality code and apps. Automated regression, performance, and other tests are run in the CI/CD pipeline during continuous testing.

CI/CD and DevOps

Every DevOps approach is built on the foundation of continuity. Continuous integration, continuous delivery, and continuous deployment are all part of the DevOps process. Let's take a closer look at each in Figure 3-1, which offers a snapshot of which stage each one of them is applied in and the advantages of incorporating them into the software development process.

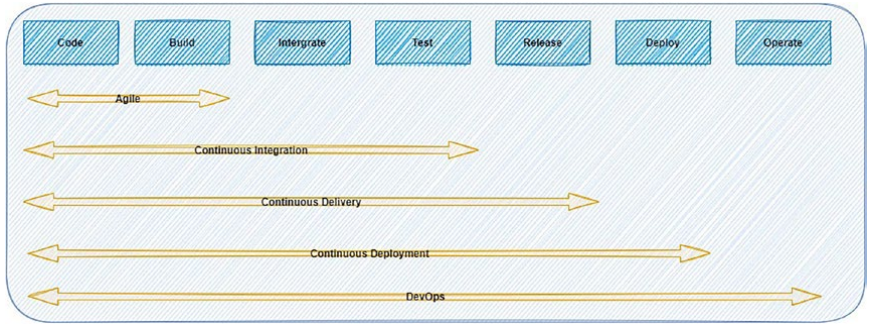

Figure 3-1. *CI/CD big picture*

One benefit of CI/CD is being able to automate things. For example, if CI/CD is applied, there is no need to interrupt production for releases; therefore, you can build quickly. Deployment pipelines are triggered automatically for every change. Because you deploy small batches of changes, releases are less risky and simpler to correct in the event of a problem; plus, customers see a steady stream of changes, and quality improves daily than monthly, quarterly, or annually.

Each process comes with specific requirements. To implement continuous integration in your company, you need the following:

- Your team must build automated tests for each new addition, enhancement, or bug correction.
- You'll need a continuous integration server to monitor the main repository and execute tests on every new commit.
- Developers should integrate modifications as often as feasible, at least once a day.

Satisfying these requirements will provide the following benefits to a company:

- As automated testing catches regressions early, fewer problems are submitted to production.
- The release is simple since all integration difficulties were addressed early on.
- Developers are warned when they break the build and can concentrate on repairing it before moving on to another activity, resulting in less context switching.
- Testing costs are dramatically lowered, as your CI server can execute hundreds of tests in seconds.
- The QA team will be able to spend less time testing the code and more time improving the quality culture and the code itself.

Next, before implementing continuous delivery, you'll need to address the following:

- Continuous integration requires a solid foundation, and your test suite must cover enough with your code.
- Automated deployments are required. Although the trigger is still manual, human involvement should not be needed after the deployment.
- Your team will undoubtedly require feature flags to ensure that unfinished features do not affect consumers in production.

Satisfying these requirements and implementing continuous delivery offers these benefits:

- The difficulty of software deployment has been removed. Your team no longer needs to spend days preparing for a release.
- You can release your code more often.
- There will be less pressure on the development team, especially for small changes.

Lastly, here are the elements you will need to consider before implementing continuous deployment:

- The environment should be implemented as the best practice to avoid many issues; if this is not done later, it can affect the automation quality, which is supposed to simplify the infrastructure.
- Documentation, documentation, documentation: it must stay current with the deployment schedule.
- Define the process of each release and what changes should be made such as feature flags, bug flags, etc.

By addressing all of these needs, there's no need to interrupt production for releases, and for every modification, deployment pipelines are launched automatically.

This means you can build quickly. Installing and maintaining a continuous integration server are typical continuous integration costs. But you can significantly reduce the cost of adopting these practices by using a cloud service, which we will discuss in this chapter.

Jenkins users, for example, establish their pipelines in Jenkinsfiles, which explain several steps such as build, test, and deploy. The file declares environment variables, choices, secret keys, certificates, and other factors, subsequently referenced in stages. Error circumstances and alerts are handled in the post section.

Build, test, and deploy are the three steps of a typical continuous delivery pipeline. At various phases, the following activities might be included:

- You can execute a build after cloning the code from version control.
- You can use stage gates to automate the needed and supporting approvals when required.
- All essential infrastructure tasks are automated as code to set up or pull down cloud infrastructure.
- You can get code to run in the desired environment (staging, QA, Prod).
- Managing and customizing environment variables depend on the environment.

- The components of an application are pushed to their relevant services.
- Updating the configuration depends on the infrastructure side.

In the next section, we'll explore cloud services covering CI/CD; specifically, we'll focus on Amazon Web Services, specifically, what services you need to use and how to configure them to meet your company's needs.

Note You'll find the projects I cover in this book much easier to follow if you already have an AWS account. For more information on signing up for an AWS account, navigate to `https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-sign-up-for-aws.html`.

Continuous Integration

I mentioned CI, a software development method in which developers regularly merge code changes into a shared repository before running automated builds and testing. CI shortens the time to detect and fix defects, enhances software quality, and approves and deploys new software upgrades.

AWS provides the following three services that relate to CI.

AWS CodeCommit

AWS CodeCommit (Figure 3-2) is a managed source control service that hosts private Git repositories and is safe and highly scalable. CodeCommit removes the need for you to run your source control system, and there is no hardware to provide or scale and no software to install, set up, or run. CodeCommit can be used to save everything from code to binaries, and it supports all of Git's essential features, enabling it to operate in tandem with your current Git-based tools. Your team may also use CodeCommit's online coding tools to read, modify, and work together on projects.

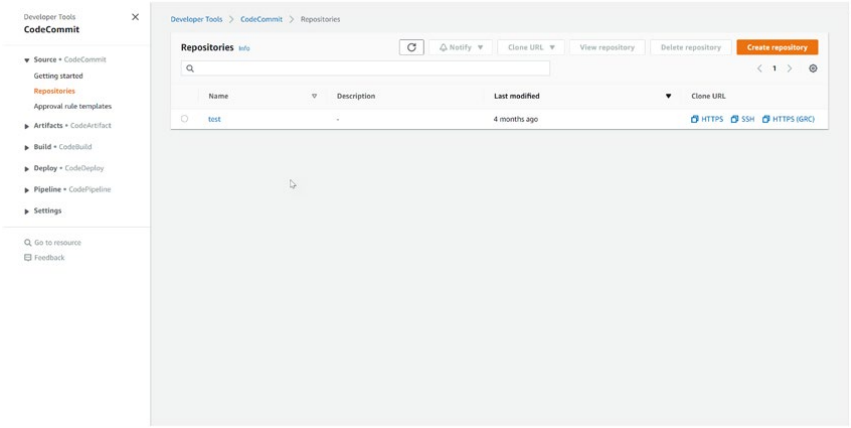

Figure 3-2. *AWS CodeCommit*

AWS CodeCommit provides several advantages.

- You can use HTTPS or SSH to send your files to/from AWS CodeCommit. Your repositories are automatically secured at rest using customer-specific keys through AWS Key Management Service (AWS KMS).
- CodeCommit uses AWS Access Control AWS Identity and Access Management (IAM) to govern and monitor who has access to your data and how, when, and where they have access. CodeCommit now uses AWS CloudTrail and Amazon CloudWatch to help you keep track of your repositories.
- AWS CodeCommit saves your repository in Amazon Simple Storage Service (Amazon S3) and Amazon DynamoDB for high availability and durability. Your protected data is kept in numerous locations for redundancy. Thanks to this design, the data in your repository will be more available and more durable.
- You may now be notified when anything happens in your repositories. Amazon Simple Notification Service (Amazon SNS) notifications will be used for notifications. Each notice will provide a status message and a link to the resources that caused the notification

to be produced. You can also use AWS CodeCommit repository triggers to send alerts, generate HTTP webhooks, and call AWS Lambda functions in response to your selected repository events.

- AWS CodeCommit is a collaborative software development platform. You can commit, branch, and merge your code, allowing you to keep track of the projects in your team. Pull requests, which offer a method for requesting code reviews and discussing code with collaborators, are now supported by CodeCommit.

Creating a CodeCommit Repository on AWS

You can create a new CodeCommit repository using the AWS CodeCommit GUI or AWS Command Line (AWS CLI). After you've created a repository, you can add tags to it.

Let's start with the GUI option.

Create a Repository (Console)

To create a CodeCommit repository using the GUI, you need to follow these steps:

1. From the AWS Portal, search for AWS *CodeCommit*; it's essential to choose the correct region for you (Figure 3-3).

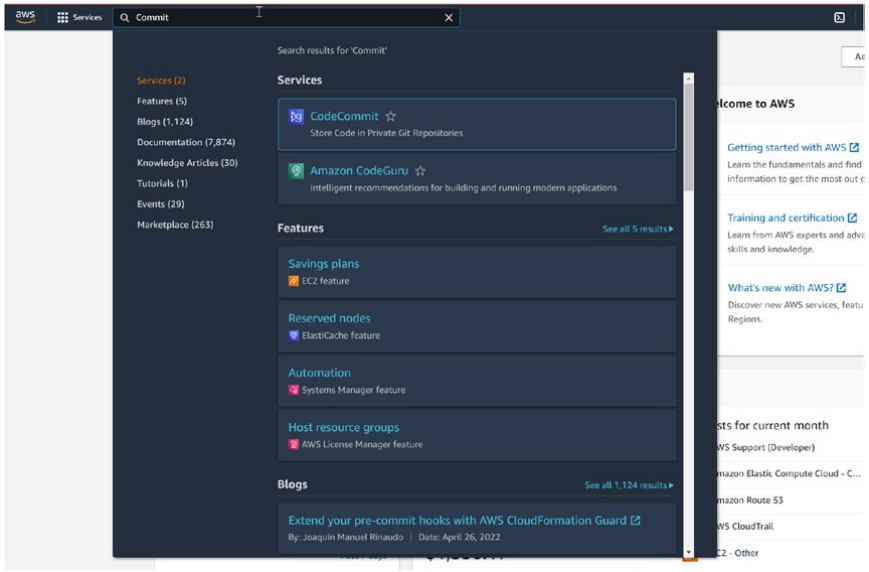

Figure 3-3. *Choosing CodeCommit from the AWS Portal*

2. Choose to create a repository from the page, as shown in Figure 3-4.

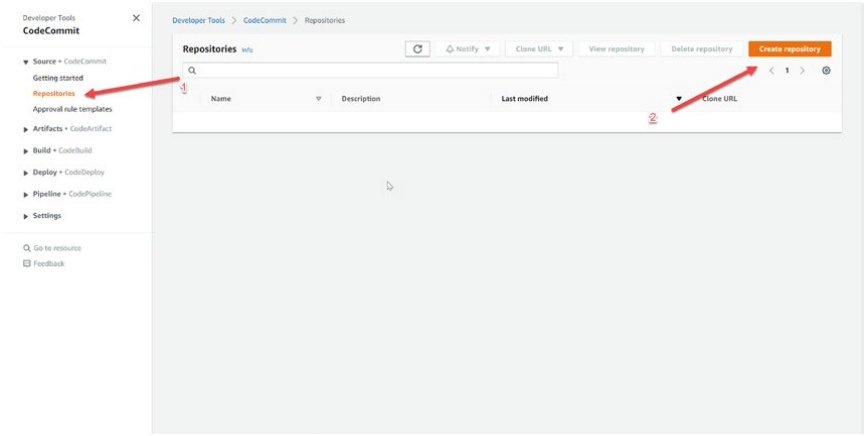

Figure 3-4. *Creating a CodeCommit repository*

3. In the Create repository dialog (Figure 3-5), specify a name for the repository in the "Repository name" field.

   The description is optional, but I usually prefer to write something explaining what this repo is for and which project, especially if you have more than one; tags are essential to make AWS management more efficient. When you are done, click Create.

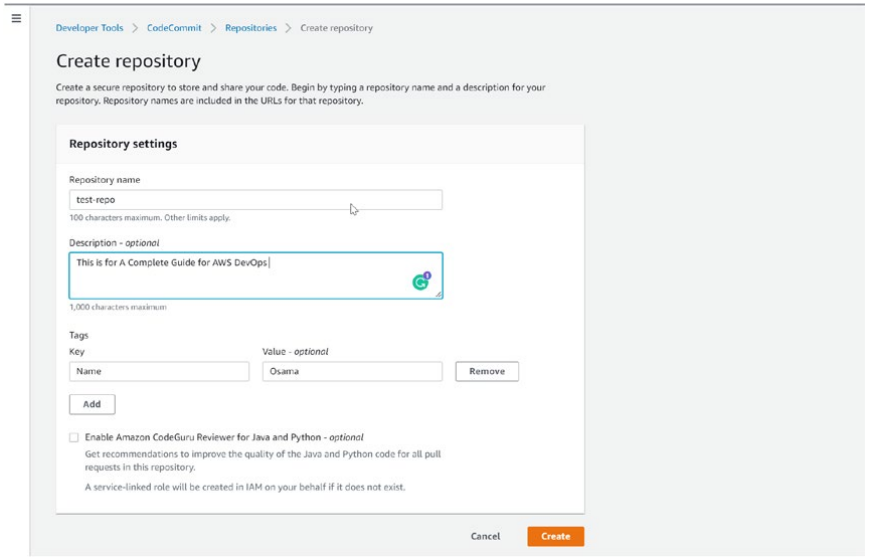

Figure 3-5. *Choosing the repository name*

> **Note** Case is important when naming repositories. Your AWS account's name must be unique in the AWS region.

Create a Repository (AWS CLI)

The first step to working with the AWS CLI is configuring the command-line tools to allow you to connect to the AWS Portal and deploy things, so let's quickly go through that process.

1. Open the IAM console (Figure 3-6) in the AWS Management Console after logging in.

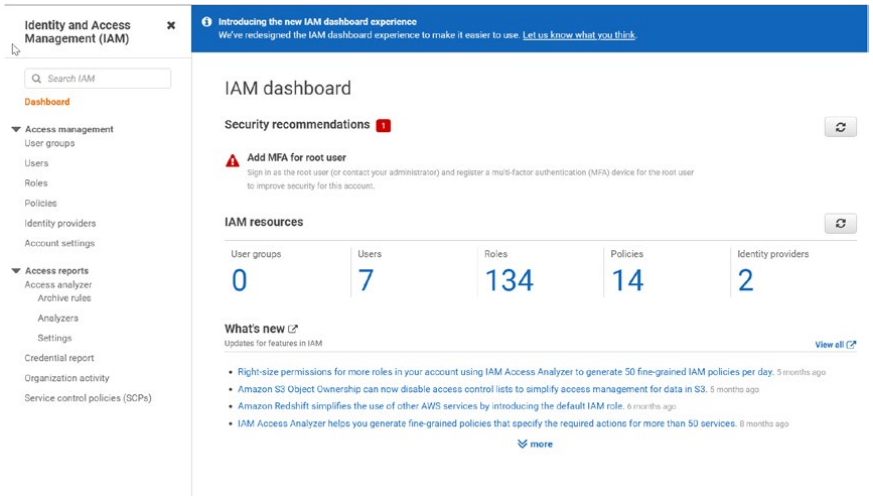

Figure 3-6. *IAM page*

2. After creating a user and assigning them the appropriate permissions, select the user. The new screen will be shown in Figure 3-7; choose the security credentials and click "Create access key."

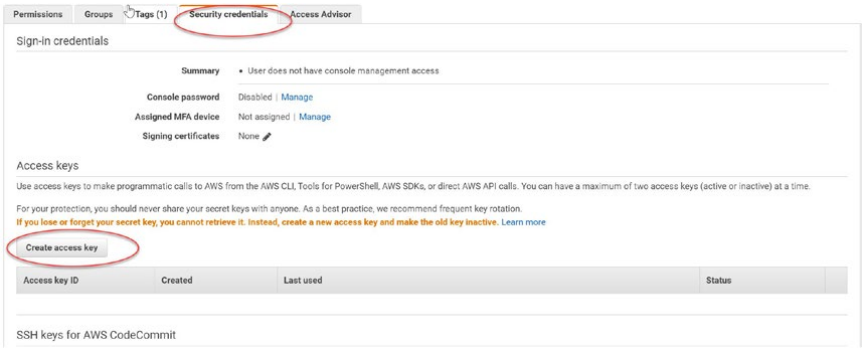

Figure 3-7. *Creating access key and secret key*

3. Choose Show to see the new access key pair. After this dialog box closes, you can no longer access the secret key. This is how your credentials will appear.
   - *Access key ID*: AKIOMKOUSMP7COURSE
   - *Secret access key*: kdasdasm9nmasdc/lkCOURSEKEY
4. *Optional*: Choose the `Download.csv` file to get the key pair. Place the keys in a safe place. After this dialog box closes, you can no longer access the secret key.

Now you have the access key and secret key, but before moving on, let's also go through the installation process for the AWS CLI. I prefer using v2 instead of v1.

When using the latest Amazon command-line interface (CLI) version, all timestamp response values are returned in ISO 8601 format. In the first release of the Amazon CLI, commands returned timestamp values in whatever format the HTTP API response supplied. This format might be different for each service.

To Install the AWS CLI, you can check the AWS documentation at `https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html`.

Once you install the tool on your machine, you can set it up; it's simple using the following snippet:

```
1. $ aws configure
2. AWS Access Key ID [None]: AKIOMKOUSMP7COURSE
3. AWS Secret Access Key [None]: kdasdasm9nmasdc/lkCOURSEKEY
4. Default region name [None]: eu-west-1
5. Default output format [None]: json
```

As a DevOps team member, you need to understand the AWS CLI, and it's highly recommended to use it instead of the GUI since the GUI has some limitations; it's mandatory to mention this is the direct way to configure the CLI.

With experience, CLI setup may be much quicker than GUI configuration. The user may set up the network's inbound and outbound connections and any necessary routing protocols and access controls with only a few simple instructions. With a graphical user interface, accessing these features would need many clicks and a search for the appropriate menus and panels.

Imagine, for example, having multiple accounts under the AWS organization, a service from Amazon AWS that allows companies to have various accounts such as Prod, Dev, and QA; in that case, you may be the easiest way to use AWS SSO.

After configuring the AWS CLI, you can run the commands to create the repository, but there are some elements you should consider first.

- Choose a name that distinguishes the CodeCommit repository from others using the `--repository-name` option; this name must be unique across all AWS services.
- The `--repository-description` attribute is optional and will allow you to add comments or explain how this repository is being used.
- Another optional attribute is `--tags`, which, naturally, allows you to add tags and key values.

The command that will be used to create the code commit repository is shown here:

```
1. aws codecommit create-repository --repository-name test-repo
--repository-description "first repo for AWS DevOps" --tags Team=DevOps
```

If the command ran successfully without any issues, the output would have the following information:

```
 1. {
 2.     "repositoryMetadata": {
 3.         "repositoryName": "test-repo",
 4.         "cloneUrlSsh": "ssh://git-codecommit.eu-west-1.amazonaws.com/
            v1/repos/test-repo",
 5.         "lastModifiedDate": 1336081622.594,
 6.         "repositoryDescription": "first repo for AWS DevOps",
 7.         "cloneUrlHttp": "https://it-codecommit.eu-west-1.amazonaws.com/
            v1/repos/test-repo",
 8.         "creationDate": 1336081622.59,
 9.         "repositoryId": "d897513-c95w-1002-aaef-799cBook",
10.         "Arn": "arn:aws:codecommit:eu-west-1:111111111111:test-repo",
11.         "accountId": "111111111111"
12.     }
13. }
```


One of the most common cases using the AWS CLI is that the DevOps team member forgot the name or the ID. To retrieve them using the CLI, follow these steps:

1. To see a list all of CodeCommit repositories, run the following; you can use `--sort-by` or `--order`:

   ```
   1. aws codecommit list-repositories
   ```

   The output will be formatted as JSON, like the one below.

   ```
    1. {
    2.     "repositories": [
    3.         {
    4.             "repositoryName": "test-repo"
    5.             "repositoryId": "d897513-c95w-1002-aaef-799cBook",
    6.         },
    7.         {
    8.             "repositoryName": "Book-repo"
    9.             "repositoryId": "fgf0024c-dc0v-44dc-1191-799cBook"
   10.         }
   11.     ]
   12. }
   ```

2. If you need more information about a single CodeCommit repository, use this:

   ```
   1. aws codecommit get-repository --repository-name test-repo
   ```

   The output will be JSON, as the previous is related to the repository.

3. To get information about numerous CodeCommit repositories, use this:

   ```
   1. aws codecommit batch-get-repositories --repository-names
   test-repo book-repo
   ```

If it is successful, this command produces a JSON with the following information: any CodeCommit repositories that could not be located and the CodeCommit repositories that can be found with information such as repository name, repository description, repository's unique ID, and account ID.

When you initially link to a CodeCommit repository, you usually clone the repository's contents to your local workstation. Straight from the CodeCommit console, you may add and modify files in a repository. You may also add a CodeCommit repository as a remote repository if you already have a local repository. This section explains how to connect to a CodeCommit repository.

Before connecting to CodeCommit, these are the prerequisites:

- You must install the necessary software such as Git and settings on your local computer to be able to link to CodeCommit. You can do this at `https://git-scm.com/book/en/v2/Getting-Started-Installing-Git`.
- You'll need the clone URL for the CodeCommit repository you'd want to connect to. To access information about available repositories, use the AWS CodeCommit GUI, AWS CLI, or Git from a local repo linked to the CodeCommit repository.

Once you set up Git, the user needs to be configured to use CodeCommit; from IAM, choose the username, and under the username, choose the security credentials; check Figures 3-8 and 3-9; both show how to configure the needed to grant access to the user.

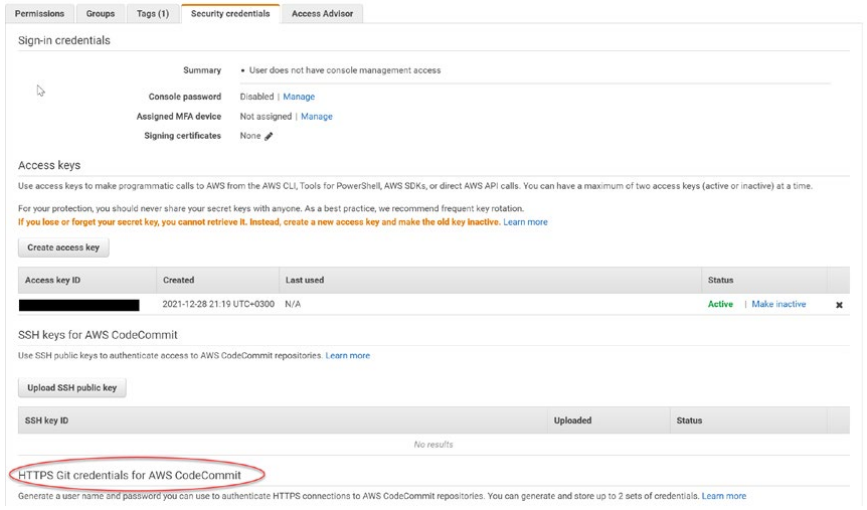

Figure 3-8. *IAM user information*

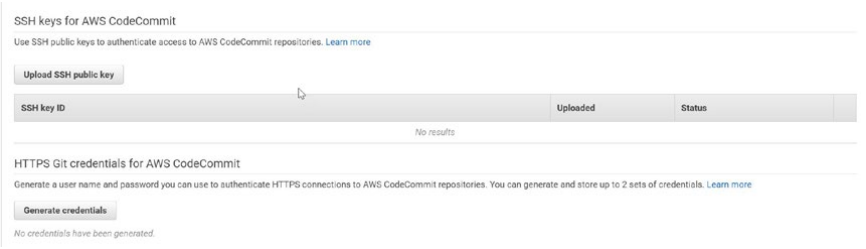

Figure 3-9. *IAM code commit configuration*

Copy the username and password IAM produced for you by displaying, copying, and pasting this information into a secure file on your local computer or selecting "Download credentials" to download this information as a `download.csv` file. To connect to CodeCommit, you'll need this information.

Now to connect, you have two options.

1. Connect to the CodeCommit repository by cloning the repository.
2. Connect a local/existing repo to the CodeCommit repository.

Let's start with the first part, connecting using the clone option:

3. Once you complete the prerequisites, you can choose any directory or location on your PC, Linux, Mac, or Windows and run the clone commands; remember `First-Repo`, the folder name.

```
1. git clone https://git-codecommit.eu-west-1.amazonaws.com/v1/
repos/test-repo First-repo
```

You can also configure the AWS CLI to set a default profile and region; I don't prefer that, especially if you have multiple accounts and need to switch between them.

You will use a source profile we will discuss later in this book, using `git-remote-codecommit` via HTTPS, with the AWS CLI default profile and AWS region set as follows:

```
1. git clone codecommit://test-repo First-repo
```

Some companies disable HTTPS for security reasons and allow only SSH; in that case, each DevOps within the team needs to provide a public key, such as this:

```
1. git clone ssh://git-codecommit.eu-west-1.amazonaws.com/v1/repos/test-
repo First-repo
```

Figure 3-10 shows that the test-repo has been created and the options that we have to clone the repository.

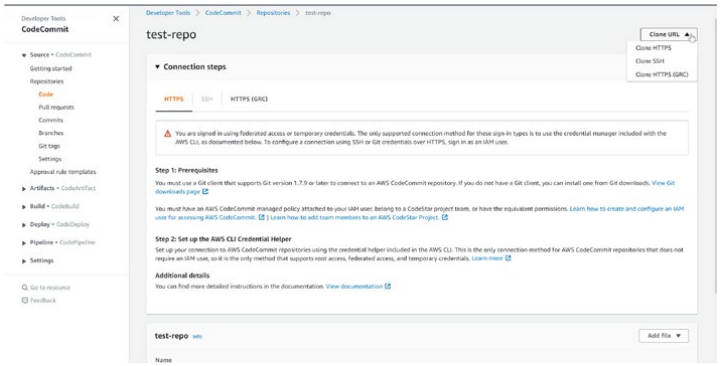

Figure 3-10. *Repository information*

Connect a Local/Existing Repo to the CodeCommit Repository

There are two other options that pertain to creating a CodeCommit repo: if you currently have a local repository and want to make a CodeCommit repository the remote repository, ad if you already have a remote repository and want to push your changes to both CodeCommit and that remote repository.

One of the most common use cases is to keep utilizing your current Git repository solution while experimenting with AWS CodeCommit.

1. Run the Git `remote -v` command from your local repo directory. It would be best if you got something like this as a result:

   HTTPS

   ```
   1. origin  https://git-codecommit.eu-west-1.amazonaws.com/v1/
   repos/test-repo (fetch)
   2. origin  https://git-codecommit.eu-west-1.amazonaws.com/v1/
   repos/test-repo (push)
   ```


   SSH

   ```
   1. origin  ssh://git-codecommit.eu-west-1.amazonaws.com/v1/
   repos/test-repo  (fetch)
   2. origin  ssh://git-codecommit.eu-west-1.amazonaws.com/v1/
   repos/test-repo  (push)
   ```

2. Once you run the previous command, run the following command, which specifies where you want to host your code:

   ```
   1. git remote set-url --add --push origin some-URL/repo-name
   ```

   To modify a Git remote origin using an HTTPS URL, one must first open the Git terminal and verify the current remote URL, for example.

   ```
   1. git remote set-url --add --push origin  https://git-
   codecommit.eu-west-1.amazonaws.com/v1/repos/test-repo
   ```

3. Rerun the `git remote -v`.

   HTTPS

   ```
   1. origin  https://git-codecommit.eu-west-1.amazonaws.com/
   v1/repos/test-repo (fetch)
   2. origin  URL/RepoName (push)
   ```

   SSH

   ```
   1. origin  ssh://git-codecommit.eu-west-1.amazonaws.com/v1/
   repos/test-repo  (fetch)
   2. origin  URL/RepoName  (push)
   ```

4. The CodeCommit repository should now be added; run `git remote set-url --add --push origin` again but with the repository URL.

   HTTPS

   ```
   1. git remote set-url --add --push origin https://git-
   codecommit.eu-west-1.amazonaws.com/v1/repos/test-repo
   ```

SSH

```
1. git remote set-url --add --push origin ssh://git-
codecommit.eu-west-1.amazonaws.com/v1/repos/test-repo
```

5. Run the `git remote -v` command again.
6. Make sure you're pushing to both remote repositories now by adding a dummy text file.
7. After creating the text, run the following:

```
1. Git add .
2. git add dummy.text
```

8. Run the following command and choose a proper comment:

```
1. git commit -m "Added dummy text file for texting."
```

9. You need to push this file from the local to the CodeCommit repository. To do this, run the following command:

```
1. git push -u remote-name branch-name
```

Once you run the previous command, the file will be uploaded, and you will see output like the following (remember this for HTTPS; it is almost the same for SSH):

```
 1. Counting objects: 5, done.
 2. Delta compression using up to 4 threads.
 3. Compressing objects: 100% (3/3), done.
 4. Writing objects: 100% (3/3), 5.61 KiB | 0 bytes/s, done.
 5. Total 3 (delta 1), reused 0 (delta 0)
 6. To URL/RepoDestation
 7.    a5ba4ed..250f6c3  main -> main
 8. Counting objects: 5, done.
 9. Delta compression using up to 4 threads.
10. Compressing objects: 100% (3/3), done.
11. Writing objects: 100% (3/3), 5.61 KiB | 0 bytes/s, done.
12. Total 3 (delta 1), reused 0 (delta 0)
```



```
13. remote:
14. To https://git-codecommit.eu-west-1.amazonaws.com/v1/repos/
test-repo
15.    a5ba4ed..250f6c3  main -> main
```

View Repository Details (Console)

Follow these steps to see some details about the repository:

1. Open the CodeCommit console as mentioned before in this chapter.
2. Select the repository's name from the Repositories menu.
3. Choose the Clone URL and the protocol to clone the repository.
4. Choose Settings in the navigation pane to see the settings for the repository and data, such as ARN and ID.

View CodeCommit Repository Details (AWS CLI)

It's crucial as DevOps team members to know all the available options; sometimes, you need to use CLI for things not supported using the GUI and Git commands previously mentioned in this chapter.

I prefer to use CLI since it's everything you need; once you start using CLI, you will find everything else easy.

Now dealing with CodeCommit, it's like you are dealing with Git, but for AWS, which is the same concept. However, in this case I need to change the settings for the repository.

After going through a couple of code commit configurations, you can share a CodeCommit repository with other users once you've established it. First, choose the protocol, either HTTP or SSH, to propose to users when cloning and connecting to your repository using a Git client or an IDE.

The URL and connection details should then be sent to the people you want to share the repository. You may also need to create an IAM group, apply managed policies to that group, and update IAM rules to refine access, depending on your security requirements.

Here is a summary of what is needed by both protocols:

HTTPS

1. Install the Git tool on your local PC.
2. Generate HTTPS credentials for the users, which were covered earlier.

SSH

1. It would help if you generated a public-private key using one of the tools, such as Putty keygen.
2. Save these keys for later.
3. The public key will be connected with your cloud IAM user.
4. Configure the local PC, as mentioned earlier.

View CodeCommit Repository Details (Git)

You should be familiar with Git commands to connect and start using the code commit using Git commands.

Run the `git remote show remote-name` command from a local repo to get information about CodeCommit repositories.

The output for this command will look like the following, formatted for HTTPS:

```
1. remote origin
2.   Fetch URL: https://git-codecommit.eu-west-1.amazonaws.com/v1/repos/
     test-repo
3.   Push  URL: https://git-codecommit.eu-west-1.amazonaws.com/v1/repos/
     test-repo
4.   HEAD branch: (unknown)
5.   Remote branches:
6.     MyNewBranch tracked
7.     main tracked
8.   Local ref configured for 'git pull':
9.     MyNewBranch merges with remote MyNewBranch (up to date)
10.  Local refs configured for 'git push':
11.    MyNewBranch pushes to MyNewBranch (up to date)
12.    main pushes to main (up to date)
```


Configuring Notifications for AWS CodeCommit

It's imperative to understand code commit basics and the configuration; you may establish notification rules for a repository to email users about repository event types you define. When occurrences fit the notification rule parameters, notifications are delivered. To utilize notifications, you can either establish an Amazon SNS subject or use one already in your Amazon Web Services account. Utilize the CodeCommit console or the AWS CLI to set up notification rules.

From the AWS Portal, go to CodeCommit and choose the repository you want to set up notifications for, as shown in Figure 3-11.

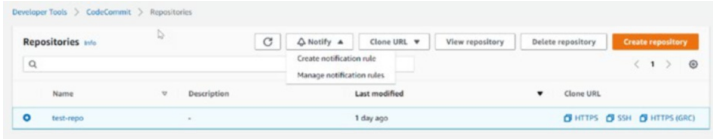

Figure 3-11. *AWS CodeCommit notification*

Next, click "Create notification rule." A new screen will open, as shown in Figure 3-12. Enter a name for the rule in the "Notification name" field.

For "Detail type," you have two options.

5. Basic if you want only the information provided to Amazon EventBridge included in the notification
6. Full if you want to include information provided to Amazon EventBridge and the CodeCommit or notification manager might supply it

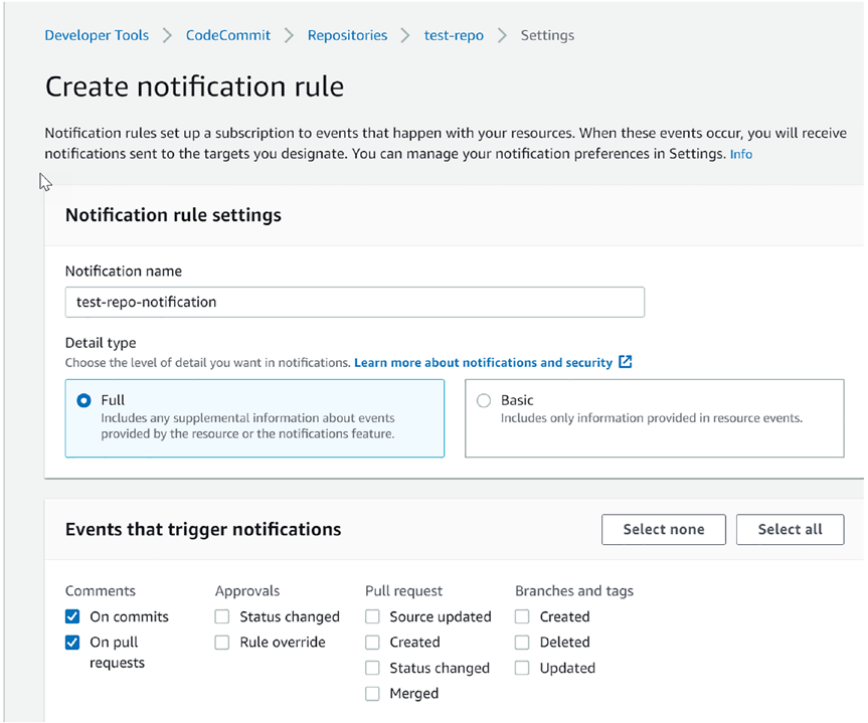

Figure 3-12. *CodeCommit notification settings*

The next step is to select the events you want to send alerts under events that trigger notifications; for the target type, you have two options if you have already configured the target.

7. *Slack*: If you want to get a notification for this repository on the Slack Channel, use the webhook.
8. *SNS topic*: This is provided by AWS, which allows you to send emails or SMS.

Figure 3-13 illustrates these options.

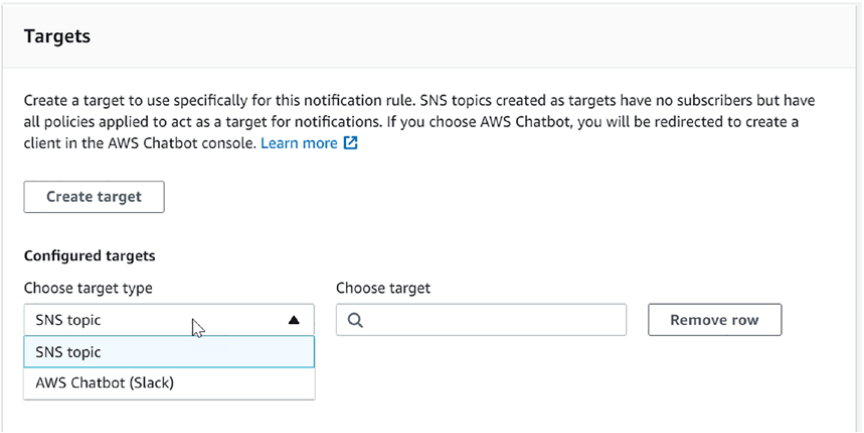

Figure 3-13. *Notification target*

I've now covered the elements essential to using CodeCommit, but know that it's a vast topic I encourage you to explore further. For the needs of this book, however, you should now be set in terms of using CodeCommit.

Now that you've established your CodeCommit repository, it's time to start coding. We'll be utilizing AWS CodeBuild.

AWS CodeBuild

AWS CodeBuild is a managed service continuous integration solution that generates code, performs tests, and creates ready-to-deploy software packages. You don't have to maintain, scale, or provision your development servers. CodeBuild may utilize Git or any version control as a source provider.

CodeBuild grows indefinitely and can handle several builds at once. CodeBuild provides several preconfigured environments for Windows and Linux. Customers may also use Docker containers to transport their customized build environments. Open-source technologies like Jenkins and Spinnaker are also integrated with CodeBuild.

Unit, functional, and integration tests may all be reported using CodeBuild. These reports show how many test cases were run and whether they succeeded or failed. The build process may also occur if your integrated services or databases are placed within a VPC.

Your build artifacts are secured with customer-specific keys maintained by the KMS using AWS CodeBuild. You may provide user-specific rights to your projects with IAM.

Like any other service, Code build provides different benefits for any company, such as the following:

9. CodeBuild sets up, patches, updates, and manages your build servers.
10. CodeBuild grows to match your build requirements on demand. You pay for the amount of construction time you use.
11. For the most common programming languages, CodeBuild offers predefined build environments. To begin your first build, all you have to do is point to your build script.

How CodeBuild works

Like any other services on Amazon AWS, you can manage/run code build using different ways:

12. Console/GUI
13. AWS CLI
14. AWS SDK
15. CodePipeline

All of them will lead to the same results, but it's good to know them; see Figure 3-14.

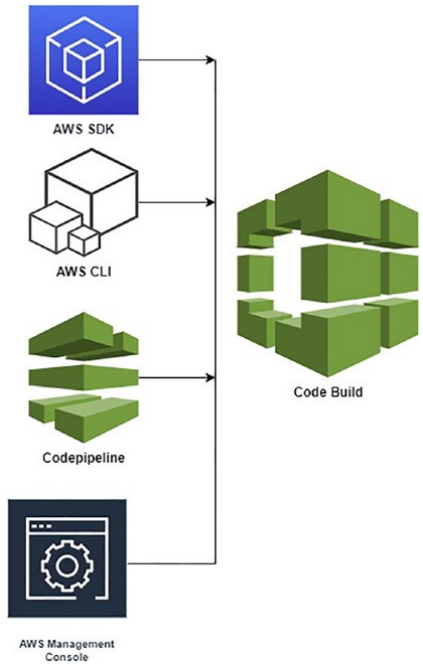

Figure 3-14. *Which AWS services you can use to run CodeBuild*

Before you start using CodeBuild, you need to answer questions that will help you understand more about what you will do with the configuration and setup.

Where will your code be stored?

CodeBuild supports a different version/source code repository, but the build specification (*build spec*) file must be included in the source code so the code build will understand the stages.

A *build spec* is a YAML-formatted collection of build commands and associated parameters used by CodeBuild to conduct a build.

The following are the supported version control/source code repositories:

16. CodeCommit
17. Amazon S3

18. GitHub
19. Bitbucket

Which build commands should you use, and in what sequence should you execute them?

When we talk about the commands, buildspec files are necessary; they indicate how these commands will be executed, and in which order.

What do you need to complete the build, either tools or runtimes?

You code in Python, Java, or even Ruby.

Do I need an extra package such as Maven, or am I good with what I have now?

The following is an example of what buildspec look like; this a sample for a small Node.js application. Remember, it's YAML.

```
version: 0.2
phases:
  install:
    runtime-versions:
      nodejs: 10
    commands:
      - echo Installing Mocha...
      - npm install -g mocha
  pre_build:
    commands:
      - echo Installing source NPM dependencies...
      - npm install
      - npm install unit.js
  build:
    commands:
      - echo build started on 'date'
      - echo Compiling the Node.js code
      - mocha test.js
  post_build:
    commands:
      - echo build completed on 'date'
artifacts:
```



```
  files:
    - app.js
    - index.html
    - package.json
    - node_modules/async/*
    - node_modules/lodash/*
```

Now that you have a high-level understanding of CodeBuild, let's explore it in more detail. First up is how to use CodeBuild with a code pipeline. As was the case with setting up the repository, you have multiple options, and again we'll focus on using either the GUI or AWS CLI.

Using CodeBuild via the Console

CodeBuild services allow DevOps to build an entirely different version depending on the configuration so that the release will be known to the company.

For example, the company added a new feature called a *menu*, known as a code change. This was added so that CodeBuild will generate from the source code a release called `menu-v1.0.zip`; this release will be deployed later, and so on.

Configuring CodeBuild is straightforward, but sometimes the setup differs from one company to another.

As you can see later in this chapter, some integrate cloud build with various tools or services such as CodePipeline or Jenkins to automate the whole process. There are common steps you need to follow, shown here:

1. You need to have source code.
2. Create a buildspec file.
3. Create an S3 bucket (the best practice is two buckets).
4. Upload your source code to version control such as GitHub or CodeCommit.
5. Create and configure the build project.
6. Test the build project by running it.
7. Check the summary information.
8. Check the output for the generated artifact.

Step 1: You Need to Have the Source Code

Let's go with the first step, which is the source code; either you will create a free account on GitHub and upload the code you want to use there or you will go to my GitHub account at https://github.com/OsamaOracle.

Clone the repo called CodeBuild-repo-example (https://github.com/OsamaOracle/CodeBuild-repo-example).

Inside the repository, you can check and see Java code plus the buildspec file.

See Figure 3-15, which shows the repository structure.

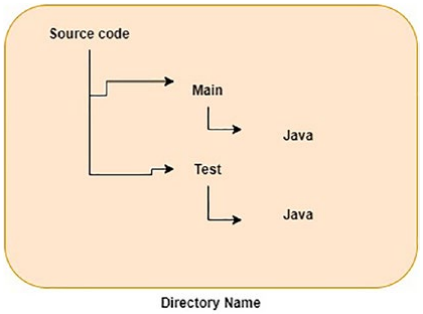

Figure 3-15. *Repo folder structure*

Step 2: Create a Build Specs File

At this process stage, you will construct a build specification file, also known as a *build spec*. CodeBuild utilizes this document to execute a build. A build spec is a collection of instructions and parameters formatted in YAML.

CodeBuild cannot correctly transform your build input into build output or identify the build output artifact in the built environment so that it may be uploaded to your output bucket if you do not provide it with a build spec.

```
version: 0.2

phases:
  install:
    runtime-versions:
      java: corretto11
```



```
  pre_build:
    commands:
      - echo nothing to do in the pre_build phase...
  build:
    commands:
      - echo Build started on 'date'
      - mvn install
  post_build:
    commands:
      - echo build completed on 'date'
artifacts:
  files:
    - target/messageUtil-1.0.jar

```

Step 3: Create an S3 Bucket

In a straightforward step, you can create two buckets, which will allow you to understand and organize the output.

Step 4: Upload Your Code

You must upload the code to the version control tool, such as GitHub, Bitbucket, or CodeCommit.

Step 5: Create the Code Build Project

AWS CodeBuild performs the build when you create a project, so let's do this:

1. Go to the CodeBuild service from the console, as shown in Figure 3-16.

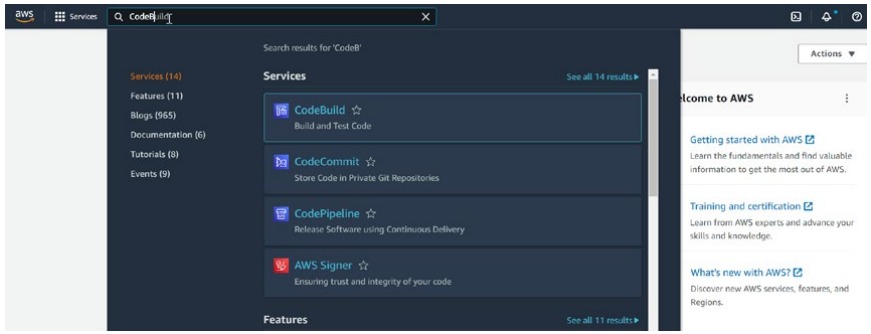

Figure 3-16. *AWS Portal*

2. Create the project, as shown in Figure 3-17.

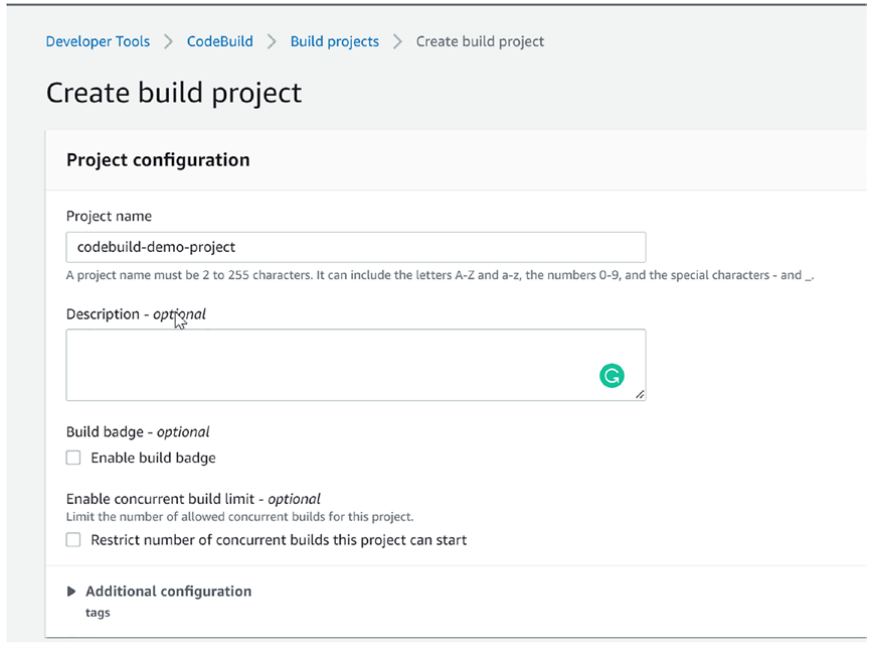

Figure 3-17. *Creating a project*

Using CodeBuild via the AWS CLI

With the help of the create-project command and the –generate-cli-skeleton option, you may construct a skeleton JSON file:

```
1. aws codebuild create-project --generate-cli-skeleton
```

The output of the first command will generate a JSON file and copy the JSON file.

Switch to the directory where you stored the file and then execute the `create-project` command again.

```
 1. {
 2.   "name": "codebuild-demo-project",
 3.   "source": {
 4.     "type": "S3",
 5.     "location": "codebuild-region-ID-account-ID-input-bucket/
        MessageUtil.zip"
 6.   },
 7.   "artifacts": {
 8.     "type": "S3",
 9.     "location": "codebuild-region-ID-account-ID-output-bucket"
10.   },
11.   "environment": {
12.     "type": "LINUX_CONTAINER",
13.     "image": "aws/codebuild/standard:4.0",
14.     "computeType": "BUILD_GENERAL1_SMALL"
15.   },
16.   "serviceRole": "serviceIAMRole"
17. }
```

Please copy the previous code, and modify it depending on the project name, location of the bucket, and environment configuration as the project needs.

Save this file inside the exact repository location (GitHub, Bitbucket, etc.) and run the following command:

```
1. aws codebuild create-project --cli-input-json file://create-project.json
```

Execute the create-project command once again while specifying your JSON file, and the project will be created.

Next is the project's source; as mentioned, we created an S3 bucket before so that will be our source, check Figure 3-18.

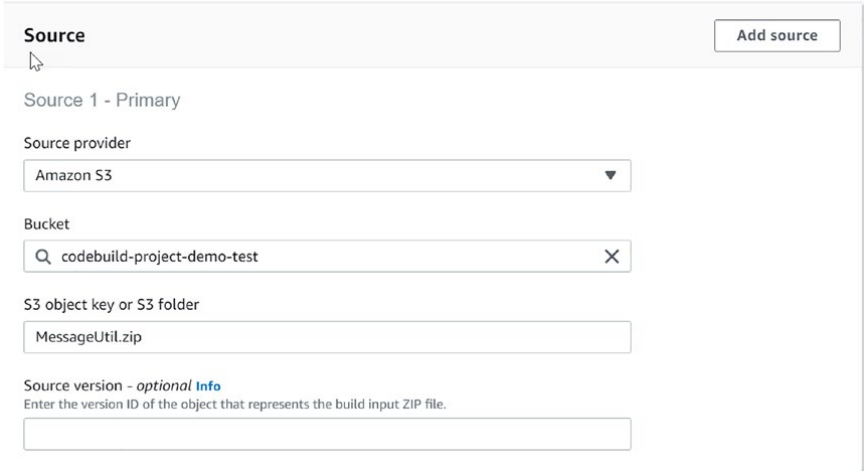

Figure 3-18. *CodeBuild source configuration*

Step 6: Test and Run the Code

Now, where will you run the code build is called the *environment*. For this one, you have two options.

20. A managed image, selections from the operating system, runtime(s), image, and image version.
21. A custom image, such as Windows, Linux, Linux GPU, or ARM. If you choose another registry, enter the data and tags of the Docker image in Docker Hub as the external registry URL.

If you check the privileged box, because you want to use this building project to create Docker images, the building environment image you selected does not include Docker support from CodeBuild. Otherwise, all builds trying to communicate with the Docker daemon will fail. To interact with your builds, you must also run the Docker daemon. The following build commands can be used to establish the Docker daemon during your build spec install step. Do not perform these instructions using a CodeBuild build environment image with Docker support.

The last section is the service role, an AWS service that takes on a service role when it performs activities on your behalf. As a service that conducts backup operations on your behalf, AWS Backup requires that you pass it a role to assume when performing backup operations. If you have already created one, there is no need to do it again; choose it.

You could do extra configurations such as VPC, timeout, compute options, and variables. See Figure 3-19.

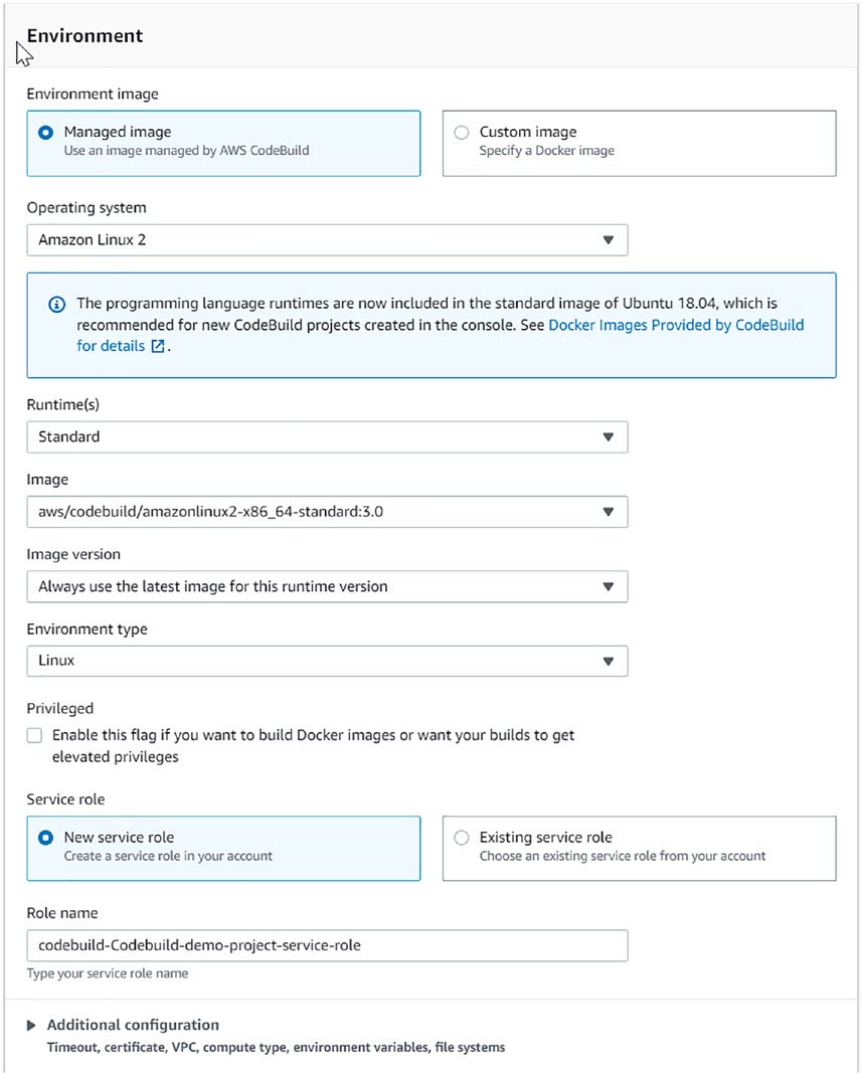

Figure 3-19. *CodeBuild environment configuration screen*

The next step is to specify the build spec file; if you check the repository, you will find a file with that name responsible for telling CodeBuild what to do; there is no need to mention the filename unless you change the default name.

You can insert the command manually, or you can enter the filename if your source code/repository already includes the file with the default name (`buildspec.yml`).

If your build spec file has an alternative name, for example, `buildspec-project-one`, YAML, or location, enter the path to it in the build spec name starting from the root. See Figure 3-20.

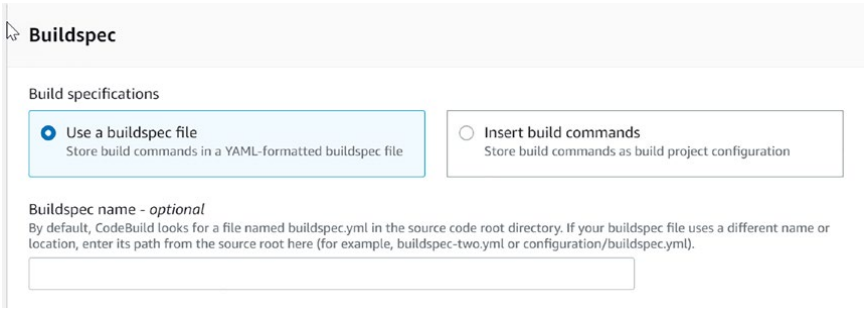

Figure 3-20. *Defining the buildspec file*

A batch configuration is a collection of builds that may be executed as a single operation. When starting the build, the advanced option also allows for batch configuration, and once you choose it, the following options will appear:

- You create a service role or choose the existing one.
- Set the allowed compute type(s) for batch, with optional compute types
- Set the maximum builds allowed in batch.
- Set the batch timeout.

The next option is the artifact, a product created during development. A data structure, a prototype, a flow chart, a design specification, or a configuration script are all examples.

Some artifacts are necessary throughout the development cycle and must be conveniently located.

As you can see from Figure 3-21 and Figure 3-22 we chose S3 as our artifact destination, in the same bucket; you can set a name for the output in case you want it to be compressed.

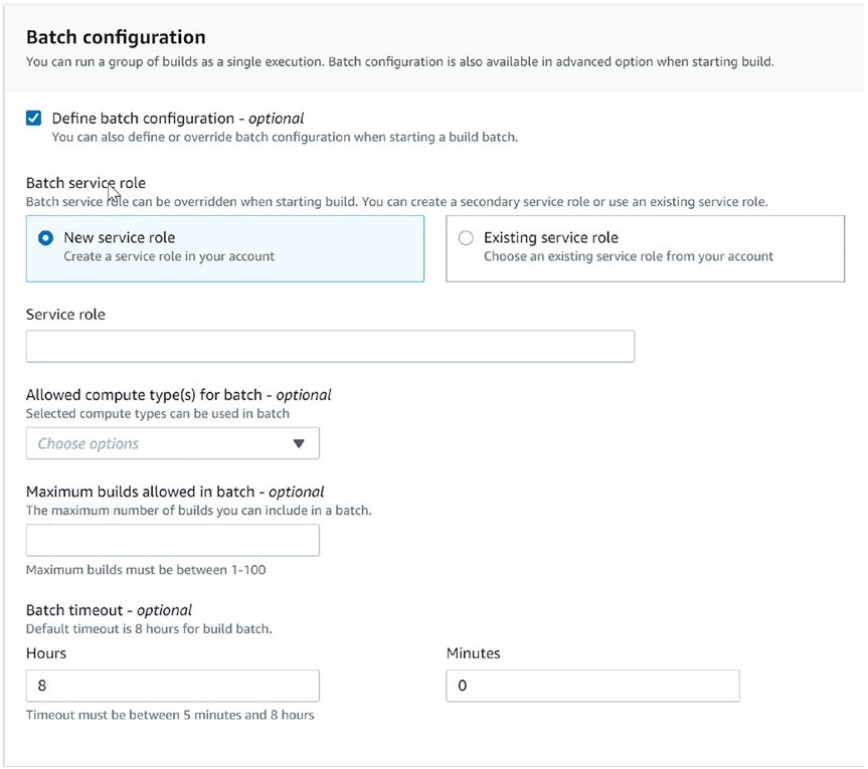

Figure 3-21. *Batch configuration*

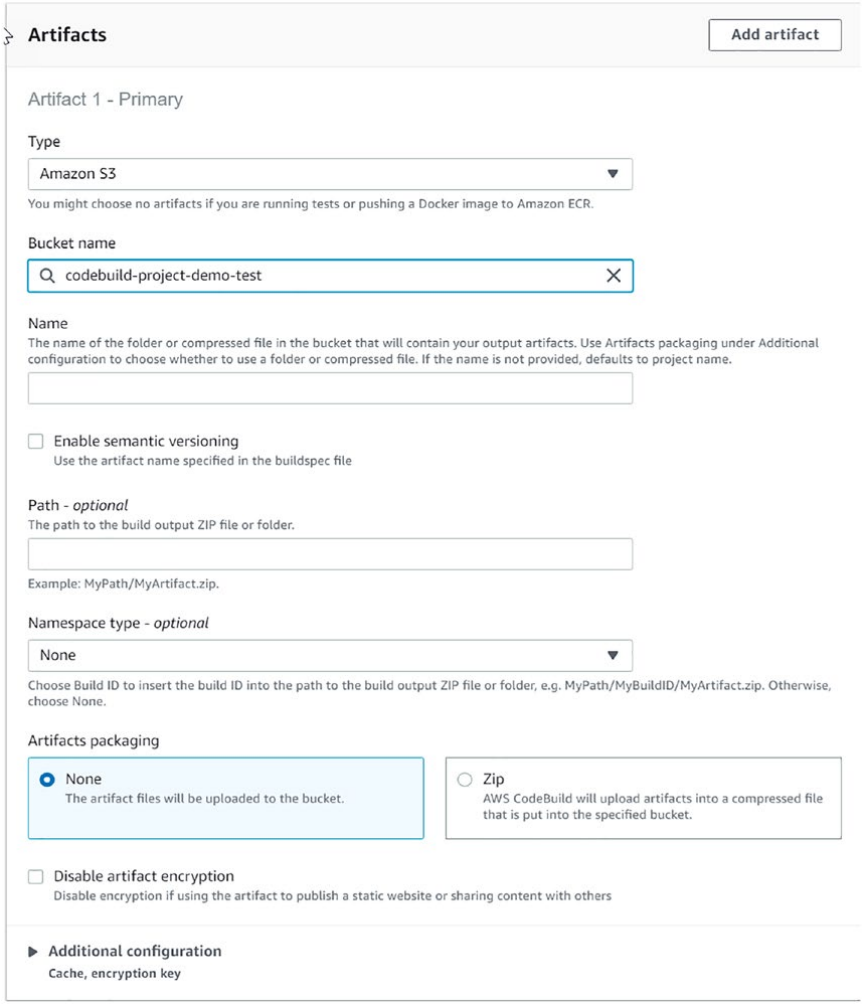

Figure 3-22. *CodeBuild artifact screen configuration*

Finally, I disable the CloudWatch monitor because of the cost, but in case this is production, so you may need to enable it for troubleshooting in the future, Check Figure 3-23.

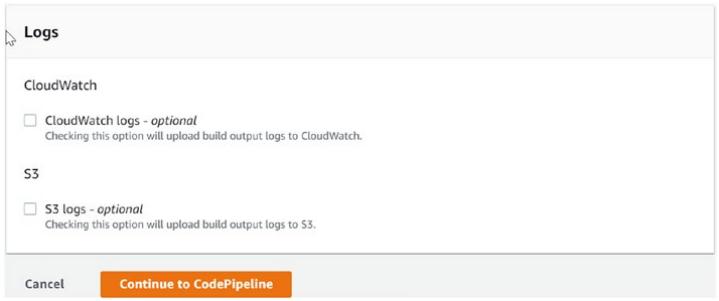

Figure 3-23. *Logs*

Once you are done with creating the project, click Start, check Figure 3-24.

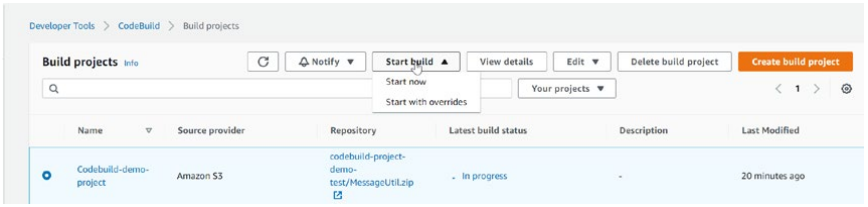

Figure 3-24. *Starting the build*

You can run the following command line to start the build using the command line:

```
1. aws codebuild start-build --project-name project-name
```

How to Create a CodeBuild-Based Pipeline

The following steps will show you how to create a pipeline using CodeBuild for the company code:

1. Access the AWS Portal with the proper user permission to work with CodeBuild:
   a. Working as an Administrator/root account is usually not recommended by AWS.
   b. You might be an Administrator user with specific permission; we will discuss this later in this book.

   c. You might be an IAM User with permission assigned to it and the permission; you need the following (based on AWS best practices):

```
1. codepipeline:*
2. iam:ListRoles
3. iam:PassRole
4. s3:CreateBucket
5. s3:GetBucketPolicy
6. s3:GetObject
7. s3:ListAllMyBuckets
8. s3:ListBucket
9. s3:PutBucketPolicy
10. codecommit:ListBranches
11. codecommit:ListRepositories
12. codedeploy:GetApplication
13. codedeploy:GetDeploymentGroup
14. codedeploy:ListApplications
15. codedeploy:ListDeploymentGroups
16. elasticbeanstalk:DescribeApplications
17. elasticbeanstalk:DescribeEnvironments
18. lambda:GetFunctionConfiguration
19. lambda:ListFunctions
20. opsworks:DescribeStacks
21. opsworks:DescribeApps
22. opsworks:DescribeLayers
```

2. Search for CodePipeline (we will cover it deeply later in this book).
3. Create a pipeline and follow the screens and instructions depending on your setup and configuration; if you are unsure what you are doing, leave things as default, at least for the first time; Check Figure 3-25.

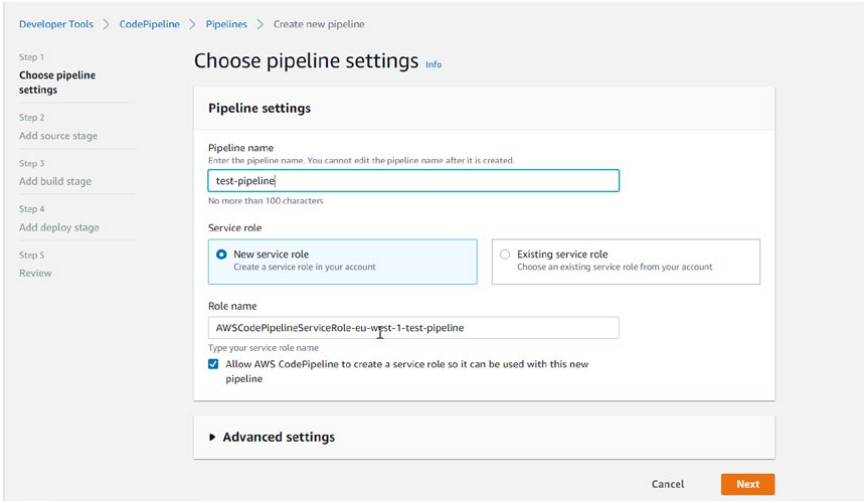

Figure 3-25. *Creating a pipeline*

4. Next, choose the code source; if you save your code on ECR, S3, or CodeCommit, then select it, and once you do this, a new screen will pop out, including the configuration part for the repository; Check Figure 3-26.

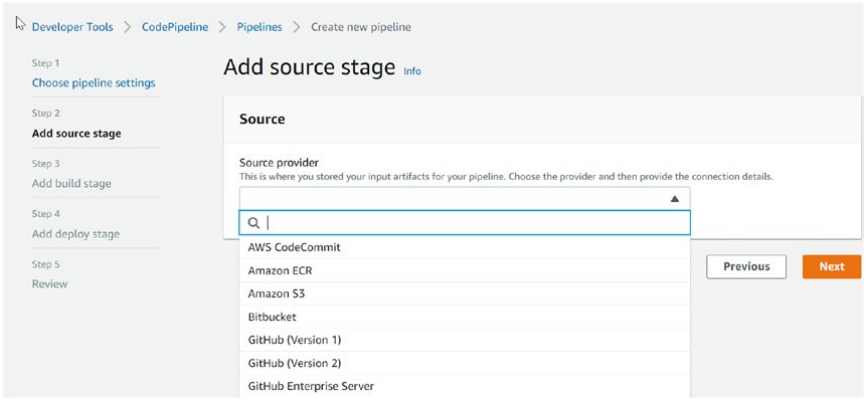

Figure 3-26. *Source stage*

5. Next will be the build stage for the code; by default, AWS provides two options: Jenkins or CodeBuild; as you can see from the screen from Figure 3-27, you need to choose the project name if you already have created one. Otherwise, you need to create one; check Figure 3-28.

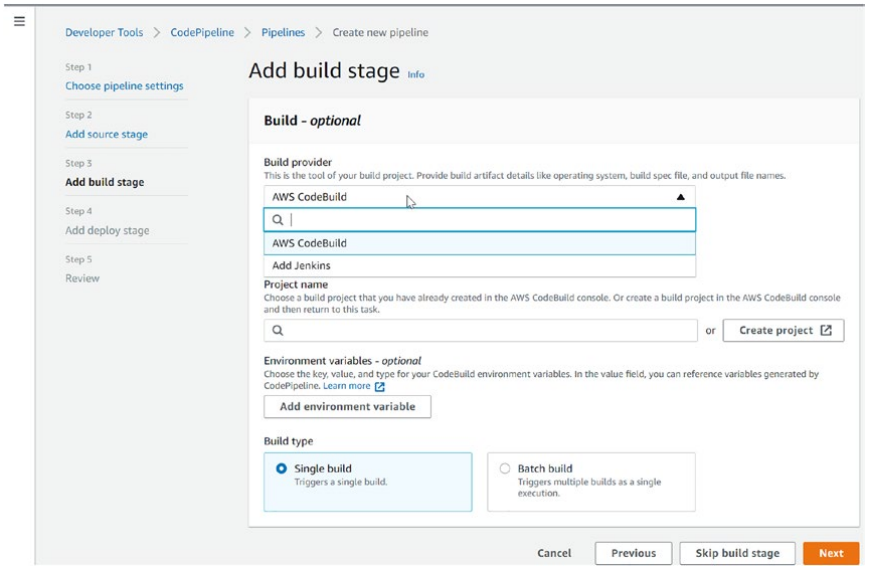

Figure 3-27. *Build stage*

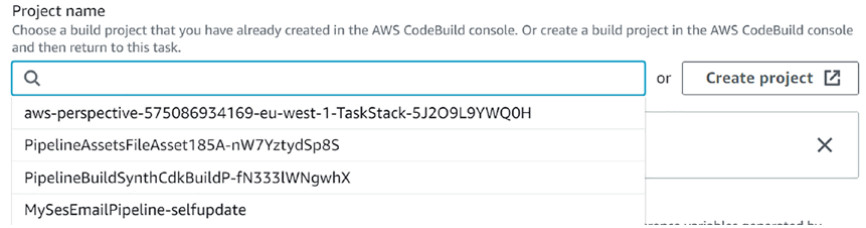

Figure 3-28. *Project name already exists*

Note The "Create project" button will allow you to create a new CodeBuild project that meets your configuration; click it and follow the previous screenshots mentioned earlier in this chapter.

6. Now, you create the project; the last step is to deploy the provider, choose to skip it, and accept this decision when offered if you do not want to deploy the build artifact.

   The second option is to choose a deployment provider for the deploy provider and then enter the parameters when requested if you want to deploy the build artifact.

7. Review your options on the Review screen before clicking "Create pipeline."

8. Once the pipeline runs successfully, open the S3 services; for each pipeline, you are creating by default, create a bucket. The format of this bucket name is as follows:

   ```
   codepipeline-region-ID-random-number
   ```

   Or you can find out the bucket name by retrieving the information for this pipeline using the AWS CLI command; this is the power of the command line.

   ```
   aws codepipeline get-pipeline --name my-pipeline-name
   ```

Navigate to the folder corresponding to your pipeline's name. Then access the folder corresponding to the value you noted before for the Output artifact. You can download and extract the file's content (`filename.zip`).

As mentioned earlier, you can add and test actions to CodeBuild, allowing DevOps to automate and eliminate the manual process.

It's simple to do this. To add a build action or test action, all you have to do is follow these steps:

1. Open CodePipeline Services.
2. Choose the pipeline you want to edit.

3. Choose the tooltip from the details pages and the source action and click Edit.
4. Now, between the source and Build stage, add another stage.
5. Choose the stage name and add an action; enter the action name.
6. For the Action provider, choose CodeBuild.
7. Choose the output artifact you identified previously in this method under "Input artifacts."
8. Enter a name for the output artifact in the "Output artifacts" field.
9. Choose to add an action.
10. Save and release the change.

Use AWS CodeBuild with Jenkins

Not all companies will use CodePipleine; AWS allows you to integrate the services with other solutions such as Jenkins, one of the most common CI/CD tools.

Jenkins is an open-source DevOps automation software designed in Java for CI/CD. It's used to set up CI/CD processes and provides many plugins that allow DevOps to use them for different solutions.

1. To use Jenkins, you should install and configure it; you can set up an EC2 cluster. To do that, follow the official documentation at `https://www.jenkins.io/doc/book/installing/linux/`.
2. Next, install the CodeBuild plugin for Jenkins, which is done by following the official instruction: `https://github.com/awslabs/aws-codebuild-jenkins-plugin`.
3. Next, use the installed plugin, following the instructions mentioned earlier.

Use AWS CodeBuild with Codecov

I have seen another typical case used for CodeBuild, Codecov, a free application for open-source software that helps contributors enhance test coverage and maintain code quality.

And you can sign up for free at `https://about.codecov.io/sign-up/`; check Figure 3-29.

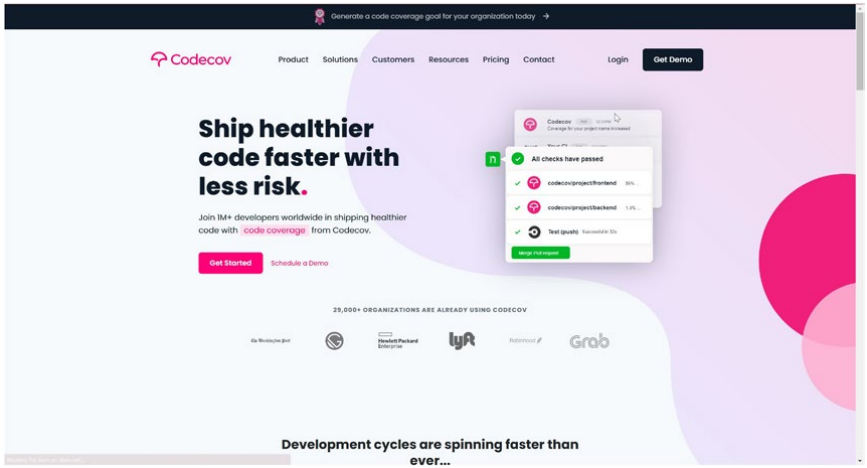

Figure 3-29. *Codecov main page*

Follow these steps:

1. Add the repository to Codecov for which you require coverage.
2. Choose Copy when the token information appears; check the below Figure 3-30.

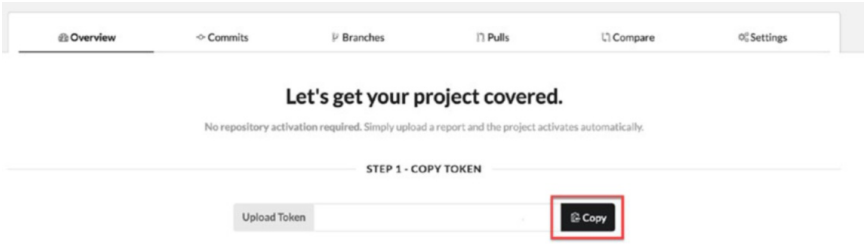

Figure 3-30. *Generate token and copy it*

3. Once you have copied the token, add this token as an environment variable called `CODECOV_TOKEN` to your project.

4. Create a bash script, for example, `codecov_script.sh`, and insert the following lines:

```
1. #/bin/bash
2. bash <(curl -s https://codecov.io/bash) -t $CODECOV_TOKEN
```

5. Create your buildspec file; I usually use Python for DevOps. You can use any programming language you want.

```
1. build:
2.   - pip install coverage
3.   - coverage run -m unittest discover
4. postbuild:
5.   - echo 'CodeCov Connection.'
6.   - bash  codecov_script.sh
```

6. Run the build; it should be working now.

We discussed AWS CodeCommit and CodeBuild as part of continuous integration; now, we will go for the last part, CodeArtifact.

CodeArtifact

CodeArtifact securely saves, distributes, and shares software packages used in the software development process. So that developers have access to the most recent versions, CodeArtifact can be set to download software packages and dependencies from public artifact repositories automatically.

Software development teams increasingly depend on open-source packages to conduct routine operations in their application package. It is increasingly vital for software development teams to retain control over a specific version of open-source software free of vulnerabilities. You can use CodeArtifact to create rules that enforce those mentioned earlier.

In addition, CodeArtifact works with package managers and building tools such as Maven, Gradle, npm, yarn, and pip.

Before working with CodeArtifact, you need to understand the concept of this service so it will be much easier for you to deal with it.

Domain

The first concept you need to understand is the domain. Repositories are grouped into a domain, which is a higher-level object. The domain stores all package assets and information, making managing numerous repositories within a business more straightforward. You may apply permissions to several repositories held by distinct AWS accounts using a domain. Even though an object is accessible from several sources, it is kept only once in a domain.

This is useful for copying, unique names, and storage, and you can apply policy across multiple repositories.

Repository

CodeArtifact repositories are not the same ones we are using in version controls; a CodeArtifact repository has a collection of package versions, each corresponding to a group of assets. Using tools such as the NuGet CLI, the npm CLI, the Maven CLI (mvn), and pip, each repository offers APIs for acquiring and publishing packages. Each domain has a limit of 1,000 repositories.

Package

A package is a collection of software and information for resolving dependencies and installing the program, consisting of a package name and description.

Package Version

A package version allows the DevOps to identify the package version, for example, @types/node 1.2.3.

Package Version Revision

A new package version revision is produced each time a package version is modified.

For example, a recent version revision is created if you release a new source code for Python and want to add another package to the source code by default.

Upstream Repository

When the package versions in one repository can be accessed via the downstream repository's repository endpoint, the contents of the two repositories are essentially merged from the client's perspective. Create an upstream connection between two repositories using CodeArtifact.

Asset

An asset refers to an individual file stored in CodeArtifact and associated with a package version.

Package Namespace

To understand this concept, let me give you a much more straightforward example; CodeArtifact usually organizes packages into logical groups to disallow name conflicts.

For example, if you have an `npm` package, CodeArtifact will create the name like `@types/node`, and `@type` will indicate the package scope and name of the node; this depends on the following documentation: `https://docs.npmjs.com/cli/v7/using-npm/scope`.

For Maven, the approach is different; for example, `org.apache.logging.log4j:log4j` is divided into two things:

- Group ID: `org.apache.logging.log4j`
- CodeArtifact ID: `log4j`

Each package manager has a unique namespace to allow DevOps to understand the type of these packages and their meaning.

Configure CodeArtifact

The first thing you need to do is open the AWS account if it isn't already. Then access CodeArtifact; check Figure 3-31.

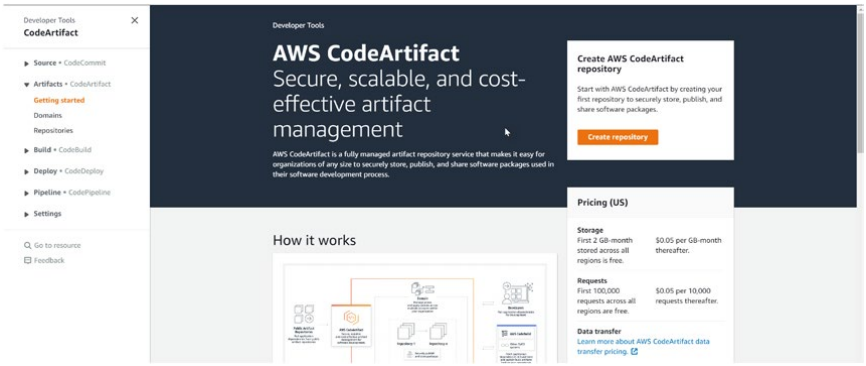

Figure 3-31. *CodeArtifact AWS Portal*

To create your first domain and repository, open the CodeArtifact interface, choose to create a domain and repository, and follow the instructions in the launch wizard.

Also, don't forget to install AWS CLI.

If you want to provide IAM users with access to CodeArtifact, you need to create an IAM policy; you can use the default one provided by AWS, `AWSCodeArtifactAdminAccess`.

The following policy is a custom IAM policy that will allow users to get information about repositories and domains of the CodeArtifact:

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "codeartifact:List*",
            "codeartifact:Describe*",
            "codeartifact:Get*",
            "codeartifact:Read*"
         ],
         "Resource": "*"
      },
      {
         "Effect": "Allow",
         "Action": "sts:GetServiceBearerToken",
         "Resource": "*",
         "Condition": {
            "StringEquals": {
               "sts:AWSServiceName": "codeartifact.amazonaws.com"
            }
         }
      }
   ]
}
```

Another policy example will allow the user to retrieve specific information about the domain you defined in the policy.

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



```
         "Action": "codeartifact:ListDomains",
         "Resource": "arn:aws:codeartifact:eu-west-1:account-
         number:domain-name*"
      }
   ]
}
```

The last step for configuring the CodeArtifact is related to what package you will use.

- *Python*: You can use pip to install the package.
- *Maven*: You should Gradle or mvn.
- *Npm*: You can use the `npm` CLI.
- *NuGet*: You can use `dotnet` or AWS toolkit in Visual Studio Code, as you can see in Figure 3-32.

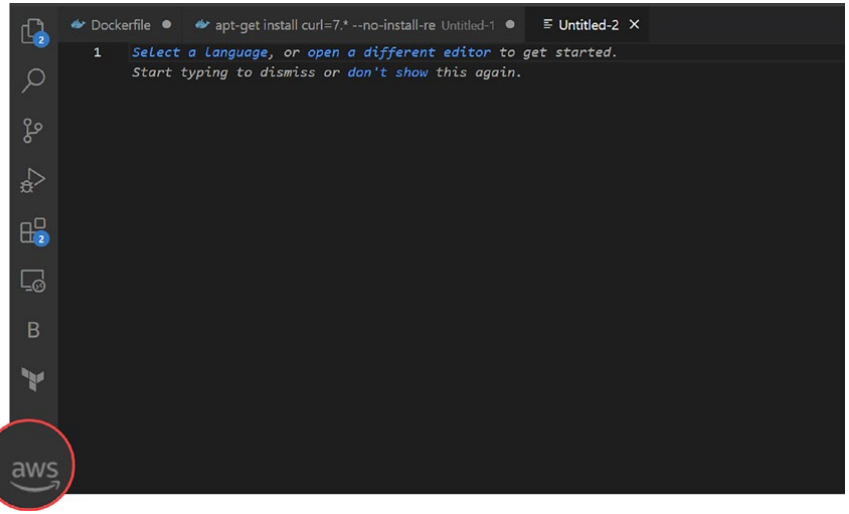

Figure 3-32. *Visual Studio Code, AWS toolkit*

Create a CodeArtifact Domain

Configure a domain by clicking "Create domain," as shown in Figure 3-33.

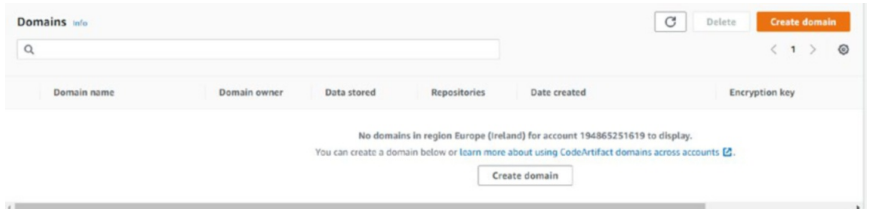

Figure 3-33. *Creating a CodeArtifact domain*

Once you do this, a new screen will show up; check Figure 3-34, and all you have to do is choose a name and KMS; or you can use the command line, as shown here:

```
1. aws codeartifact create-domain --domain my_domain
```


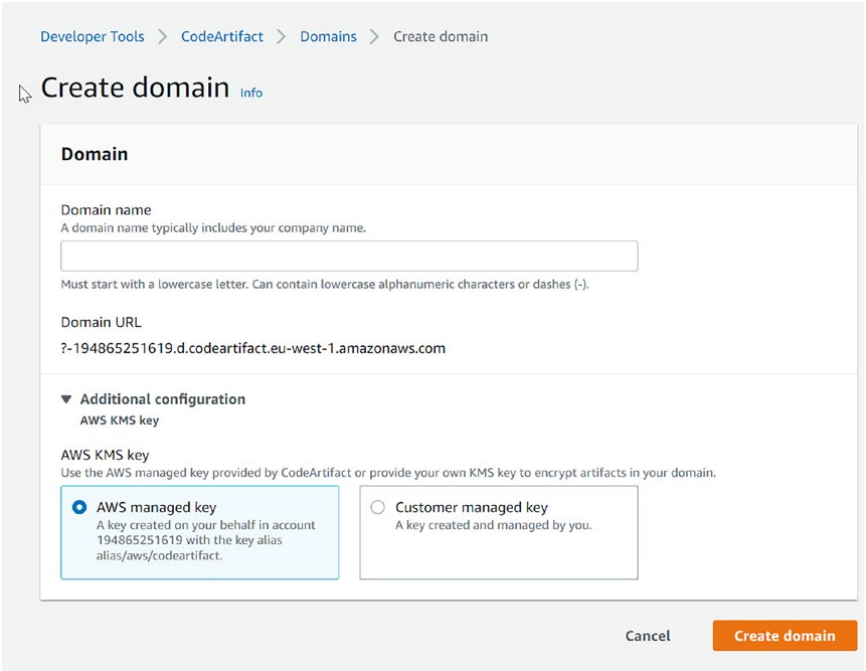

Figure 3-34. *Creating a domain screen*

Create a CodeArtifact Repository

The CodeArtifact console or AWS CLI can be used to establish a repository. There are no packages in a repository when you create it.

In the left panel, choose Repository; check Figure 3-35.

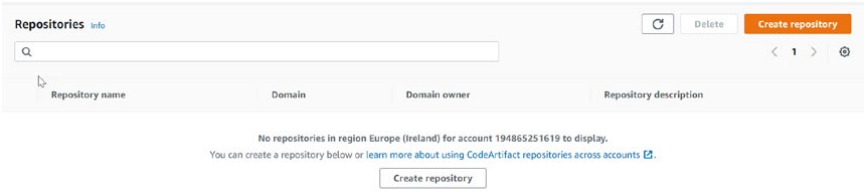

Figure 3-35. *Creating a CodeArtifact repository*

Click "Create repository." Enter your repository's name in the "Repository name" field Enter an optional description for your repository in the "Repository description" field. Publish upstream repositories and add intermediate repositories to link your repositories to package authorities like Maven Central and `npmjs.com`; check Figure 3-36.

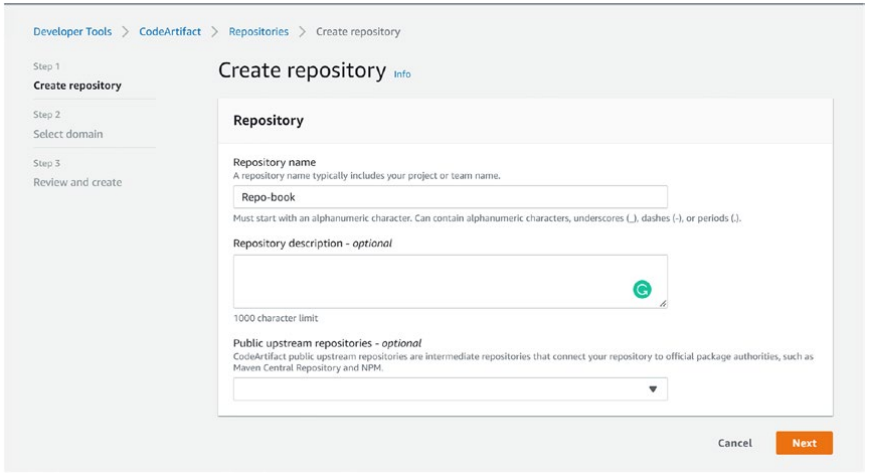

Figure 3-36. *CodeArtifact repository screen*

The domain screen will allow you to choose a domain within the account or a different account, as shown in Figure 3-37. Once you are done, review what CodeArtifact is making for you in the "Review and create" step.

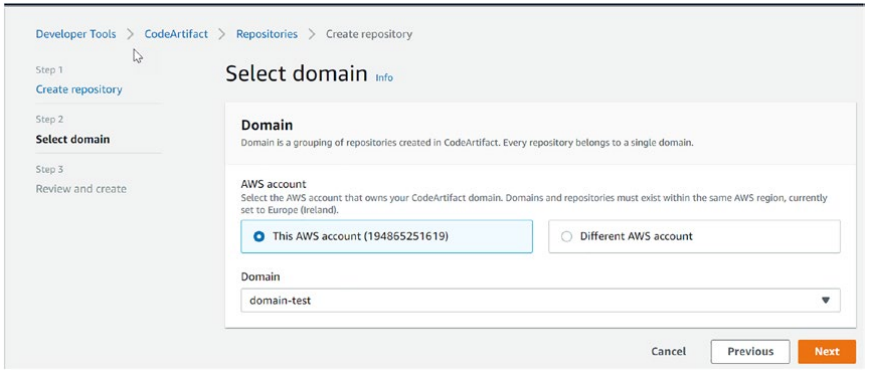

Figure 3-37. *Domain screen*

This can be done also using AWS CLI, as shown here:

```
1. aws codeartifact create-repository --domain book_domain --domain-
owner account-number-for-the-domain --repository book-repo  --description
"Book Repo"
```

Example CodeArtifact for Python

There is no console for this, so you need to use AWS CLI; each supporting package has its own configuration.

```
1. aws codeartifact login --tool pip --domain book_domain --domain-owner
account-number-for-the-domain --repository book-repo
```

The previous example shows how to use Python as a package manager configuration; it's different if you use Maven or `npm`.

Before we end this chapter, you should know each artifactory has something called *dependency caching*, limiting the number of dependencies that must be retrieved from CodeArtifact for each build activating local caching in CodeBuild.

Each package manager has a `Cache` folder; if you use this folder, then you will enable the dependency caching.

Package Manager Tool	Folder
pip	/root/.cache/pip/**/*
mvn	/root/.m2/**/*
Gradle	/root/.gradle/caches/**/*
npm	/root/.npm/**/*
NuGet	/root/.nuget/**/*

For example, you can use dependency caching by mentioning it in the buildspec file, as shown here:

```
1.  cache:
2.   paths:
3.     - '/root/.cache/pip/**/*'
```

Summary

In summary, technology businesses should adopt DevOps ideas and techniques to make the transition to the cloud as easy, efficient, and successful as possible; with this, you will make your life easier. Most IT organizations, if not all, now practice continuous integration.

Applying this solution benefits any company. AWS provides different solutions and services. As mentioned earlier in this chapter, any company can offer a complete automation cycle without a third-party solution.

But at the same, CI/CD offers a simple and efficient method to deploy new app code in minutes. AWS CodePipeline, AWS CodeCommit, AWS CodeBuild, AWS CodeDeploy, and several additional tools may assist developers in integrating, testing, and deploying new code. You can even use these solutions to combine them with infrastructure as code, which will automate the process in the company.

We rarely have the time or talent to plan ahead of time in business, especially for new product development. We can estimate more accurately and confirm more often by completing fewer steps. A shorter feedback loop allows for more iterations. Learning is driven by the number of iterations, not the hours spent.

In the next chapter, I'll cover AWS for continuous deployment concepts and what Amazon AWS can offer as services for continuous deployment. I will also share an example and configuration for these services.

CHAPTER 4

AWS Services for Continuous Deployment

In the previous chapter, I covered CI/CD, focusing on continuous integration, its benefits, and AWS as a cloud provider for constant integration, and I showed some examples.

In this chapter, I will discuss continuous deployment. You already have an idea of what the difference is between CI and CD and when you should use each one.

The isolated modifications integrated and confirmed during CI can be coupled with the remaining product code in a continuous delivery cycle. The pair is then put through a series of more thorough tests. Continuous delivery aims to demonstrate that the ultimate product is deployable, not necessarily to deploy it. The continuous delivery pipeline (CDP) is the method for continuous delivery.

First, however, I'll explain what continuous deployment is, what continuous delivery is, and the difference between the two.

In this chapter, I will cover the continuous deployment and delivery of Amazon Web Services. I'll also walk you through projects that will give you a complete understanding of the different relevant services and how to use them.

Introduction to Continuous Deployment

Continuous deployment is the last step in an automated software development process. Each check-in on a successful build to your source control is delivered to a production-like environment using continuous deployment; you'll deploy the program to production sooner or later. The sooner you do this, the more likely you are to be able to resolve errors quickly. It is simpler to recall what you did yesterday that may have created the issue than to recognize what you did two months ago. Imagine checking some code into source control and receiving error signals from your production environment five minutes later.

O. Mustafa, *A Complete Guide to DevOps with AWS*, https://doi.org/10.1007/978-1-4842-9303-4_4

You'll be able to discover and correct the issue right away, and the production program can be up and running without bugs in five minutes. Unfortunately, the concept of automatic deployment, let alone automated deployment on every check-in, makes most managers and software owners uneasy.

When should you change to continuous deployment?

- *Manual inspections*: There are restrictions to what can be done here if a deployment is not automated. Even if it is automated, an organization may decide not to deploy every version, in which case a human check can be implemented. A user acceptance test (UAT) is typically used to perform such a check early in the pipeline.
- *Deployment in stages*: Even if you want to deploy the newest version from the pipeline automatically, you can do it in stages. You can, for example, publish it to a site that is available only to a subset of users who want to check out the new version, or you can transparently redirect select users to the latest release. If any issues arise during the evaluation period, only a subset of users will be impacted and can be referred to the prior version. The remaining users can be forwarded to the new version if everything seems OK. This is called *canary deployment*.
- *Deployment strategy*: We will have a chance to discuss the deployment strategies in the next chapter; each strategy will need a different configuration and setup.

Continuous Delivery

Continuous delivery and continuous deployment are sometimes confused. Modifications through the pipeline are automatically sent to production, resulting in many daily production deployments. With continuous delivery you can often conduct frequent deployments or choose not to if you want a slower deployment pace. Continuous delivery is required for continuous deployment.

The isolated modifications integrated and confirmed during CI can be coupled with the remaining product code in the continuous delivery cycle. The pair is then put through a series of more thorough tests. Continuous delivery aims to demonstrate that the ultimate product is deployable, not necessarily to deploy it. The continuous delivery pipeline (CDP) is the method for continuous delivery.

With continuous delivery, such as DevOps, you have the capability to change the type of deployment you use, and you can deploy new features, bugs, fixes, and more to production or any other environment.

When continuous delivery is implemented, the following occurs:

- Software is deployed through a cycle.
- Your team emphasizes software deployment above new feature development.
- Anyone can obtain instant, automatic feedback on their system's production readiness whenever they make a change.
- On-demand deployments of any version of the program to any environment are possible.

The following are the main advantages of continuous delivery:

- *Reduced deployment risk*: Because you're making more minor changes, less can go wrong, and correcting errors is quicker.
- *Noticeable progress*: Many people keep track of their success by logging their hours worked. It is considerably less credible if "done" implies "developers announce it to be done" than if it is deployed into a production environment and works.
- *Feedback*: The greatest danger of every software project is that you will create something useless. The sooner and more often you put functional software in front of actual users, the faster you will learn how beneficial it is.

Developers can use continuous delivery to automate testing beyond unit tests, allowing them to evaluate application improvements across several dimensions before releasing them to users. UI testing, load testing, integration testing, API reliability testing, and other tests allow developers to test upgrades and identify concerns ahead of time extensively. Automating the creation and replication of many environments for testing, which was hard to achieve on-premises, is now simple and cost-effective in the cloud.

Amazon AWS provides two services for continuous delivery: AWS CodeDeploy and AWS CodePipeline.

AWS CodeDeploy

AWS CodeDeploy is a fully managed service tool that automates program deployments to Amazon Elastic Compute Cloud (Amazon EC2), AWS Fargate/ECS, AWS Lambda, and your on-premises servers.

CodeDeploy may deploy application content stored in Amazon S3 buckets, GitHub repositories, or Bitbucket repositories that run on a server. A serverless Lambda function can also be deployed using CodeDeploy. CodeDeploy can be used without making any modifications to your current code.

You can use CodeDeploy to make it simpler to do the following:

- Release new features on time.
- Update versions of AWS Lambda functions.
- Avoid any downtime during application deployment.
- Take on the complexity of upgrading your apps while avoiding many of the hazards of manual deployments.

The service grows with your infrastructure, allowing you to quickly deploy one or hundreds of instances. You can use almost any application material, including code, serverless AWS Lambda functions, web and configuration files, executables, packages, scripts, and multimedia files.

CodeDeploy offers various advantages that support the DevOps idea of continuous deployment, starting with deployments being automated using CodeDeploy, enabling you to deploy software reliably and quickly. CodeDeploy gives you centralized management over your application deployments.

CodeDeploy allows you to effortlessly create and check the progress of your deployments using the AWS Management Console or the AWS CLI; you can see when and where each application revision was deployed using CodeDeploy's full report. You can also set up push alerts to obtain real-time deployment changes.

On the other hand, to reduce downtime, CodeDeploy assists you in maximizing the availability of your application throughout the software deployment. It gradually introduces updates and monitors application health using customizable rules. Software deployments can be halted and turned back if there are any issues.

You can halt and roll back deployments automatically or manually if there are issues.

In addition, CodeDeploy works with multiple deployment strategies, discussed in the next chapter.

There is a different use case for CodeDeploy. However, I will show how to use a couple of examples so you can understand when you need to use it.

AWS CodeDeploy Project

Imagine development teams frequently asking you to deploy a web page on an EC2 instance repeatedly. This is a repetitive, tedious manual task. To automate this process, we will use CodeDeploy; the infrastructure is pretty simple and will be as follows:

- The application is based on WordPress, which requires PHP and MySQL.
- The operating system is Amazon Linux or Red Hat Enterprise Linux.
- There is a single EC2 instance.

The high-level steps for this project are as follows:

1. Configure an EC2 instance and install the CodeDeploy Agent.
2. Configure what needs to be deployed to the EC2 instance.
3. Upload the WordPress code to S3.
4. Update the WordPress applications depending on the needs.

Step 1: Configure an EC2 Instance

Sign in to the AWS Console, choose EC2, and click "Launch instances" (Figure 4-1).

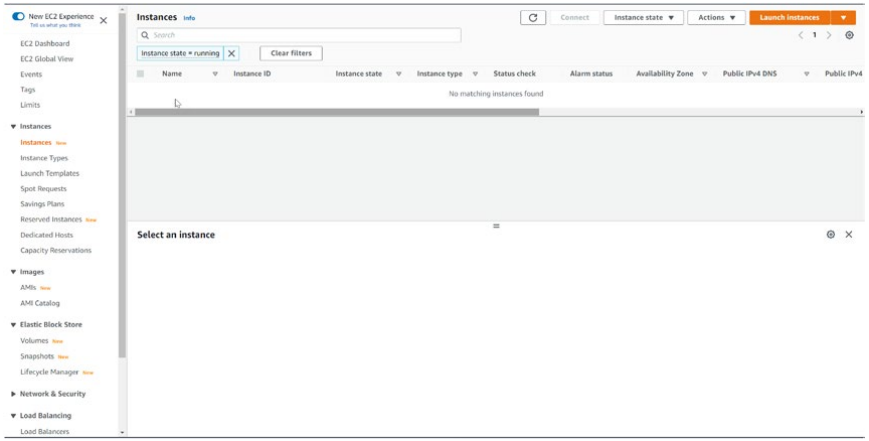

Figure 4-1. *Creating an EC2 instance*

Next, you need to fill in some information, such as the name, network, and key pair (SSH); what you need depends on your configuration, as shown in Figure 4-2.

- Choose an Amazon Machine Image (AMI), and choose an OS that is supported by CodeDeploy; you can check here for supported versions: `https://docs.aws.amazon.com/codedeploy/latest/userguide/codedeploy-agent.html#codedeploy-agent-supported-operating-systems`.
- Choose an instance type.

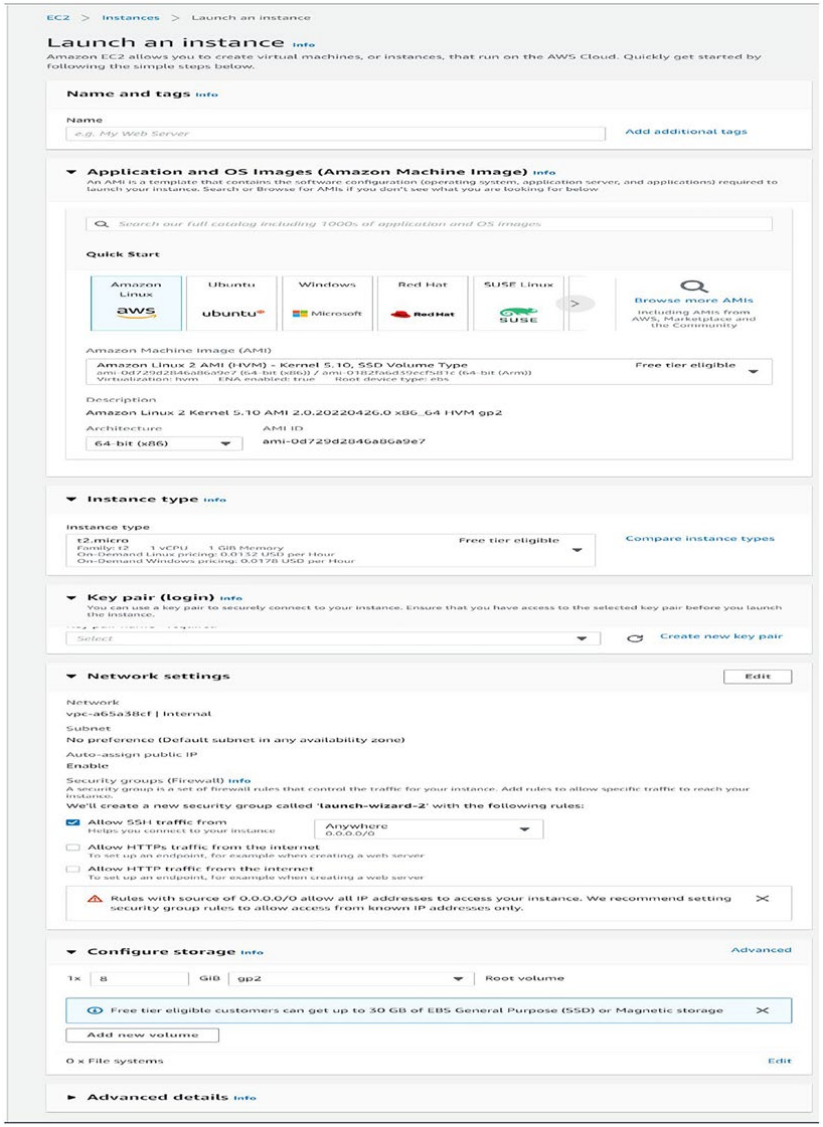

Figure 4-2. *EC2 configuration*

You can create an IAM role depending on the needed permission. To make an IAM role inside AWS, please follow the steps; when dealing with CodeDeploy and CodePipeline, you must provide permission to allow these services access to the required resources.

Caution You may notice that some steps are repeatable for CodePipeline and CodeDeploy. Nevertheless, the idea is to show how easy the configuration is for both.

CREATE AWS ROLES

If you are familiar with these steps, skip this section.

1. From the console, choose IAM, or you can access it at `https://console.aws.amazon.com/iam`.
2. The IAM screen will open; choose a role. See Figure 4-3.

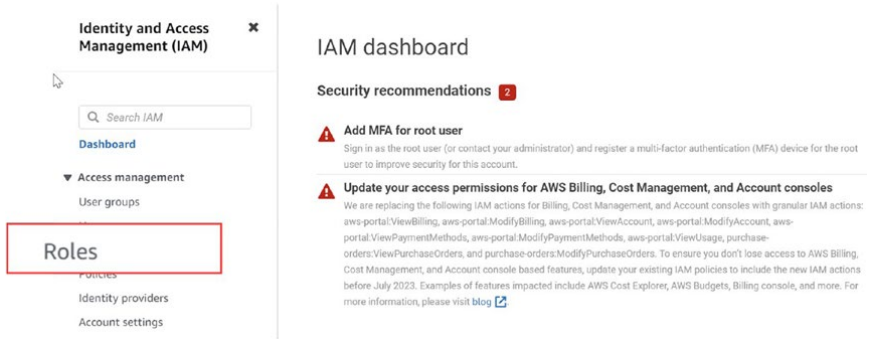

Figure 4-3. *Creating an IAM role*

3. Choose Roles from the left menu, and then select "Create role." On the "Create role" page, choose the services you will be using for the role, which is CodeDeploy here, as shown in Figure 4-4.

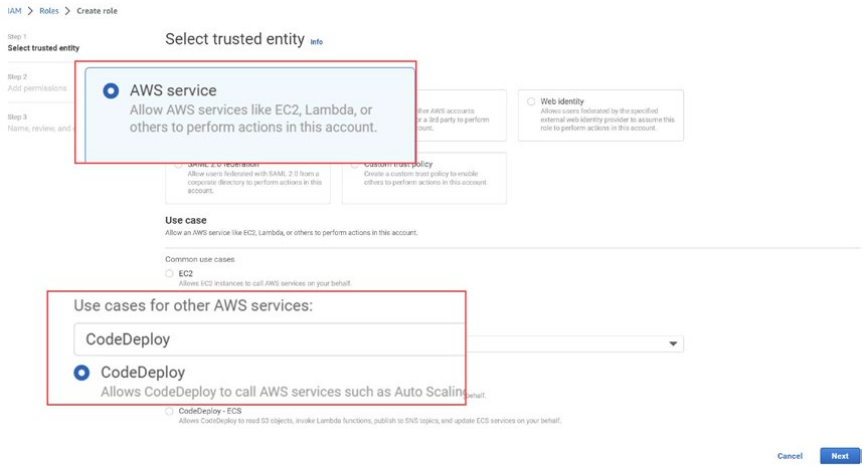

Figure 4-4. *Choosing CodeDeploy*

4. The next screen will be for adding permission; leave everything as is, as shown in Figure 4-5.

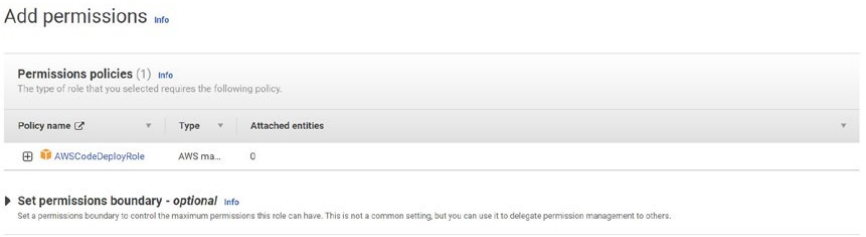

Figure 4-5. *Adding permission to a role*

5. Type the service role's name (**CodeDeployServiceRole**, for instance) into the "Role name field" on the Review screen, and then click "Create role," as shown in Figure 4-6.

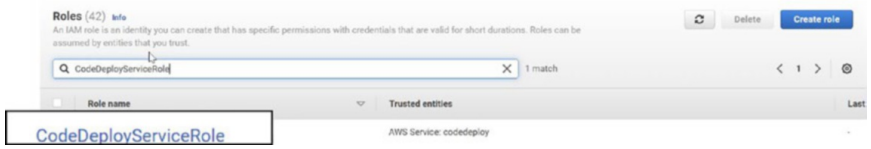

Figure 4-6. *After creating the role*

6. Now, click the role, which will open a new screen that allows you to edit the role and insists on a trust relationship; see Figure 4-7.

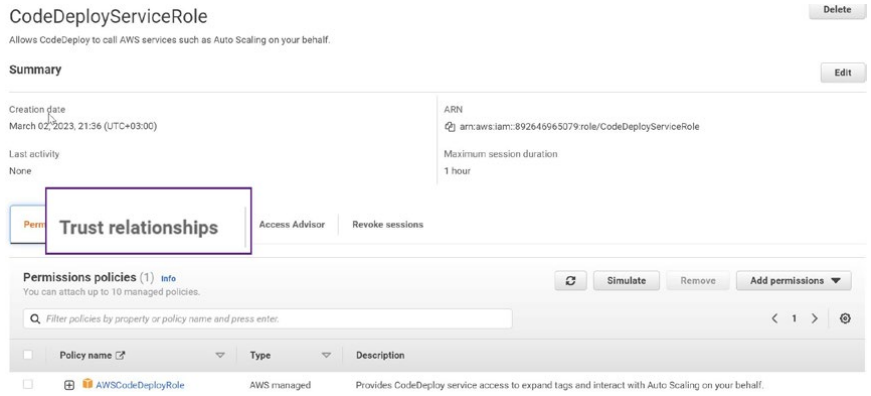

Figure 4-7. *Editing the role-trust relationships*

You will see the current policy attached to the role; you can add a different one as shown here, which allows the code deployment to access an additional AWS resource.

For example, if I need to give permission for CodeDeploy for EC2, I use this:

```
1. {
2.     "Version": "2012-10-17",
3.     "Statement": [
```

```
        {
            "Sid":" ",
            "Effect": "Allow",
            "Principal": {
                "Service": "ec2.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}

```

After that, you need to grant access to S3 so CodeDeploy can access the bucket and deploy the code to EC2.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:Get*",
                "s3:List*"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}

```

Note that for the next configuration for EC2 Storage, I recommend leaving it as is to ensure no extra cost.

- Using tags will make your life easier later.
- Configure the security group as shown in Figure 4-8.
    - For now, open ports SSH, HTTP, and HTTPS.

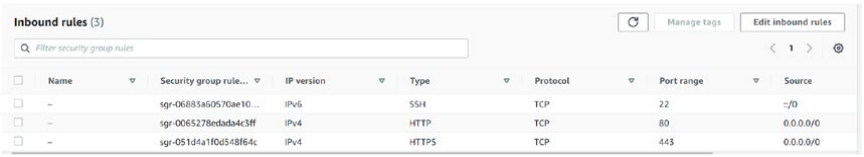

Figure 4-8. *Security group*

Continue to the "Instance launch" page and click Launch.

Do not forget to keep the private key in a secret place.

After launching the instance using the private key or SSM, we need to install the CodeDeploy Agent; if you choose to install it and follow the steps here, the installation for the agent will be different from one operating system to another.

The link in command 5 in the following snippet is a generic link, so make sure to change it depending on where the resource will be deployed and the bucket's name.

```
1.  sudo yum update
2. sudo yum install ruby
3. sudo yum install wget
4. cd/home/ec2-user
5. wget https://bucket-name.s3.region-identifier.amazonaws.com/
   latest/install
6. chmod +x./install
7. sudo./install auto
8. sudo service codedeploy-agent status
9. sudo service codedeploy-agent start
10. sudo service codedeploy-agent status
```

The name of the Amazon S3 bucket containing your area's CodeDeploy Resource Kit files is `bucket-name`. `region-identifier` is the identifier for your region, such as West Europe Region.

For example, `aws-codedeploy-eu-west-2` is the bucket name, and the region is `eu-west-2`. To find the bucket name and the available region to download the agent, please refer to the AWS documentation shown here:

https://docs.aws.amazon.com/codedeploy/latest/userguide/resource-kit.html#resource-kit-bucket-names

Step #2: Configure the Source Content to Be Deployed on EC2

To clone your WordPress code, you need to use the `git` command.

To install the `git` command, follow these steps. Each Linux command has its package manager, but the concept is the same (I am using Ubuntu).

Git packages are available using the `apt` command, commonly used in Ubuntu.

```
1. sudo apt-get update
2. sudo apt-get install git-all
```

Now, with the `git` command, you can clone the repo; you can choose to clone to any folder.

```
git clone https://github.com/WordPress/WordPress.git /home/ec2-user
```

After cloning the code and moving to the next step, you need to stop and start scripting for the application, which will be used later in the appspec YAML, the same as the buildspec discussed previously in other chapters.

1. The first script you will need is for the dependencies. This script will be responsible for installing Apache, MySQL, and PHP; you can call the script `WordPress-dependencies.sh` to install dependencies. Save all the scripts in one folder.

   ```
   1. #!/bin/bash
   2. sudo amazon-linux-extras install php7.4
   3. sudo yum install -y httpd mariadb-server php
   ```

2. The following script, which is straightforward, starts the application: `start-script.sh`.

   ```
   1. #!/bin/bash
   2. systemctl start mariadb.service
   ```



   ```
   3. systemctl start httpd.service
   4. systemctl start php-fpm.service
   ```

3. Shut down the application with `stop-script.sh`.

   ```
   1. #!/bin/bash
   2. systemctl stop httpd.service
   3. systemctl stop mariadb.service
   4. systemctl stop php-fpm.service
   ```

4. Create a WordPress database script called `create-database.sh`. This script will be used to create a database called `Wordpress_DB` to be used by the application; we notice that you also need to create a folder called `scripts` and create this file under it.

   MariaDB on Amazon Linux 2 does not have a root password by default. If you create a root password for MariaDB and lock yourself out of your database, you must reset the root password.

   And you can do that by running the following command; if you do not want to set the MariaDB password, you can do that and leave it blank.

   ```
   ALTER USER 'root'@'localhost' IDENTIFIED BY 'new_password';
   1. #!/bin/bash
   2. mysql -u root -p your-password
   3. CREATE DATABASE IF NOT EXISTS test;
   4. CREATE_WordPress_DB
   ```

5. The last step is that you need to change the script's permission to make them executable.

   ```
   1. chmod +x//home/ec2-user/scripts/*
   ```

6. If you use CodeDeploy, CodePipeline, etc., you need to write either the buildspec or AppSpec, depending on which services you will use; this time, I will use CodeDeploy.

Using the appsec file will be our choice because we are deploying to EC2; before continuing with the example, let's look at what the difference is between then.

buildspec.yml

The pipeline generates an artifact at the source, and this file is used to build it. Remember, only applications that need a build (such as Angular, React, etc.) will need this. It is unnecessary to download this file using a Node.js application.

appspec.yml

If you want to deploy your software to an EC2 instance, you will need this file. When the files are replaced in the EC2 instance, the deployment group will seek this file in your root, which contains instructions. In the case of a Node.js application, for instance, you would need to issue the command `run application` again.

```
version: 0.0
os: Linux
files:
  - source:/⩛
    destination:/var/www/html/WordPress
hooks:
  Before Install:
    - location: scripts/WordPress-dependencies.sh
      timeout: 300
      runas: root
  ApplicationStart:
    - location: scripts/start_script.sh
    - location: scripts/create_database.sh
      timeout: 300
      runas: root
  ApplicationStop:
    - location: scripts/stop_script.sh
      timeout: 300
      runas: root

```


Step 3: Upload WordPress Code to S3

Now you'll prepare your source code and submit it to a place where CodeDeploy can deploy it; the following steps demonstrate how to create an Amazon S3 bucket, prepare the application revision files for the bucket, bundle the revision files, and finally push the modification to the bucket.

1. Creating the s3 bucket is a straightforward step either by using the command line (AWS CLI) or by using console.

```
1. aws s3 mb s3://test-wordpress-code
```

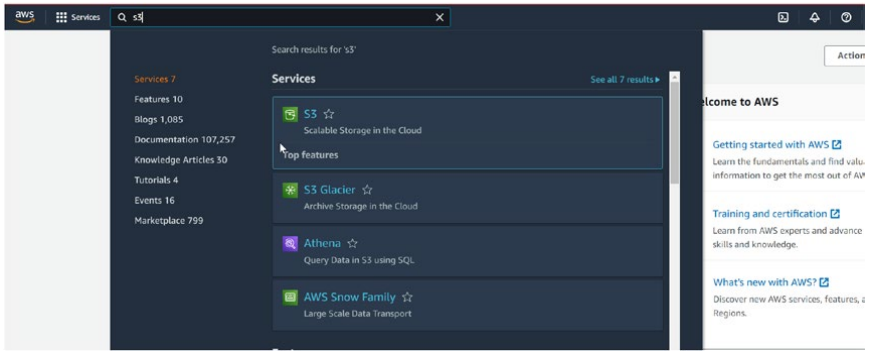

Figure 4-9. *Creating the S3 bucket from the console*

2. Choose S3, as shown in Figure 4-9, and the new screen will display. Click "Create bucket," as shown in Figure 4-10.

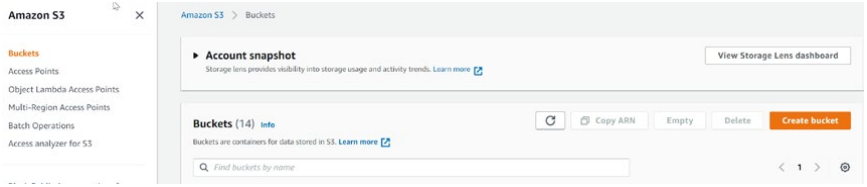

Figure 4-10. *S3 Dashboard*

3. The configuration screen will open; choose the same bucket name and the region as your CodeDeploy region, and leave the other settings as the defaults. See Figure 4-11.

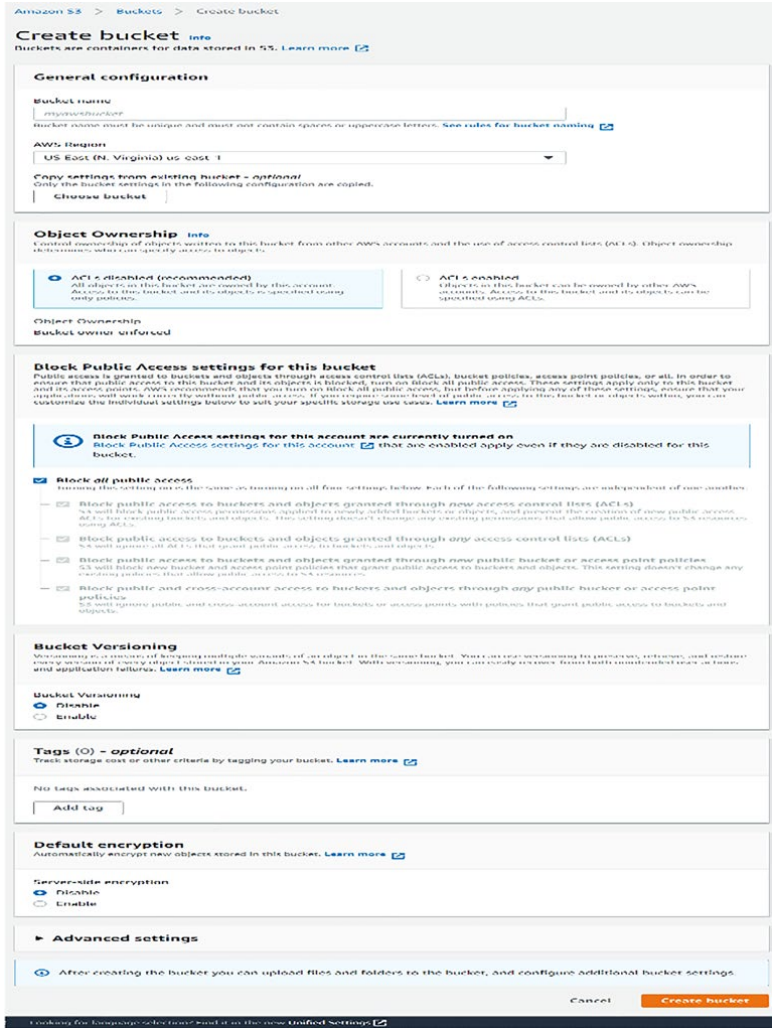

Figure 4-11. *S3 configuration screen*

4. Create an archive file with the WordPress application files and the AppSpec file, and we will register the app called WordPress. The last command will call the push command to create a bundle for all the files.

   ```
   cd/home/ec2-user
   ```

   If you had not created the role mentioned in step 1 of this example, you would face a "permission denied" error when running this command.

   ```
   1. aws deploy create-application --application-name
   WordPress_App
   ```

   ```
   1. aws deploy push --application-name WordPress_App
   --s3-location s3://test-wordpress-code/WordPressApp.zip
   --ignore-hidden-files
   ```

   The previous command compiles the files in the current directory into a single file. `WordPressApp.zip` is a single archive file that contains everything you need.

Step 4: Deploy the Code

The next step is now deploying an application. You can use the AWS CLI or the CodeDeploy interface to deploy the version and track its progress.

If you want to deploy the application using AWS CLI, follow the instructions in this section.

First, ensure you have the proper permission to deploy the app; otherwise, you will not be able to do that; you should create a service role needed for the next step.

The following is one of the examples for the Services role (saved under a file called `accessrole.json`), which will give access to CodeDeploy under all supported regions:

```
1. {
2.     "Version": "2012-10-17",
3.     "Statement": [
4.         {
5.             "Sid": "",
6.             "Effect": "Allow",
```

```
            "Principal": {
                "Service": [
                    "codedeploy.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}

```

Then, run the following command, which will create a service role:

```
aws iam create-role --role-name CodeDeployServiceRole --assume-role-policy-document file://accessrole.json
```

We need to create a deployment group connected to the previous service role. It is possible to delegate authority to a service by assigning it a special identity and access management (IAM) role. CodeDeploy can connect to your Amazon EC2 instances thanks to the service role.

```
aws deploy create-deployment -group
  --application-name WordPress_App
 --deployment group name WordPress_DepGroup
--deployment-config-name CodeDeployDefault. OneAtATime
 --ec2-tag-filters Key=Name,Value=CodeDeployDemo,Type=KEY_AND_VALUE
 --service-role-arn serviceRoleARN
```

After applying the earlier steps, refer to Figure 4-12, which explains the application deployment step.

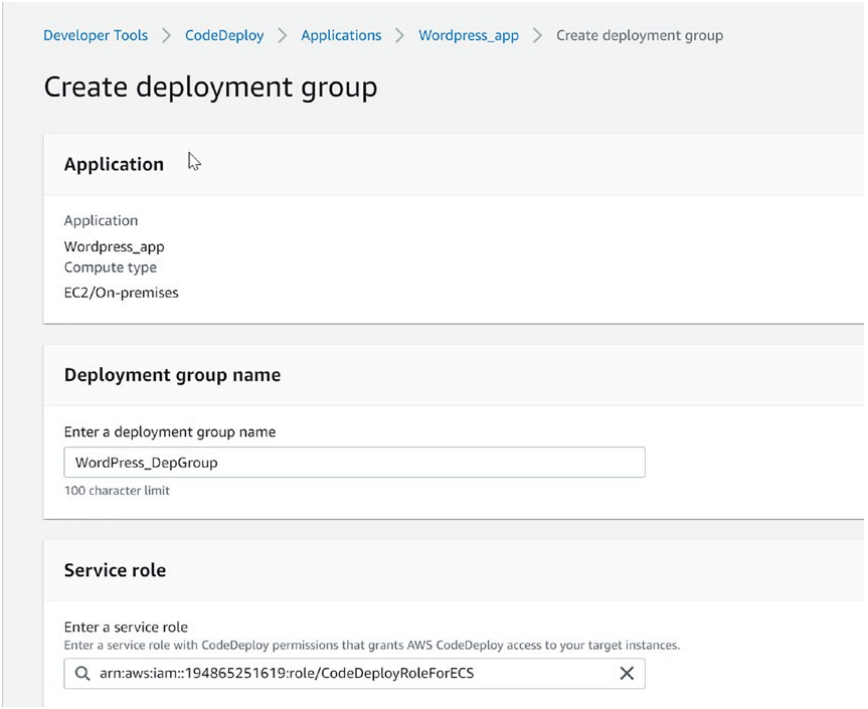

Figure 4-12. *Creating a deployment group from the console*

Before creating a deployment, ensure the CodeDeploy Agent is installed and running; otherwise, an issue will appear.

```
aws ssm create-association
  --name AWS-ConfigureAWSPackage
  --targets Key=tag:Name,Values=CodeDeployDemo
  --parameters action=Install, name=AWSCodeDeployAgent
  --schedule-expression "cron(0 2? * SUN *)"
```

On the same screen of CodeDeploy, you will see the system manager section, which is responsible for deploying the CodeDeploy Agent via System Manager; see Figure 4-13.

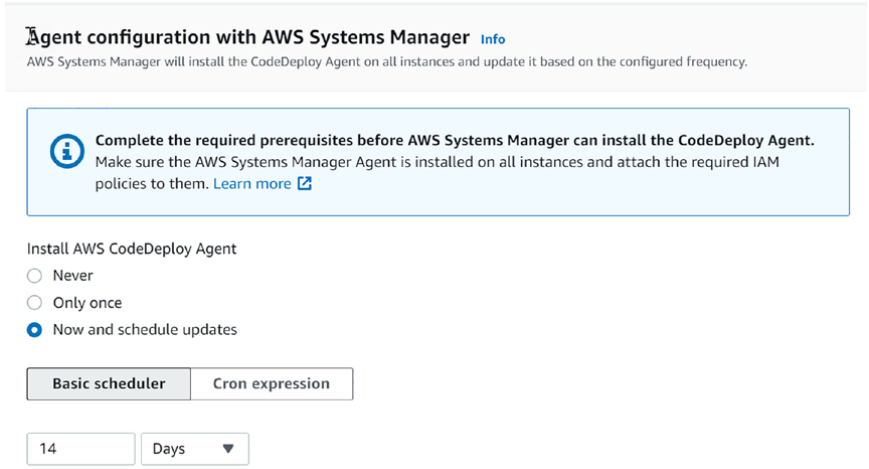

Figure 4-13. *Installing the agent through the console*

Let's deploy now by running the following command:

```
1. aws deploy create-deployment
2.  --application-name WordPress_App
3.  --deployment-config-name CodeDeployDefault. OneAtATime
4. --deployment-group-name WordPress_DepGroup
5.  --s3-location bucket=codedeploydemobucket, bundleType=zip,
    key=WordPressApp.zip
6.
```

The following section concerns deployment settings and how fast the application will be deployed; see Figure 4-14.

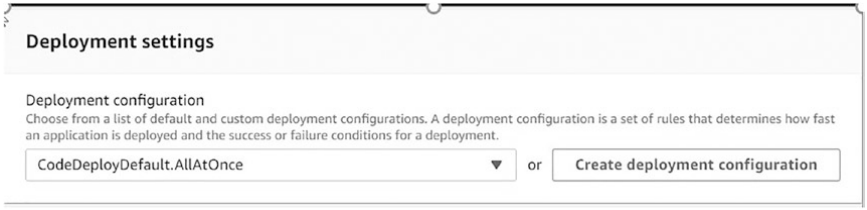

Figure 4-14. *Deployment settings*

Step 5 (Optional): Redeploy the Code

The question is, what will happen if you change something inside the code? How can you redeploy again; the answer is simple, as Amazon AWS CodeDeploy allows you to redeploy again using AWS CLI or the console.

Using the same command as earlier, you can redeploy your code without issues.

```
1. aws deploy create-deployment
2.  --application-name WordPress_App \
3.  --deployment-config-name CodeDeployDefault. OneAtATime
4. --deployment-group-name WordPress_DepGroup
5.  --s3-location bucket=codedeploydemobucket, bundleType=zip,
    key=WordPressApp.zip
6.
```

Or, in the console, select your application and click "Deploy application," as shown in Figure 4-15.

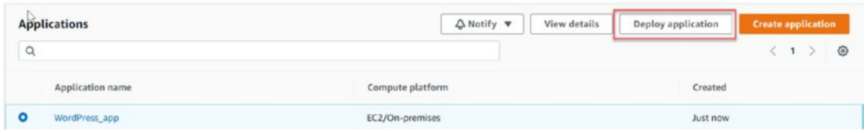

Figure 4-15. *Deploying the WordPress application*

Overall, once the application is deployed and you try to access using the public DNS of the EC2 instance, the first page will be like that; this page will allow you to configure the connection to the database and enter a username and password.

Since the next page is HTTP, you need to open the port under the security group to make it work; otherwise, it will not be open.

In Figure 4-16, you can see the application home page (configuration page).

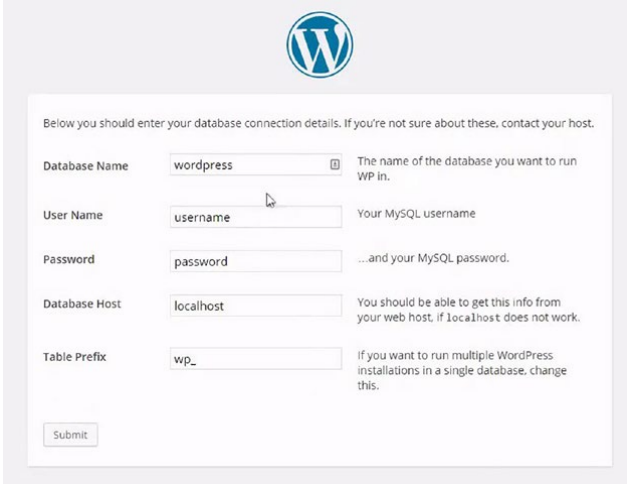

Figure 4-16. *The WordPress configuration page*

You can also use CodeDeploy for on-premises servers; for sure, you will need a VPN in that case, and it can be used to automate Lambda functions; at the end of this chapter, I will include one of the projects for CodeDeploy.

Before starting with the following topic, I will explain the difference between CodeDeploy and CodePipeline, as you can see from Table 4-1, which shows a difference between CodeDeploy and CodePipeline and will allow you to understand each one of them.

Table 4-1. *CodeDeploy vs. CodePipeline Key Differences*

Factors	CodeDeploy	CodePipeline
Usage	Use to automatically deploy code to EC2 instances, Lambda, or on-premises servers.	This is one of the continuous deployments that allow you to build, test, and deploy once your code changes.
Known name	Deployment as services.	Continuous deployment.
Features	Helps to minimize downtime during deployment. Can work with any complexity of code update. Track application health status. Accessible to roll back updates across EC2 instances.	Provide different prebuilt plugins that allow you to integrate with other tools. Integration with third-party solutions. Can follow up with the deployment via workflow model.

AWS CodePipline

The previous table gave you an idea about what CodePipeline provides. This service allows you to automate the manual step to release your software so you can quickly configure stages for the automation software release process. But what else can you do with CodePipline, or what is the use case for this service?

First, you can automate your software releases. As a company, you want to minimize the potential for human error; automating your software releases does that.

Uploading your code source to version control allows you to test, build, or deploy that code. The workflow differs from company to company. Some companies will make the approval manual to ensure the code meets the standard; others integrate the workflow with other QA automation tools, and so on.

One of the most critical use cases is speeding up the delivery time and making the development team focus on the code quality more than the deployment time.

After the development cycle, CodePipline allows you to check the deployment status using the console and check which stage has been implemented thus far (see Figure 4-17).

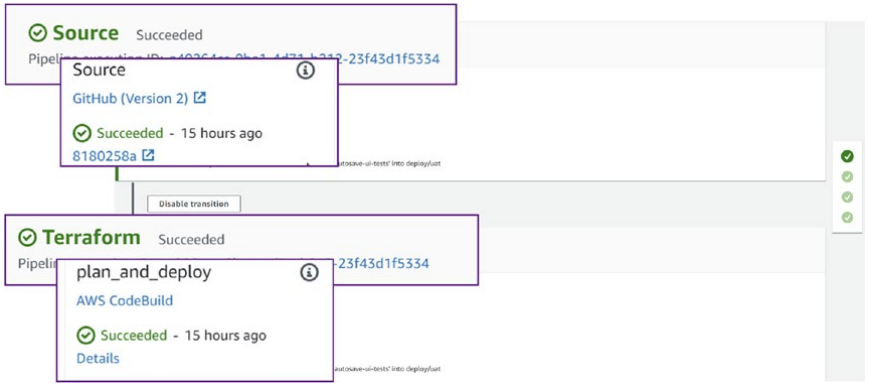

Figure 4-17. *Checking the deployment*

Integration with third-party software is another consideration. For example, the company can use GitHub, Gitlab, or Bitbucket to upload the code to CodeCommit and trigger action on Jenkins. Again, the configuration could differ from one company to another; it all depends on the use case.

One more critical feature of CodePipeline is that it allows you to check the deployment history, see whether it has failed or succeeded, and review what has been deployed to the server and what changes the development team made. As shown in Figure 4-18, the history has three statuses.

- Succeeded means the pipeline has been deployed to the target without any issues.
- Failed means one or more stages has failed for some reason.
- Superseded means the user should stop this pipeline or eliminate one of the stages.

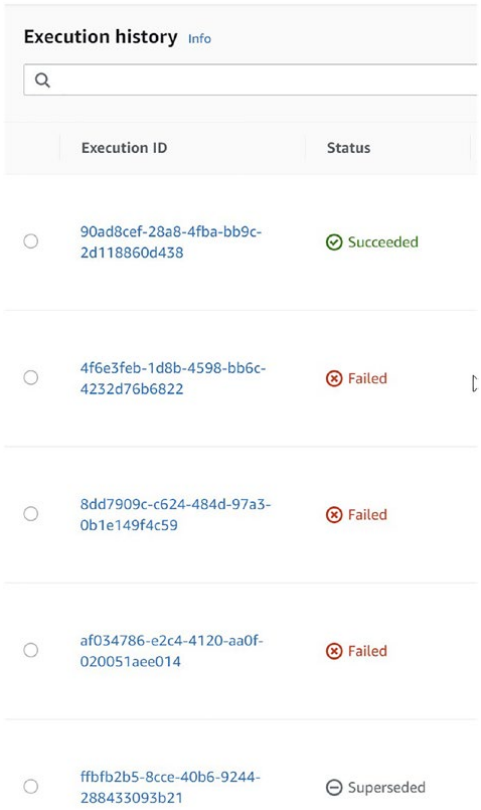

Figure 4-18. *CodePipeline history*

CodeDeploy Component

When talking about CodeDeploy, a different component, starting with the application, is considered the first part of the CodeDeploy component, a collection of deployment groups such as EC2 instances or on-premises servers; moreover, a deployment configuration is how your deployment will work.

Another component mentioned earlier, the buildspec**,** is the most important part; it is responsible for how the application will be deployed to the instance or target. Last but not least, the deployment group acts like an environment such as Prod, QA, UAT, or Dev.

When working with deployment, you must understand the deployment types or strategies and which DevOps will be used to meet the company's needs. The deployment strategies define the network traffic to the environment, such as production, which can control the downtime and minimize it as much as possible. When you replace the old deployment with the new one, how will this be done?

To understand more about the deployment strategies, refer to Chapter 5.

Project: Create an Essential Pipeline Using AWS CodePipeline

One of the easiest ways to create a pipeline in CodePipeline is to use AWS's Setup Wizard setup. In this section, I'll show you how to deploy a web application in the S3 bucket to an EC2 instance using CodePipeline.

The steps are as follows:

1. Create an S3 bucket.
2. Install the CodeDeploy Agent on the EC2 instance.
3. Create the application inside CodeDeploy.
4. Create a pipeline to deploy the application to EC2.

Step 1: Create the S3 Bucket

This step is straightforward; from the console, choose S3, as shown in Figure 4-19.

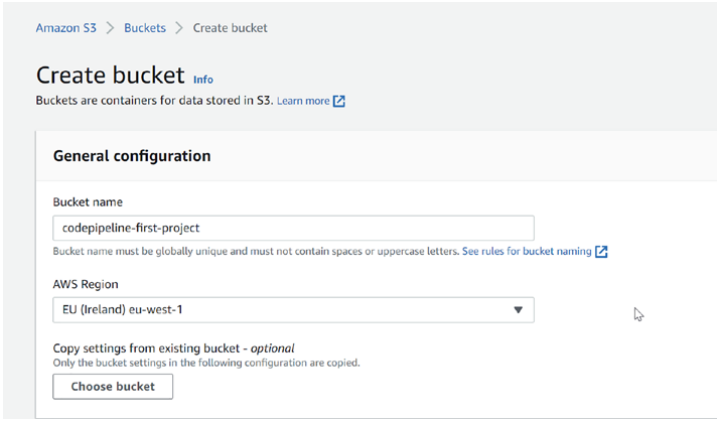

Figure 4-19. *Creating the S3 bucket code pipeline*

Download the application, which is already uploaded to GitHub (`https://github.com/OsamaOracle`). I am using an application that will be deployed on the Linux operating system. Then, upload the file to the S3 bucket we created before, as illustrated in Figure 4-20.

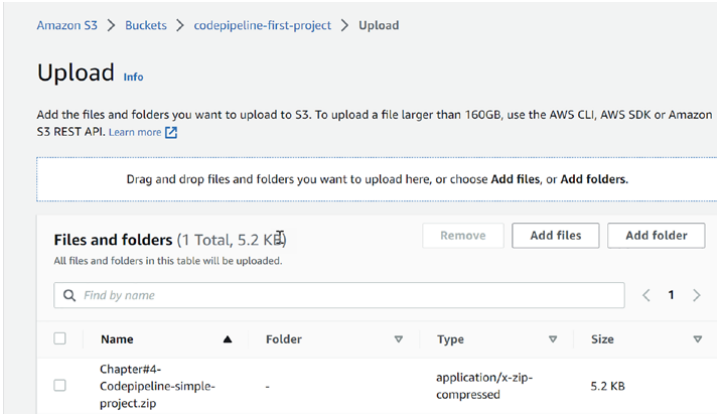

Figure 4-20. *Uploading the application to S3*

Grant Access to EC2

The next step will be creating an IAM role to grant access to the EC2, as demonstrated in Figure 4-21. (You can skip this if you're already familiar with this process.)

Caution You may notice that some of the steps in this section are the same for CodePipeline and CodeDeploy. The idea is to show and allow you to understand how easy the configuration is for both.

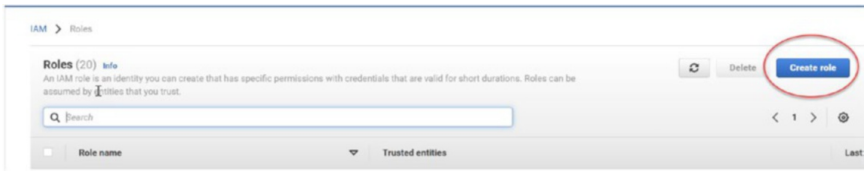

Figure 4-21. *Creating an IAM role*

Once you click "Create role," a new screen will appear, as shown in Figure 4-22. Click Next; making this role allows EC2 instances to call AWS services on your behalf.

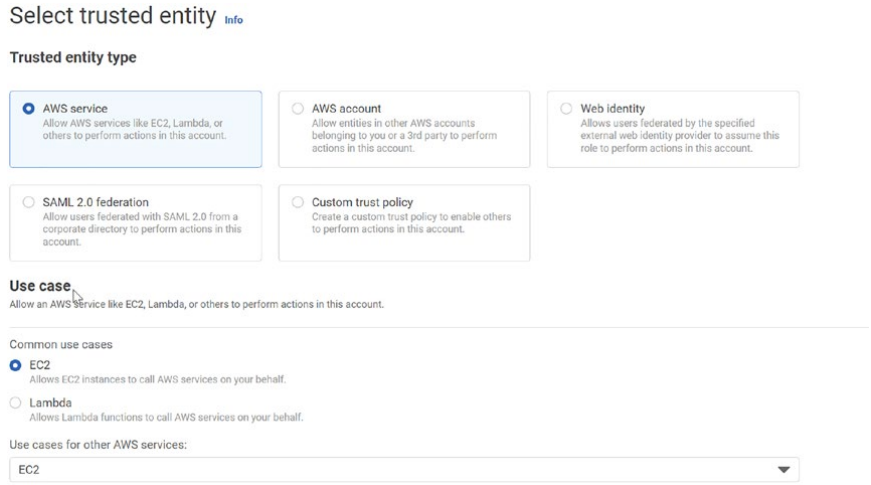

Figure 4-22. *Creating a a role for EC2*

On the next screen, you need to add permission to this role and search for a policy called *AmazonEC2RoleforAWSCodeDeploy* plus another policy called *AmazonSSMManagedInstanceCore*; see Figure 4-23 and Figure 4-24.

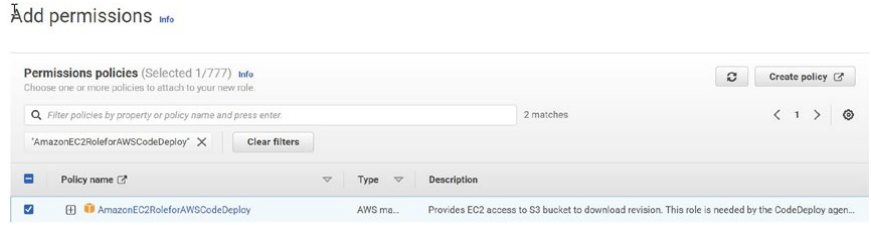

Figure 4-23. *Adding the AmazonEC2RoleforAWSCodeDeploy policy*

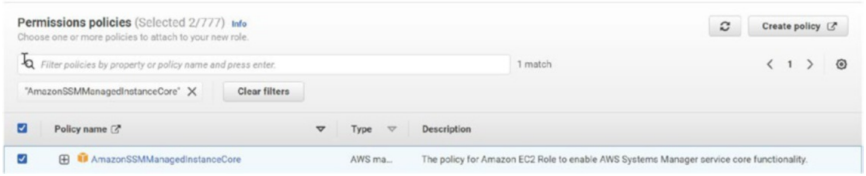

Figure 4-24. *Adding the AmazonSSMManagedInstanceCore policy*

Once you add these policies, name the role something you will remember later; for me, I called it **Ec2Codepipeline**, as shown in Figure 4-25.

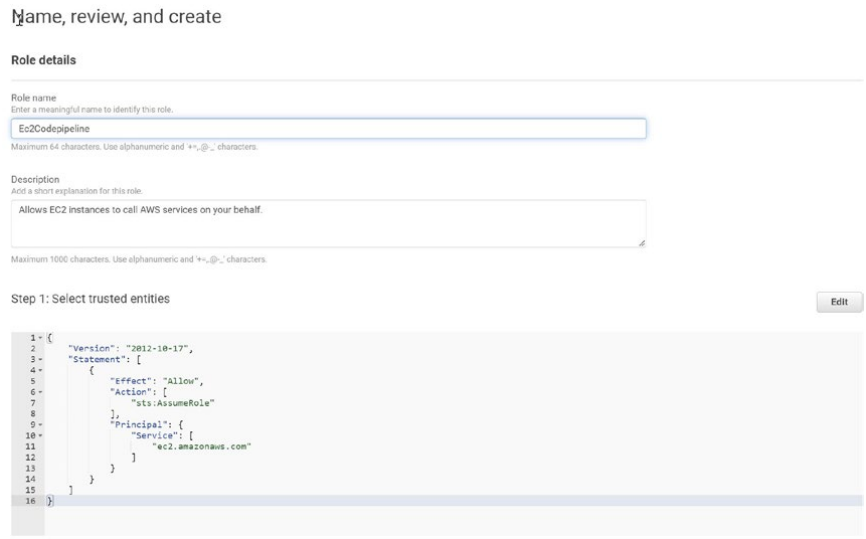

Figure 4-25. *Naming the role*

Step 2: Launch the EC2 Instance

Next, we need to launch the EC2 instance; I will not repeat the steps since they were discussed earlier in this chapter with two differences: assign the role to the EC2 instance, and the instance number needs to be 2. See Figure 4-26.

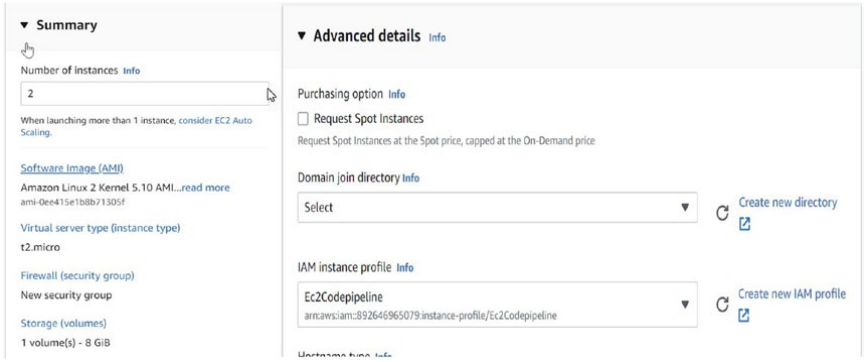

Figure 4-26. *Creating an EC2 instance for the CodeDeploy project*

Step 3: Install the CodeDeploy Agent on EC2

In this step, we need to create the application in CodeDeploy.

We will allow CodeDeploy to install the agent inside EC2 and choose the deployment strategy that CodeDeploy supports.

To do this, choose to create an application; the name is up to you. In our case, it will be the first project, but the compute platform should be EC2/on-premises, as shown in Figure 4-27.

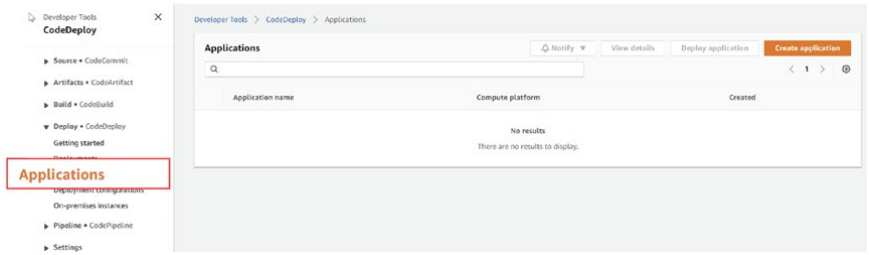

Figure 4-27. *Choosing an application, CodeDeploy*

- After creating the application, we need to create a deployment group for the application to choose the appropriate name.
- The services role should be created before.
- Under the deployment type, choose In-place.
- Set the deployment setting to CodeDeployDefault.OneAtOnce.
- Disable the Load Balancer settings.

  Figure 4-28 illustrates all of this.

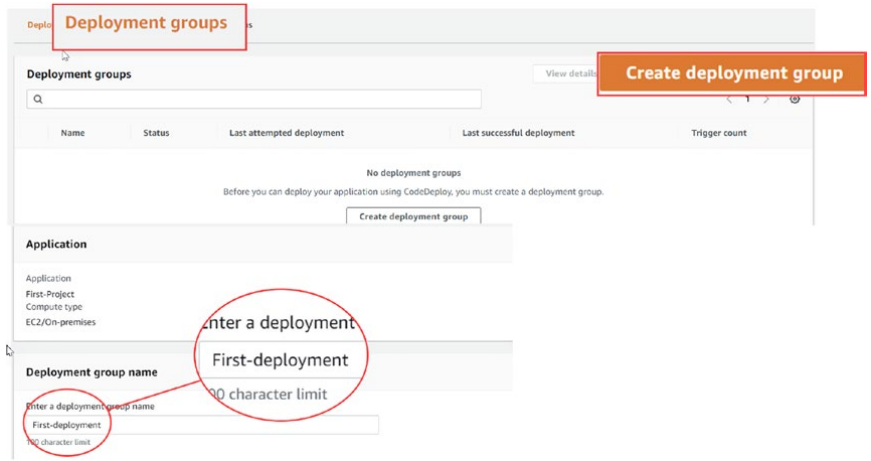

Figure 4-28. *After creating the application screen, you will be able to create a deployment group*

You need to configure the deployment type as shown in Figure 4-29.

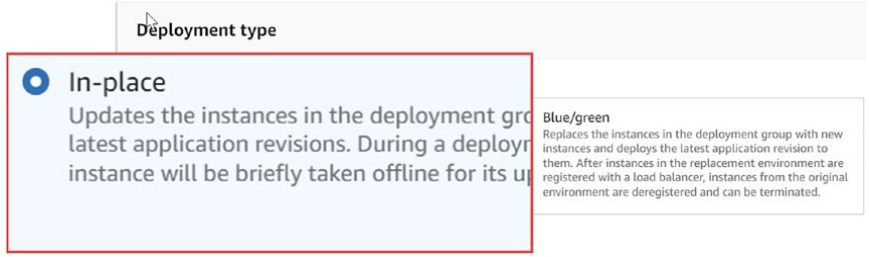

Figure 4-29. *Deployment type*

The next section is "Environment configuration," as shown in Figure 4-30.

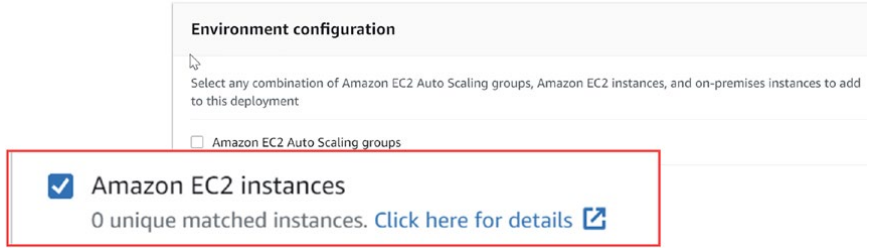

Figure 4-30. *Environment configuration settings*

Deployment configuration will be responsible for installing the CodeDeploy agent with AWS System Manager; you can choose how to install it. See Figure 4-31.

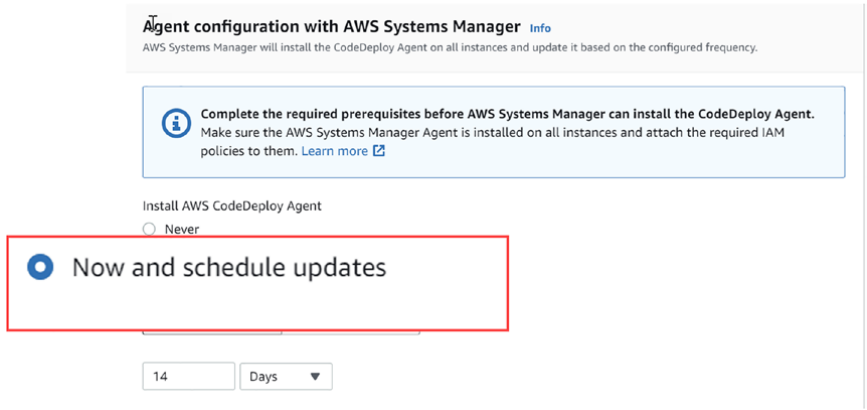

Figure 4-31. *Installing the CodeDeploy Agent*

The last step in the deployment group is to set the deployment settings, such as how fast an application is deployed and the success or failure conditions for deployment; see Figure 4-32.

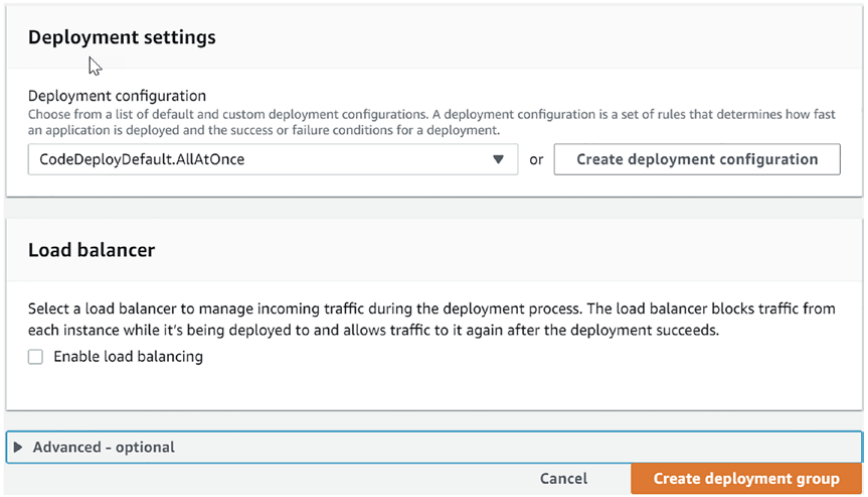

Figure 4-32. *Deployment group, deployment setting*

Step 4: Create the Pipeline

Now, we will create the pipeline from the console and choose CodePipeline, as demonstrated in Figure 4-33.

Figure 4-33. *CodePipeline main page*

Once you click to create a pipeline, a new screen will appear, asking you to choose a name for your pipeline.

For the role of the service, you have two options: create a new one or choose a previously created one; see Figure 4-34.

For the advanced settings, leave everything with the default values.

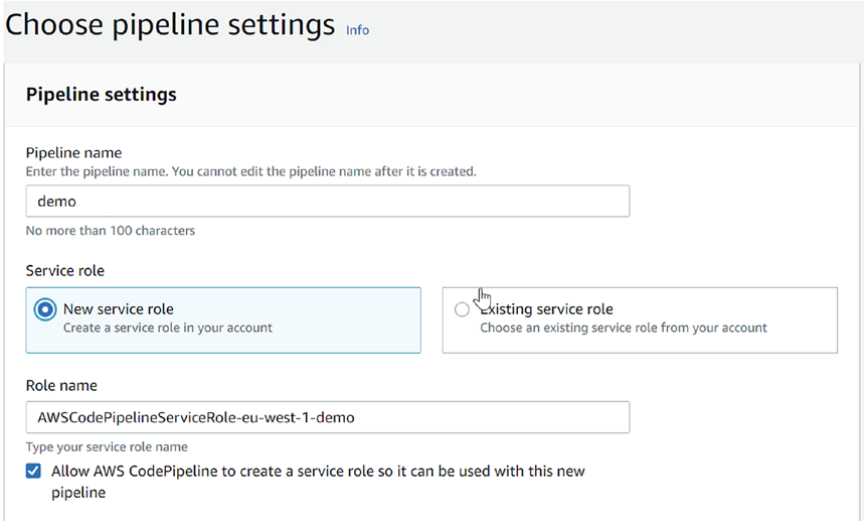

Figure 4-34. *CodePipeline settings*

Next, we must choose the application we need to create the pipeline for, which we already uploaded to the S3 bucket; see Figure 4-35.

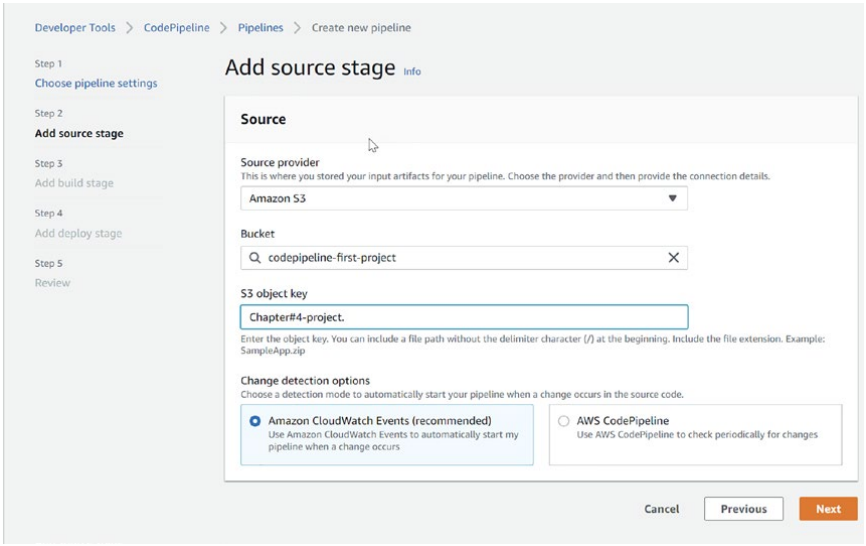

Figure 4-35. *CodePipeline's Add Source stage*

For the build stage, skip the build stage since we will not build anything; then click Next and confirm.

In the last step, the "Add deploy stage" page, choose CodeDeploy, the application name we created before, and the associated deployment group, as shown in Figure 4-36.

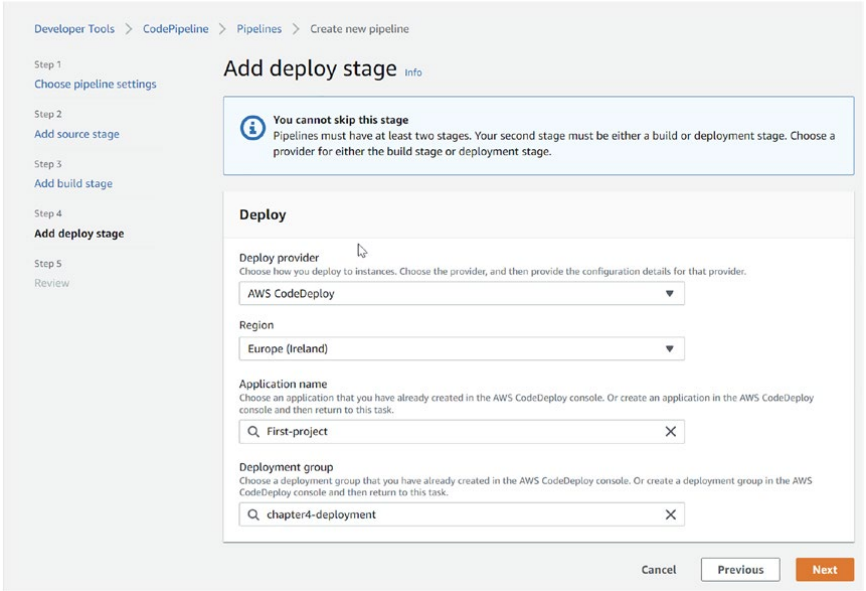

Figure 4-36. *Deployment stage for CodePipeline*

Congratulations, you have just created your first project with CodePipeline. You can add stages to your pipeline, allowing you to understand CodePipeline more.

The previous examples are straightforward and allow you to understand how CodePipeline works. I will demonstrate another example, but let's use CI/CD and integrate CodePipeline with GitHub this time.

Integrate CodePipeline CI/CD with GitHub

The idea of this project is to deploy a simple static website written in HTML from GitHub to Amazon S3; when the developer makes a code change, they will create a GitHub pull request to merge the code with the master/main branch. The merge will trigger an event (GitHub integration within CodePipeline) and push these changes depending on the configuration we wrote to S3.

The first step is to create the bucket that will host the website; in my case, I called it **website-static** with no particular configuration; the S3 bucket properties will be as follows and are shown in Figure 4-37.

- No public access.
- No version.
- No policy or security features are enabled.

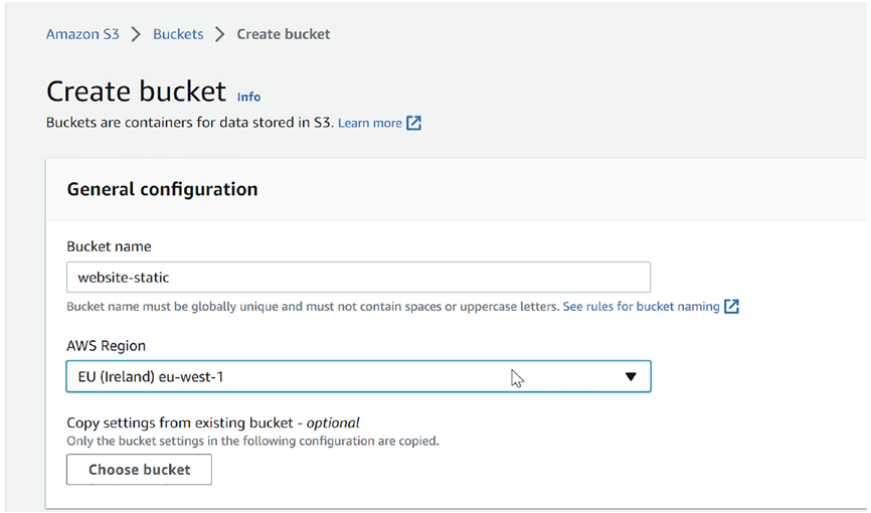

Figure 4-37. *Creating a website-static bucket*

The next step will be to create the code pipeline from the console. Select CodePipline and click "Create pipeline," as shown in Figure 4-38.

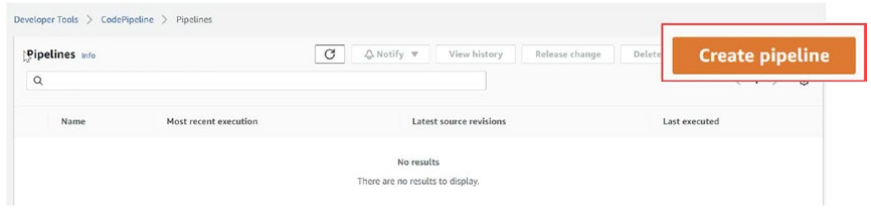

Figure 4-38. *Creating a pipeline for the project*

After clicking the button, the console will redirect you to another page, the configuration page; see Figure 4-39.

Choose a clear name for your pipeline so you will understand its purpose.

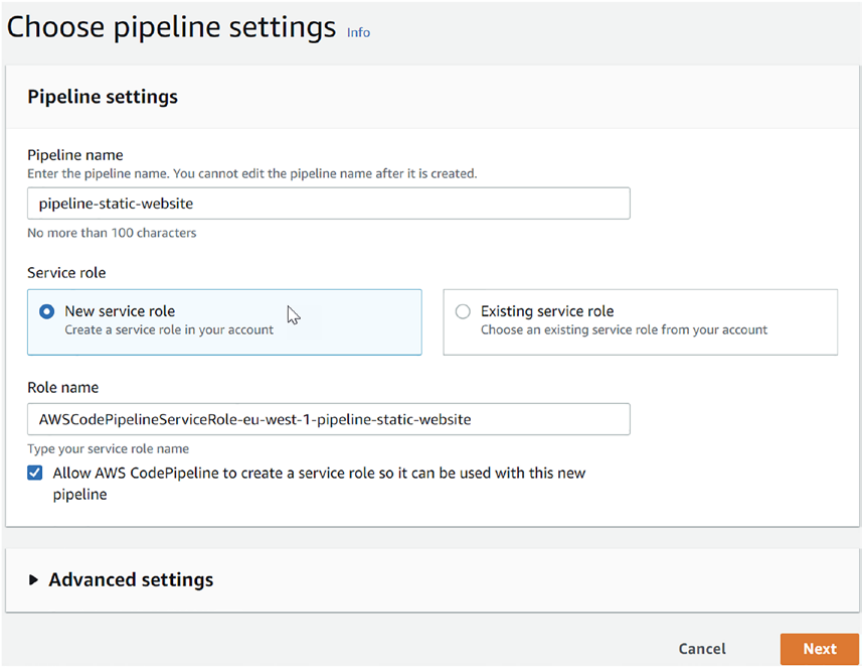

Figure 4-39. *Pipeline configuration, choosing a pipeline name*

On the same page, AWS will require you to create a service's role, which is a set of permissions to allow CodePipeline to interact with other AWS services; either you can keep the services role name set to the default or you can choose the name by yourself.

In the advanced settings, you will choose the bucket, as shown in Figure 4-40, or you can allow a pipeline to create one for you, but CodePipeline will set the name.

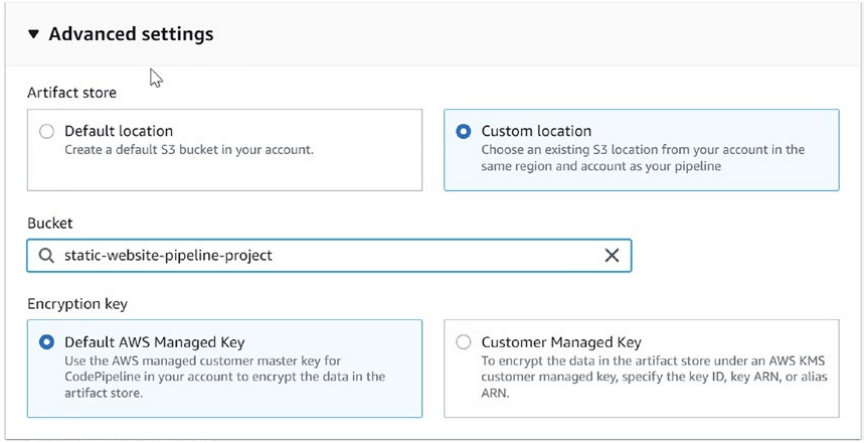

Figure 4-40. *Choosing an S3 bucket name*

Click Next, and the following screen will allow you to choose the source provider where the code was uploaded; in our case, it will be GitHub version 2.

The screen will be expanded to allow you to connect GitHub with the pipeline and create the connection, as shown in Figure 4-41.

Select the Connect to GitHub button on the right if this is your first time connecting to GitHub using CodePipeline. Doing this will enable a new AWS Developer Tools window, allowing you to establish a new connection. You will be requested to log into GitHub if you are not already.

Note The difference between GitHub version 1 and GitHub version 2 is that in GitHub version 1, you can download only the source; GitHub version 2 will allow you to clone it so you can access the metadata.

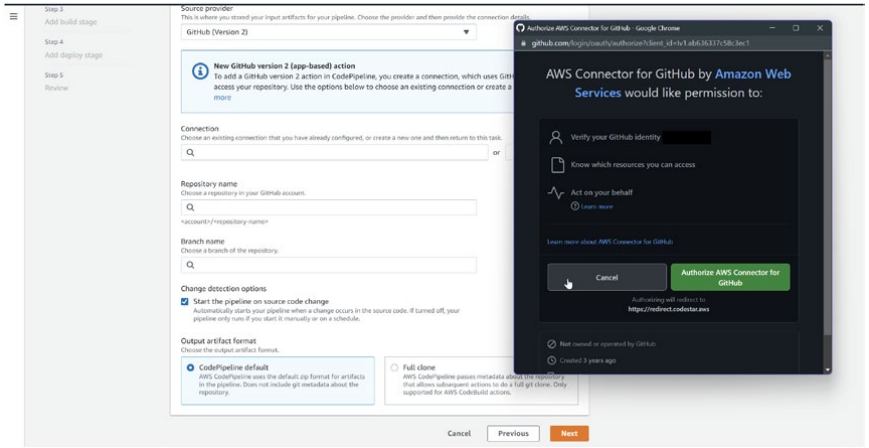

Figure 4-41. *Pipeline configuration, GitHub integration*

Allow AWS to install the connector when you create the connection, as shown in Figure 4-42.

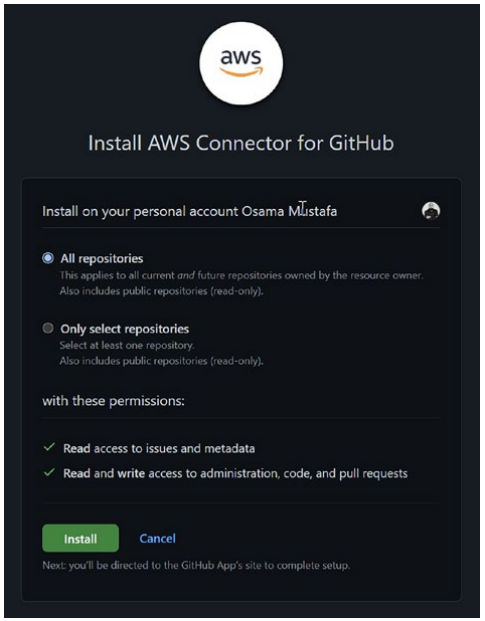

Figure 4-42. *Installing the AWS connector*

Once you are done with the connection, you will be redirected back to the screen; you need to choose the information of the code repository and which branch, as shown in Figure 4-43. Click Next once you are done.

Remember to choose the right repository name; once you create the connection, your repository names will be shown in the field, as well as which branch you will use for this code.

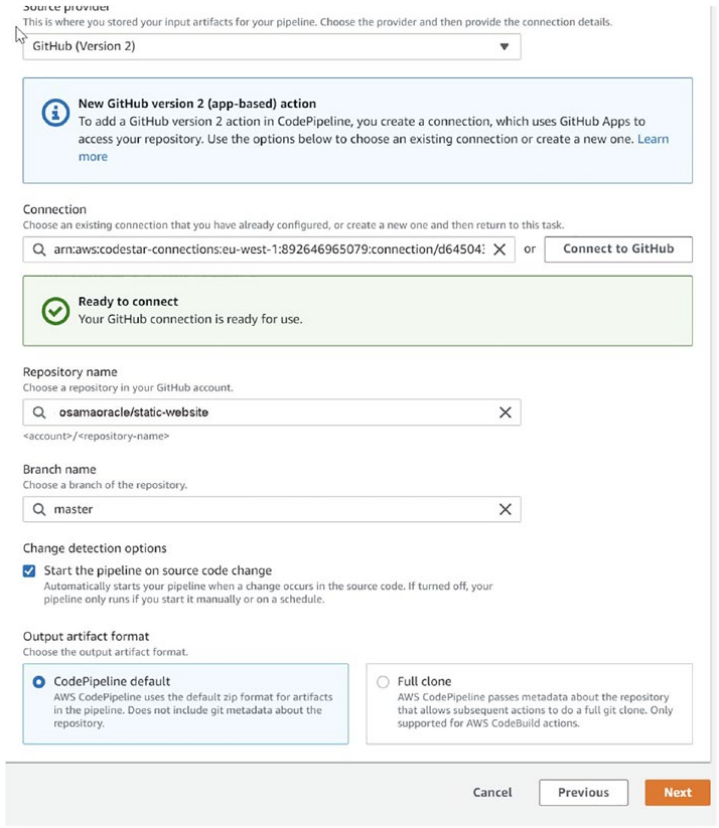

Figure 4-43. *Pipeline configuration, GitHub connection*

Once you click Next, the following configuration will be the build stage, and since this is a static website and we will not build anything or have any artifactory, we will skip, confirm, and click Next. See Figure 4-44.

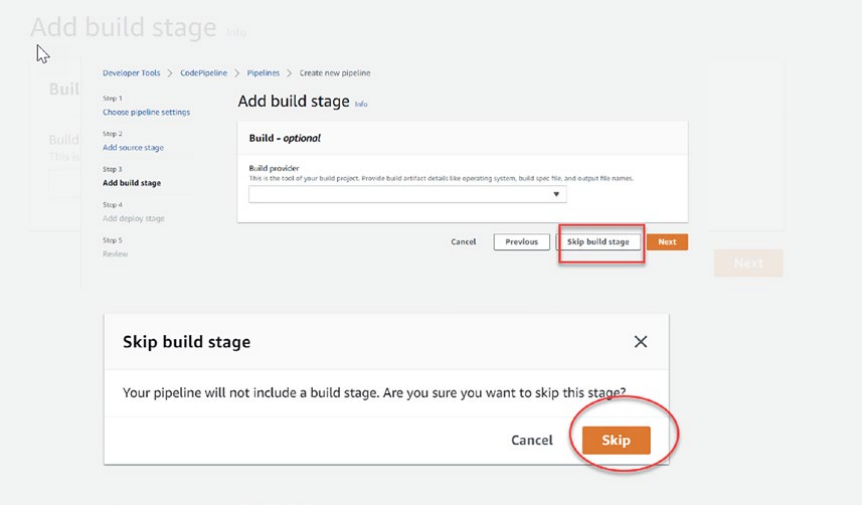

Figure 4-44. *Pipeline configuration, skipping the build stage*

On the next screen, AWS provides a different deploy provider; in our case, it will be Amazon S3. Choose the correct region for the created bucket, the bucket name, and the development path in case you deploy to a specific folder inside S3.

Select "Extract file before apply." This option will hide the S3 object key; see Figure 4-45.

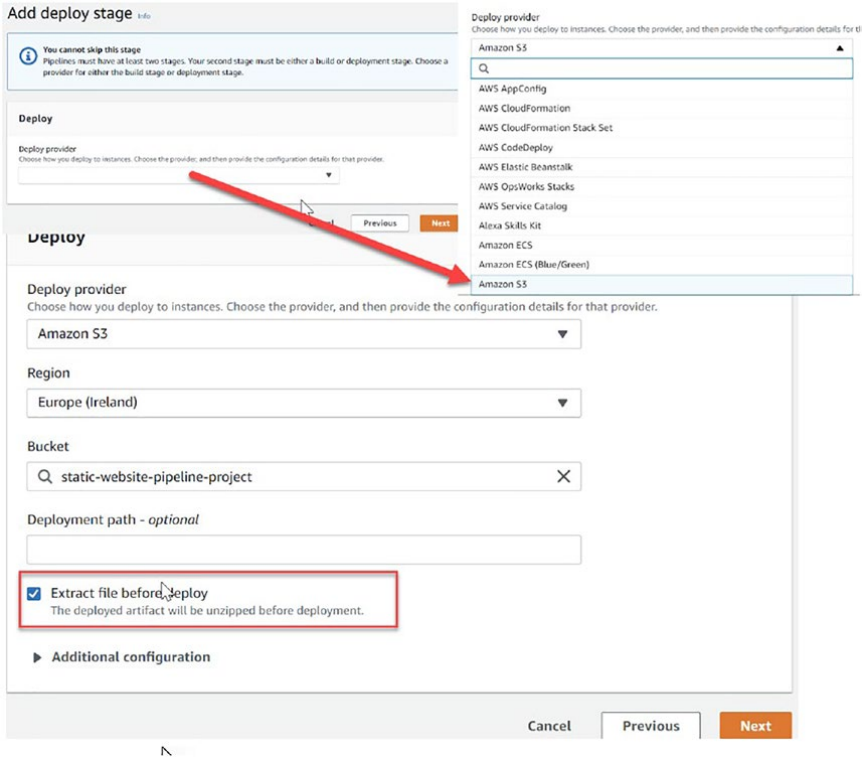

Figure 4-45. *Pipeline configuration, deployment stage*

Review all the information. If you are happy with the configuration and setup, click "Create a pipeline." The first trigger will be automatic once you click the button; see Figure 4-46.

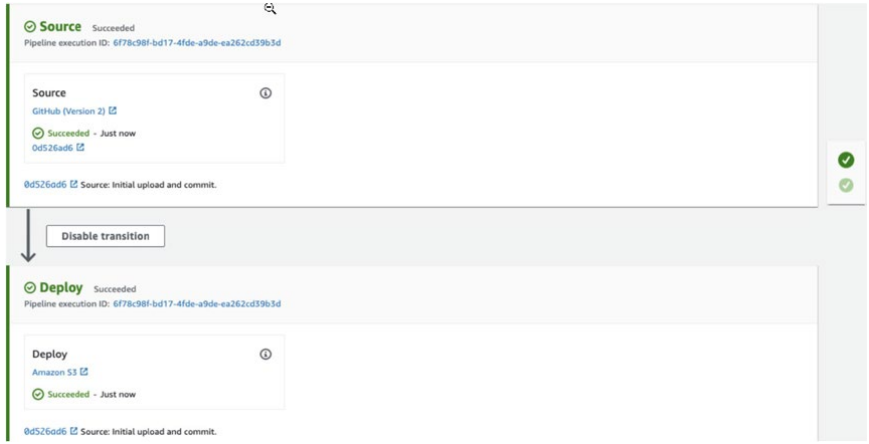

Figure 4-46. *Pipeline configuration, CodePipeline results*

If you check the S3 bucket after the pipeline run is finished, you will see the code has been uploaded there.

Summary

In this chapter, you learned more about using continued deployment inside Amazon AWS and learned what kind of services match this concept.

Additionally, we reviewed the deployment strategy in general and the one AWS provides; in this chapter, I tried to cover the concepts with real-life examples to allow you to understand the use case and the different configurations each time.

In the next chapter, we will cover one of the essential parts of the deployment, which are the different deployment strategies. In general, we will discuss each one of them in detail, compare them, and discuss the use case for them. After discussing the strategies in general, I will cover them from Amazon AWS's point of view.

CHAPTER 5

AWS Deployment Strategies

In the previous chapter, I covered deployment and how to create a pipeline. Still, one of the central concepts when building the pipeline is understanding the deployment strategy and which type you will use, because there are multiple deployment types, each serving a different purpose depending on the use case and the company approach.

In this chapter, I will discuss general and specific deployment strategies. I'll also cover the types offered by AWS.

What Is a Deployment Strategy?

A deployment strategy is the method by which a company, development team, or DevOps department releases a new version of its software. This method will depend on how network traffic will be distributed to the latest release and how to deal with it.

These are the deployment types:

- Blue/green deployment
- Canary deployment
- A/B testing deployment
- Re-create deployment
- Ramped deployment (rolling upgrade)
- Shadow deployment

Let's look at each in detail.

O. Mustafa, *A Complete Guide to DevOps with AWS*, https://doi.org/10.1007/978-1-4842-9303-4_5

Blue/Green Deployments

One of the most common deployment strategies is when the software's new version runs next to the old version.

Once the QA team tests the latest version and ensures it's running without any errors, the load balancer will switch the traffic from the old version to the new version. This is also called *red/black deployment*.

- The blue environment is running current application version.
- The green environment is running the new application version.

The first, more outdated version is the blue environment, while the latest, most up-to-date version is the green environment.

As shown in Figure 5-1, the subsequent action is to transition to a green environment by routing all user traffic to the new setting. Because this move can take place extremely rapidly, there won't be any downtime for the users. In addition, if you need to undo the changes, you can do so immediately by redirecting the traffic to the previous version.

This approach offers several advantages.

- You will have two versions of your application, so you can roll back anytime and test without affecting the live version.
- You avoid having multiple versions of the application.

However, there are some potential disadvantages.

- Replicating the environment makes the process complex and requires extra work.
- It is useful for a stateful application.
- It requires careful testing.
- There is an extra cost.

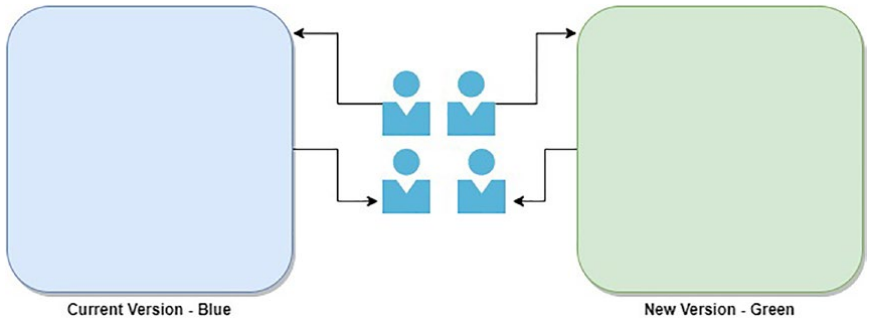

Figure 5-1. *Blue/green (red/black) deployment*

Canary Deployments

One of my favorite deployment strategies is *canary deployment*. In this type of deployment, the development team will prepare the new version, but they will gradually shift it into production based on a percentage.

For example, the DevOps will start deploying the new version gradually, say, 10 percent of it; then the QA will test that much and inform the team if there are any bugs or errors.

The percentage will be increased to 15 percent, 20 percent, up to 100 percent; this will help the team test the new version's stability bit by bit. Canary deployments offer a significant level of control. However, it may be challenging to put this into practice. See Figure 5-2.

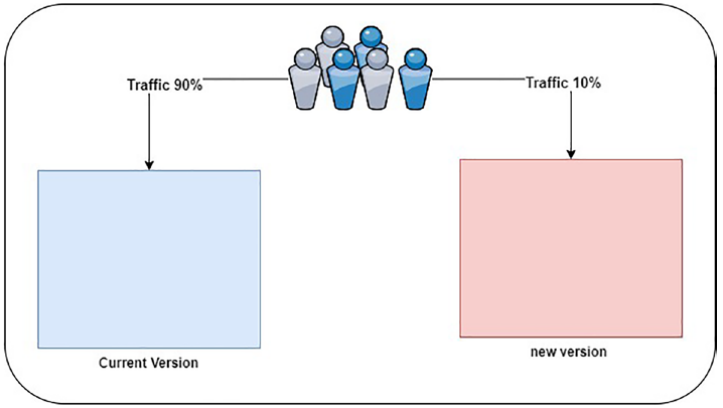

Figure 5-2. *How canary deployment works*

The advantages of this kind of deployment are many.

- *A/B testing*: You provide two alternatives to the users, increasing user engagement; the users will be assigned to group A or B.
- *No downtime*: As you can see, there is no downtime.
- *Easy rollback*: If something goes wrong with the new deployment, you can easily roll back to the previous version. See Figure 5-3.

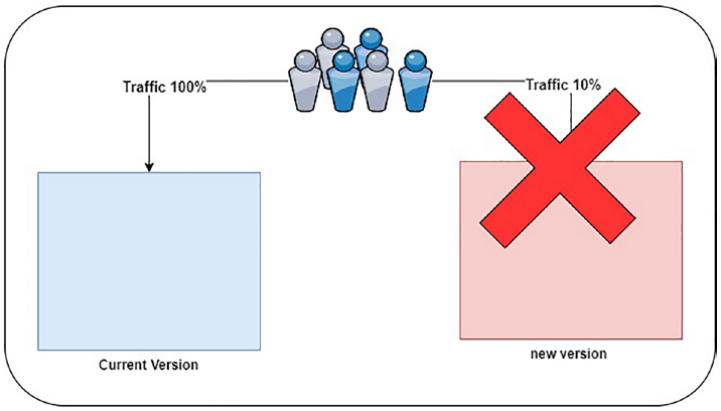

Figure 5-3. *Fast rollbacks in case of errors*

- *Fast feedback*: The QA will provide quick feedback about each stage, which makes updating the application much more effortless.

The disadvantage is that it can be slow to roll out.

A/B Testing Methodologies

In an *A/B testing* kind of deployment, the developers will deploy the new release next to the old release. The difference is that the new release will be available only for certain people, most likely QA to do testing and some developers to fix any reported issues; see Figure 5-4.

The good thing about this kind of deployment is that it's a different environment, like the blue/green deployment. This does consume more resources while the team is testing, but once the testing is done, the old version can be deleted.

Another downside is that it does create more work, because the developer will have to maintain and work on two different environments to ensure both of them are performing correctly.

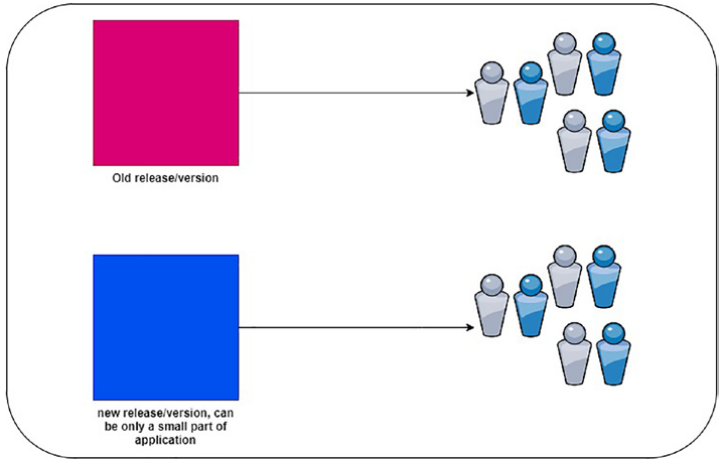

Figure 5-4. *A/B testing deployment example*

The advantage is that you can control the traffic.
These are the disadvantages:

- The load balancer setup can be complicated.
- Troubleshooting is not easy for this deployment.

Re-create Deployments

This type is straightforward; the system administrators will shut down the old system, deploy the new release, and immediately reboot the servers, as shown in Figure 5-5.

There will be downtime between the shutdown and the machine's startup.

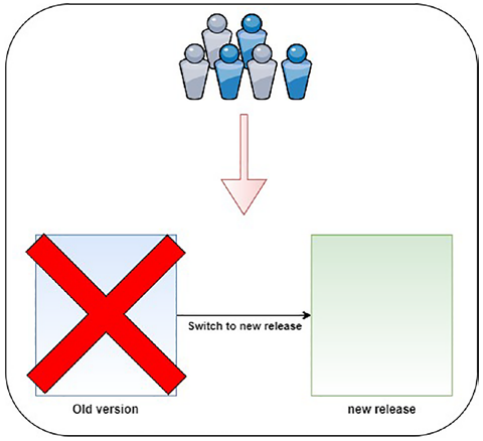

Figure 5-5. *How the re-create deployment works*

The good thing about this deployment is that it's cheaper, with no need for a load balancer to shift the traffic from one to another environment; the bad thing about it, as you can see, is the downtime, which is not reasonable for some companies.

The advantage is that it's an easy deployment to set up.

The disadvantage is that downtime may be extended and affect the users.

Ramped Deployments (Rolling Upgrades)

Please don't mix this deployment type up with canary deployment. By using canary deployment, early adopters can get their hands on a new software version before the general public. Rolling deployments focus on specific servers, while a canary technique selects a subset of users to test the new software before rolling it out to everyone.

Compared to canary deployment, rolling/ramped deployment gives certain users the updated program version.

This approach will gradually replace the old version/release with the new version/release; see Figure 5-6.

After replacing the environment entirely with the new version, the old one can be shut down; using this deployment will allow the team to monitor the performance of the new release and roll out quickly in case something happens.

IT departments need to worry about only one production environment for a web program when using the rolling deployment method. In other words, IT administrators first stagger change releases to install a new application version on select servers or instances. Some servers run the new program; others run the old one. Some users utilize the updated code, while others use the production version.

This type comprises a network of computers or instances in the cloud, each running its own copy of the program. In addition, a load balancer is often used to distribute user requests over many servers.

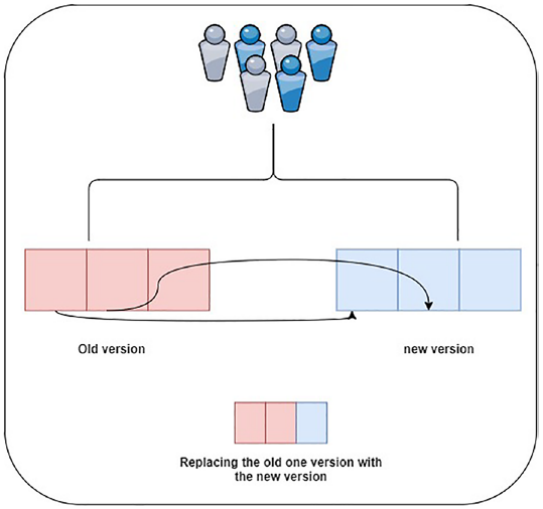

Figure 5-6. *How ramped deployment works*

Here are the advantages:

- It's an easy deployment to set up.
- The new version will be replaced slowly and needs time to be tested.
- It's one of the best deployments for stateful applications.

These are the disadvantages:

- Supporting the application can be exhausting for this kind of deployment.
- You cannot control the traffic.

Shadow Deployments

One of the most complex deployments that DevOps can do is a shadow deployment.

The team will deploy the new and old versions together, but the users cannot connect to the latest version immediately, meaning it will stay "in the shadows" (where the name comes from); see Figure 5-7.

After that, the developer will send a copy of the requests to the shadow version, which will help test the new version and the performance of the latest release.

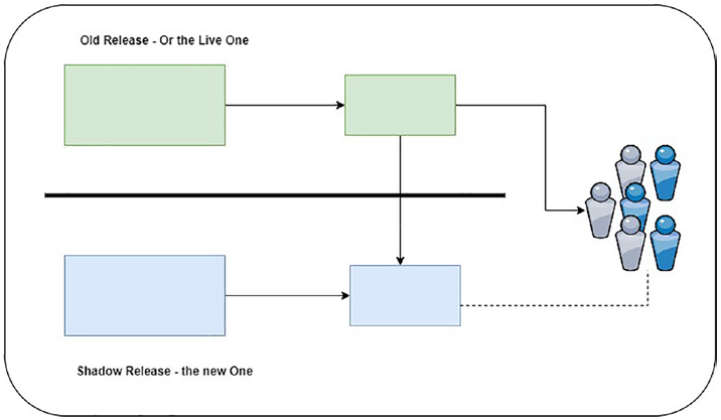

Figure 5-7. *How shadow deployment works*

These are the advantages:

- There is zero impact on the users.
- You can monitor the performance with real production traffic.

These are the disadvantages:

- Supporting this kind of development isn't straightforward.
- It is complex to set up.

The previous deployment strategies are for DevOps in general. We need to discuss the Amazon AWS deployment strategies, which are similar to the previous ones.

In the next section, I will discuss the deployment strategies from Amazon AWS's perspective; you will notice that the names are the same. In addition, Amazon AWS added a new deployment strategy to its cloud, which will be discussed in the next section.

Amazon AWS uses the strategies mentioned earlier in this chapter with an addition or a new name; this will be explained as it comes up.

Amazon AWS Deployment Strategies

To create a deployment solution that is complete and fully functional, you have first to establish the required deployment methods. Your ideal equilibrium of management, efficiency, price, tolerance for risk, and other variables may influence the deployment strategies you choose to utilize to update your application. Each deployment service offered by AWS is compatible with several different deployment techniques.

The following are the deployment options offered by AWS:

- In-place deployments
- Prebaking versus bootstrapping AMIs
- Blue/green deployments
- Canary deployments
- Linear deployments
- All-at-once deployments

In-Place Deployments

From the name, you can understand that this deployment will update the application without providing any new infrastructure; the old one will be stopped and then updated with the latest version. This kind of deployment is similar to the re-creating deployment method.

Deployment settings for one at a time, half at a time, and all at once are available with AWS CodeDeploy and AWS Elastic Beanstalk.

The following are the advantages:

- The cost remains the same; there is no need to provision new infrastructure.
- The infrastructure and administration/management will be the same.

The following are the disadvantages:

- Rollback will be slow since you need to reinstall the old version.
- The application needs to be tested well.
- Downtime may occur depending on the infrastructure setup on the cloud.

Prebaking vs. Bootstrapping AMIs

When application components are prebaked into an Amazon Machine Image (AMI), it may reduce the time needed to start and run an Amazon EC2 machine. During deployment, it is possible to rapidly pair the prebaking and bootstrapping methodologies to construct new instances adapted to the current environment.

Deploying your application with any necessary dependencies or updates when an Amazon EC2 instance is started is known as *bootstrapping* an EC2. It is possible for deployments and scaling events to be delayed if a complicated application or substantial downloads are necessary.

The following are the advantages:

- Bootstrapping lets you quickly reproduce the Dev, QA, and Production environments.
- You can construct a setting where one may heal and learn about oneself.
- It improves Amazon AWS resource management.
- It is considered a low-cost option in the cloud.

The following are the disadvantages:

- It is not a consistently smooth option and comes with a risk of failure.
- The administration's efforts probably will increase due to the complexity.

Linear Deployments

Linear is considered part of blue/green deployment with some canary deployment features; when traffic is moved via a linear deployment, each increment takes the same amount of time. The proportion of traffic transferred in each increment and the interval in minutes between each increment are specified in predefined linear options.

The following are the advantages:

- There is no downtime.
- Few users will be affected if the new environment doesn't work correctly.

The following are the disadvantages:

- The deployment will take time.
- There is an extra cost.

All-at-Once Deployments

In another type of blue/green deployment, the traffic will be moved from the old version to the new one.

The following are the advantages:

- There is no downtime.
- A few users will be affected if the new environment doesn't work correctly.

The following are the disadvantages:

- The deployment will take time.
- There is an extra cost.

Note During the writing of this book, Amazon AWS announced fully managed blue/green deployments in Amazon Aurora and Amazon RDS, something new and not yet implemented. This type of deployment will eventually make our lives much easier.

To understand what this means, you can build a distinct staging environment that is fully controlled, synchronized, and a reflection of the production environment by using blue/green deployments. The staging environment will make a copy of the primary database used in your production environment and any in-region read replicas. Blue/green deployments use logical replication to ensure these two environments remain in sync.

You won't have to worry about losing any data if you make the staging environment the production environment instead. Blue/green deployments will prevent writes from being made on both the blue and green environments to prevent any data from being lost during the switchover.

Once you do the configuration, the green environment will catch up to the blue one. The production environment's traffic is then redirected to the newly promoted staging environment using blue/green deployments, which occurs without any modifications to the application's code.

Blue/Green Deployments for MySQL RDS

Search for *RDS* in the AWS Management Console, choose the databases that must be modified in the console, and then select Create Blue/Green Deployment from the Actions drop-down option, as shown in Figure 5-8.

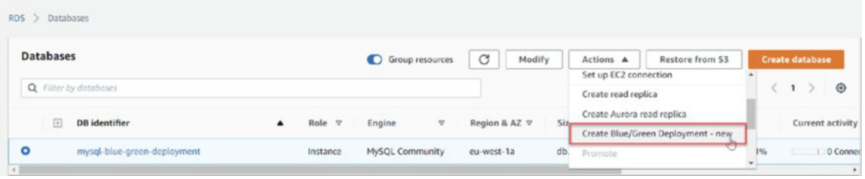

Figure 5-8. *Amazon AWS new feature RDS blue/green deployment*

You can modify the properties of your database, like the engine version, DB cluster parameter group, and DB parameter group for green databases. You can also establish a blue/green deployment identification associated with your database.

To utilize a blue/green deployment in your Aurora or RDS instance, in our case MySQL, you will need to switch on binary logging by changing the value for the `binlog_format` parameter in the DB cluster parameter group from `OFF` to `MIXED`.

Once you choose this option, the configuration is straightforward (Figure 5-9).

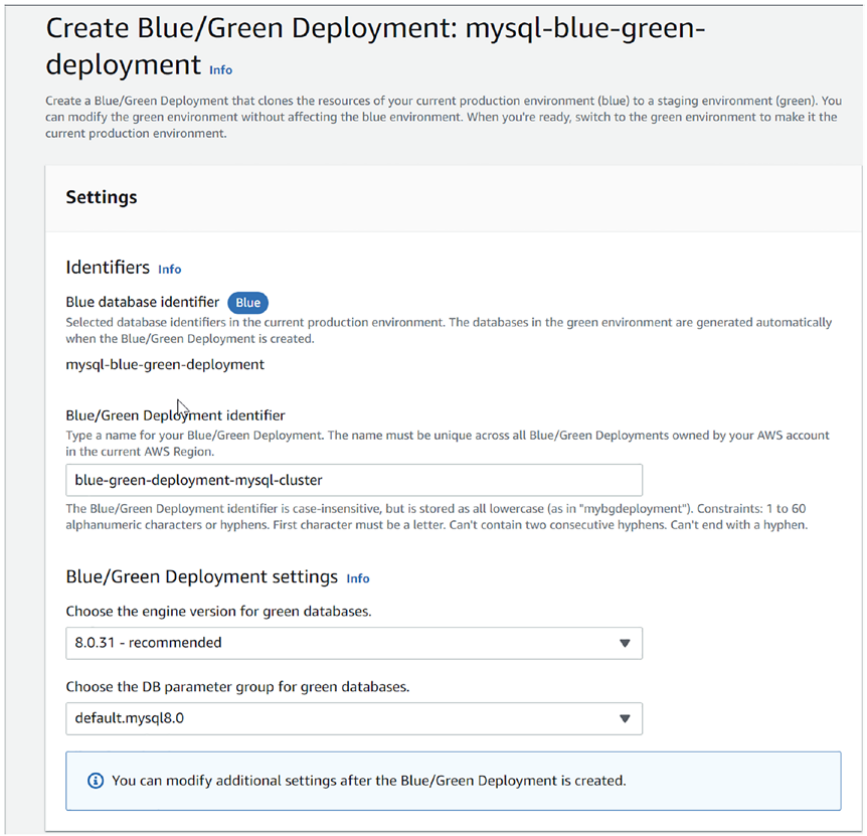

Figure 5-9. *Amazon AWS blue/green deployment configuration*

The database will be ready for production when you choose Create Blue/Green Deployment from the drop-down menu since it will create a new staging environment and execute automatic operations. Note that you will have to pay for this feature of the database, which includes reading replicas and DB instances in multi-AZ deployments and any additional options you may have activated on the green one, such as Amazon RDS Performance Insights.

When the creation process is finished, you will have access to a staging environment prepared for testing and validation before it's promoted to the new environment.

At this point, you are close to being ready to move your green databases into production; check if your deployment finished successfully and that the databases are prepared for the transition. There is also another option which is helpful, "timeout," that allows you to define the maximum time limit for your switchover.

Plus, once you do the switchover, the blue/green deployment will not delete the old production environment, just in case you still need access.

The RDS blue/green deployment option is beneficial when you need to create, for example, a DR solution or a staging environment that will allow the developer to work on it and sync with production instead of performing manual work.

If you want to do this using AWS CLI, it's simple, as shown here:

```
1. aws rds create-blue-green-deployment
2. --blue-green-deployment-name blue-green-deployment-mysql-cluster
3. --source arn:aws:rds:eu-west-1:223344555:db:mysql-blue-green-deployment
4. --target-engine-version 5.7
5. --region eu-west-1
```

Summary

In this chapter, we discussed the ways I could deploy an application into the cloud or even on-premises; in addition, we talked about types of deployment strategies.

We covered the types of deployments offered by Amazon AWS and compared each, mentioning the cons and pros.

Before deciding what to implement in your system, you need to investigate a variety of facets and contrast the possible deployment strategies. Using one of them depends on the use case and business needs.

In the next chapter, we will go through different DevOps categories. One of the most common uses is infrastructure as code; the next chapter will also cover a variety of tools, such as Terraform, CloudFormation, and Pulumi, as show how to use them.

CHAPTER 6

Infrastructure as Code

Infrastructure as code (IaC) is one of my favorite subjects; in this chapter, I'll break it down and focus on the most important details regarding how it works with AWS.

In this chapter, I will cover IaC in general, the benefits of using IaC, and why you need to use it. I'll also cover IaC tools, such as Terraform, CloudFormation, Pulumi, and Ansible, and the difference between them, as well as when you should use each one of these tools.

What Is Infrastructure as Code?

As a DevOps engineer, you will hear about IaC a lot, especially if you or your team is deploying multiple resources such as databases, virtual machines, load balancers, and more.

Say you have different environments such as QA, UAT, development, preproduction, production, and maybe more. Imagine redeploying the same resources to these environments over and over again. Figure 6-1 shows the effort you need to make without IaC.

O. Mustafa, *A Complete Guide to DevOps with AWS*, https://doi.org/10.1007/978-1-4842-9303-4_6

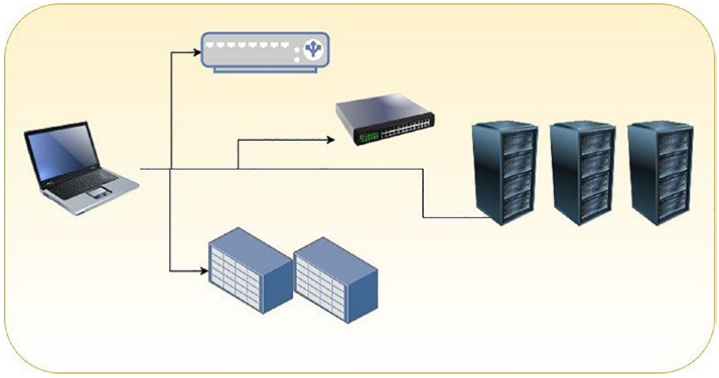

Figure 6-1. *The old way of deploying resources one by one*

You should be able to produce more value in less time using cloud and infrastructure automation. This allows you to do it more reliably, saving money and time for your company. In reality, however, using the cloud results an ever-increasing volume, level of complexity, and variety of items that need to be managed.

One of these advantages is automation. Other advantages are that the cloud can solve the resource and life-cycle management complexity, versioning, and configuration. Basically, the method of infrastructure automation known as *infrastructure as code* is modeled around and borrows techniques from the field of software development. It emphasizes the importance of consistent processes that can be repeated for modifying the configuration of resources such as compute, storage, network, load balancing, and provisioning. When changes are made to the definitions, those modifications are subsequently "pushed out" to the systems by making use of processes that are unattended and include rigorous validation. Figure 6-2 shows you how you can use IaC to deploy to different environments.

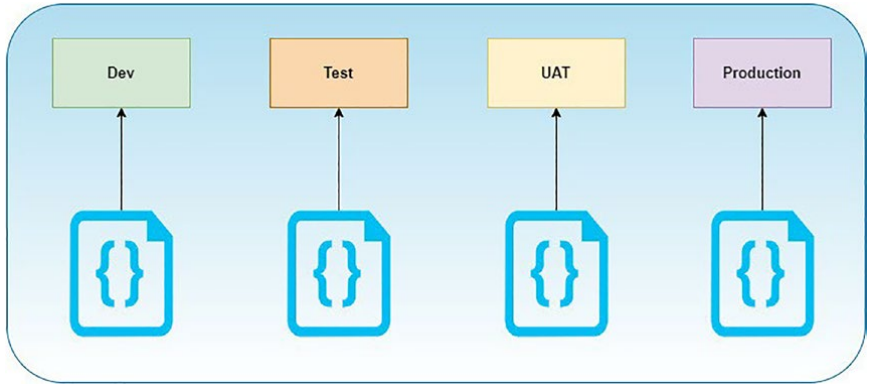

Figure 6-2. *IaC multiple-environment deployments*

DevOps engineers write the IaC files; upload them to the company version control, for example, GitHub or GitLab; and share the changes done on the code with the other team members so that the deployment and code enhancements will start.

After that, the DevOps engineer needs to keep updating the code based on the team review and pushes the changes to the pull request/merge requests; once the team comments are finished, the pull request will be approved and merged with the main branch.

To simplify, Table 6-1 shows the cloud approach versus the classical one.

Table 6-1. *Apporach Followed in the Cloud*

Factor	Classical Approach	Cloud Approach
Hardware	Physical hardware; the company needs to wait to set up this hardware.	Virtualized resources and quick setup. No need to wait.
Provisioning	It will take time at least two weeks, depending on the environment.	Fast; probably will take minutes.
Provisioning type	Manual configuration.	It can be a manual or automated configuration.
Changes	It will be risky, and you need to have a rollback plan in case the changes fail.	Can be done to improve the infrastructure; very fast rollback; most of the changes automated.
Architecture	Monolithic.	Microservices and monolithic.
Change cost	High.	Low.

Why Do You Need IaC?

The following are the kinds of results that many different teams and organizations want to accomplish by using infrastructure as code:

- With IaC, all the teams can be synced with each other, and they can follow up and know what changes have been made to the company infrastructure; there are no secrets anymore.
- Developers will have visibility over infrastructure resources and their configurations; they will also be able to define, provide, and organize their needs, eliminating the need to rely on IT employees to carry out these tasks on their behalf.
- Rollback and failure accidents are easy to fix; if you deploy something and cause an issue with IaC, roll back.
- Enhancement and improvement never stops with IaC; the time spent by IT staff members is not wasted on mundane, routine work but on activities that put their skills to good use.
- Instead of discussing potential solutions in meetings and documents, it is more effective to put them into action, testing them and measuring their effectiveness.

There are a couple of things you need to understand before you start working with infrastructure as code. Let's look at some myths about the process.

Myth 1: I will use IaC to deploy first, followed by automation.

I have seen clients writing the code for infrastructure and using it to deploy the application. It's good practice and an automated one, but don't expect to make changes from the console later or make changes manually.

Ask your DevOps engineers to update the code; everything must be done using IaC once you decide to use it.

There are three main problems in implementing automation later:

- Automation should allow quicker delivery times for everything; automation's advantages are lost if executed after the bulk of the job has already been completed.
- When you use automation, creating automated testing for your software is much less of a hassle. And when issues are discovered, they can be fixed and rebuilt more rapidly. If you include this in the construction process, you will have a superior infrastructure.
- It is challenging to add automation to an established system. Automation occurs during the planning and execution of a system. Adding automation to a system not designed initially with automation in mind requires significant changes to the system's architecture and implementation.

In the cloud, infrastructure that isn't automated quickly becomes a loss. The price of manual upkeep and repairs may add up rapidly. That is, if the function it performs goes off without a hitch.

IaC solves the problem of release pipeline environment drift. Teams must manage deployment environment parameters without IaC. Each habitat becomes a "snowflake" that can't be replicated automatically.

Environment inconsistency might hinder deployment. Manual infrastructure maintenance is error-prone and hard to monitor.

IaC infrastructure deployments are repeatable, avoiding runtime difficulties from configuration drift or missing dependencies. Release pipelines configure target environments. The team modifies the source, not the target.

Why are people trying to avoid IaC while making the changes? As shown in Figure 6-3, it's simple: fear.

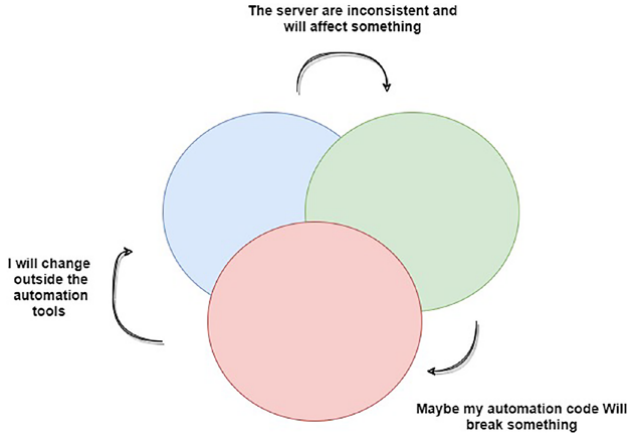

Figure 6-3. *The automation fears*

Code complicates things and requires support engineers to know a lot. This is new ground for some, but it is the most critical component for backup, feature requests, and resources. Support and code don't always match.

No one can remember every configuration of every resource in an infrastructure; thus, if you want to change a resource, you can do this without IaC and by hand in the portal without much thought about restoring this at some time.

Myth 2: It's one way, either speed or quality.

Writing the IaC will take time, only at first, and it's normal to think that delivering quick code will make it low-quality code, as you can see in Figure 6-4, which is incorrect.

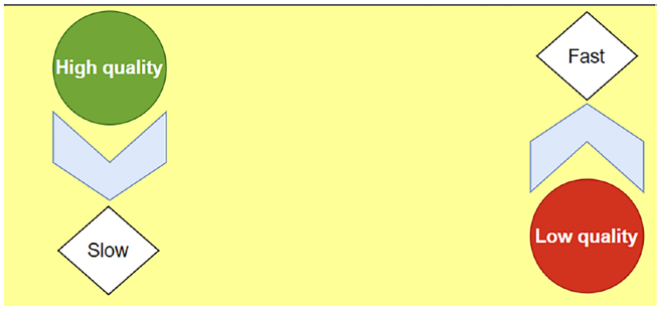

Figure 6-4. *Speed versus quality*

Teams prioritizing speed over quality produce flawed systems due to the inefficiency of their plans. It's common for established firms to feel that they've lost their "mojo" after adopting this strategy. Because of the system's complexity, even relatively straightforward modifications that used to be implemented in a matter of hours now require days or weeks.

Myth 3: A task is not repetitive so there's no need to automate it.

Ideally, after you've built a system, it will be complete. According to this theory, you don't make that many adjustments and thus automating them is not worth the effort.

In real life, when a company creates a system, it tries to enhance it as much as possible to reach a point of stability; these changes need to be documented somehow to avoid any misconfiguration in the future.

The best way to document the infrastructure is IaC; sometimes, these changes will be verified from one company to another.

- You can make changes on the network layers, either by open security port, ingress, or egress.
- You must update the servers or virtual machine with a new application upgrade or operating system.
- You've tweaked the database's configuration, which dramatically boosts its speed.
- Updating the application server is necessary to access the latest features.

One of the golden rules in the cloud is that modifying conditions creates stability.

If you can't patch your application and database regularly, then your system/application is not stable at all; if small changes cause downtime, then your system is not stable.

IaC Types

When you work with IaC, you will notice that not all tools are the same. This is the power of IaC. Each one of them has a different purpose.

Scripts

One of the most famous approaches of IaC is considered simple because the script is usually written to do only one task.

Configuration Management

Some of the most familiar IaC tools were designed to do complex tasks such as upgrade the database, install servers, apply patches, and much more. Examples of these kinds of tools are Ansible, Chef, and Puppet.

Provisioning

If you want to create a complete infrastructure, this is the category you are looking for; you can deploy any infrastructure components such as a load balancer, EC2, and much more. You could even retrieve a value deployed manually and use it inside your code; examples of these tools are Terraform and CloudFormation.

Containers and Templating

All these types of tools don't provide image templates. Assume you have an EC2 image and want to use it as an image for another infrastructure; what will you do? With the help of tools, you can generate templates and images that already have all the libraries and components an application needs to be installed; one of the best types of tools is a packer.

Tool Examples

Table 6-2 explains the differences between the tools from a high level. These are the most common tools worldwide.

Table 6-2. *DevOps tools examples and how these tools are working*

Tools	How It Works	Focus
Terraform	Agentless	Admin focused
Ansible	Agentless	Admin-focused configuration management
Jenkins	Agentless	Dev and can work for admin also, CI/CD
Puppet	Agent-based, agentless (not all the features will work)	Admin and dev focused
Chef	Agent-based can be agentless (not all the features will work)	Dev and admin focused
Salt	Agent, agentless	Admin focused

After explaining the IaC concepts, how it works, and its benefits, it's time for the technical stuff. I will divide the tools into two main categories.

- *Cloud tools*: Cloud tools will support only cloud-native solutions. These tools will allow you to deploy to different cloud providers, and some of the tools even work on specific clouds only.
- *On-premises tools*: These tools on only on-premises, allowing DevOps to configure the infrastructure components.

While infrastructure as code has many benefits, selecting an IaC solution can be complicated, because the functions of several IaC tools are similar. Many of them are free and available to the public. Some of them provide business assistance. Without firsthand experience, it's difficult to determine how to choose between them.

While it's technically accurate that you might be just as productive with any of these tools, most comparisons you see list their general features, giving the impression that you could be just as effective with any of them.

I actually use several IaC tools, but it will also work if you want to stick only to one simultaneously. I'll now cover how to how the best one (or several) for your job.

How to Choose an IaC Tool

I will compare different IaC tools, such as Terraform, Ansible, Pulumi, and CloudFormation, in this section. When you choose a tool, you'll want to consider the following factors:

- Provision versus configuration management
- Agent versus agentless
- Integration with other tools
- Mutable infrastructure versus immutable infrastructure
- General IaC language versus specifically designed language
- Paid version versus free version

Provision vs. Configuration Management

When talking about this provisioning versus configuration management, there are lots of examples. Terraform, CloudFormation, Azure Resource Manager, and Pulumi are provision tools, but also Ansible, Puppet, and Chef are configuration management tools.

IT infrastructure provisioning is a way of introducing the necessary hardware and software. Likewise, it might mean the processes involved in facilitating the access to and use of information by people and machines. Configuration follows provisioning.

Virtual machines (VMs), load balancing, databases, and so on, can all be provisioned with the help of these technologies. You can let configuration management and its tool handle the setup.

On the other hand, configuration management guarantees that hardware, software, and network infrastructure always perform as intended, where the services they deliver use the code in a manner that is both repeatable and consistent.

Simply, configuration management is not the same as provisioning.

The configuration management can still deploy the infrastructure, but it's more about controlling things such as where the application configuration should be located and where to install it.

Provision tools, Terraform, for example, provision a virtual machine called a *database instance* with specific networking requirements and instance size.


```
1. resource "aws_instance" "database-instance" {
2.   ami           = "ami-12345f4433dd1f123"
3.   instance_type = "t2.micro"
4.
5.   network_interface {
6.     network_interface_id = aws_network_interface. database-
       instance-nw.id
7.     device_index         = 0
8.   }
9.
10.  credit_specification {
11.    cpu_credits = "unlimited"
12.  }
13. }
14.
```

If you convert the previous code to configuration management, the easiest way to do this is to use a tool such as Ansible.

```
1. ---
2. - name: EC2_instance_creation
3.   hosts: localhost
4.   gather_facts: false
5.   tasks:
7.   - name: EC2_Specs
8.     block:
9.       - name: EC2_infomation
10.
11.        ec2_instance_info:
12.        register: ec2_info
13.      - name: Print info
14.       debug: var="ec2_info.instances"
15.
```

You need to complete the YAML code and create the `EC2_Block`, which will be responsible for the operating system. This is only a tiny section of the complete YAML file to provision only EC2, so usually the provision tools are much easier to use than the configuration when it's related to the infrastructure section.

```
- name: EC2_Specs
  block:
    - name: Launch ec2
      tags: chapter_IaC
      ec2:
        region: us-east-1
        key_name: Osama-PrivateKey
        group: ec2_secuirty_group
        instance_type: t2.micro
        image: ami-12345f4433dd1f123

```

Agent vs. Agentless

Some of the IaC tools require you to install agents to work correctly. Chef and Puppet, for example, require an agent. For sure, they are powerful tools that can do many things, but there is a disadvantage to these agents' tools.

Although Chef and Puppet provide varying degrees of functionality for agentless modes, these modes appear to be an afterthought. They do not cover the whole set of features of the configuration management solution. Because of this, the default setup for Chef and Puppet usually contains an agent.

- *Administration*: Having an agent is an extra job for the system administrator or site reliability engineer (SRE). This agent requires updates occasionally, and agents should be in sync with the primary node, especially if the number of servers is significant.
- *Security aspect*: It would be best if you opened an additional port to make this agent work, which sometimes is a concern to the security team.

- *Setup*: The architecture of the agent tools work can seem complicated; see Figure 6-5, which illustrates how the Chef architecture works.

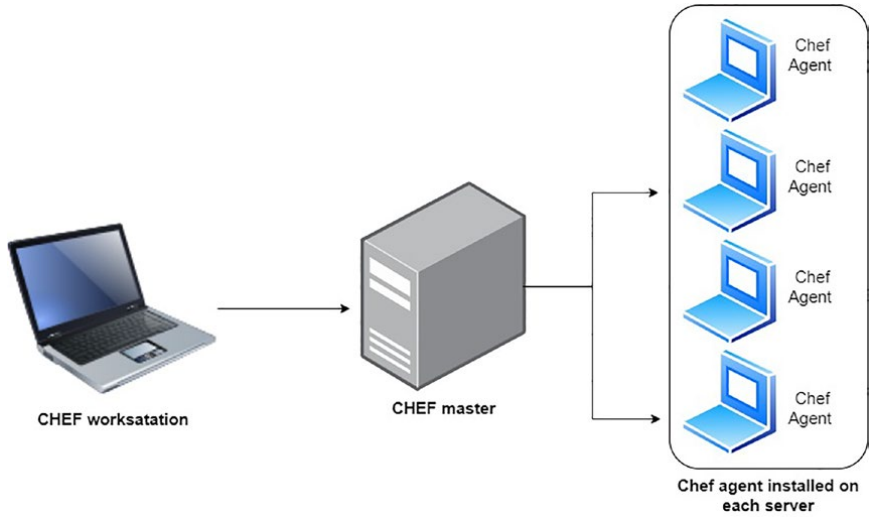

Figure 6-5. *An example of agent tools: Chef*

Integration with Other Tools

IaC is a powerful tool, but it can't do everything for you. Let's look at Terraform as an example; it can do many things and provides a complete infrastructure.

However, Terraform has limitations related to configuring the virtual machine or application, such as if you need to configure the template.

Figure 6-6 shows a different IaC use case depending on what is needed; from the bottom, Terraform can be used to provision the cloud infrastructure.

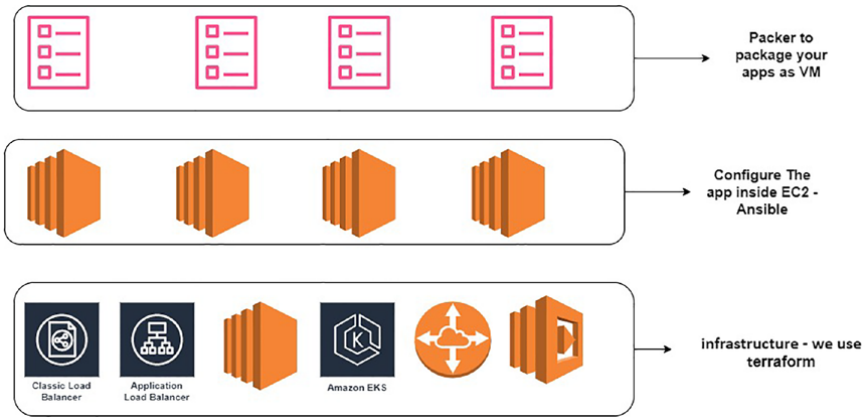

Figure 6-6. *A complete solution can be provided using different IaC tools*

Ansible can provision cloud resources, provision on-premises infrastructure, or even inject commands to the servers; moreover, a packer, which is open-source software, can replicate a machine's exact setup across various platforms.

Mutable Infrastructure vs. Immutable Infrastructure

Chef and Ansible are considered mutable tools, but what does this mean? Let's assume you want to install a Linux package using Chef. Chef will update the software on all of your servers that are in use. The changes will take effect without you having to do anything. Each server eventually develops its unique history of modifications as more and more upgrades are applied to it with time; it is *idempotent*.

Because of this, each server becomes slightly different from all the others, which can result in subtle configuration bugs that are challenging to detect and replicate. (This is the same configuration drift problem that occurs when you manage servers manually; if you have different environments, these environments will not be in sync with each other and will be hard to troubleshoot.)

On the other hand, *immutable infrastructure* refers to a method of managing service and software deployments on IT resources in which components are swapped out rather than modified in any way. Microsoft developed this method.

A perfect example of immutable infrastructure is a container because any changes to a container can be made only by creating a new version of that container.

For instance, if you want to update an image using Terraform, it will not update the image. It will just create a new one with the latest update and delete the old one. Although it is theoretically conceivable to coerce configuration management software into doing immutable deployments, this is not the approach that is often taken by such tools.

General IaC Language vs. Specifically Designed Language

Chef and Pulumi allow you to write IaC with the programming language you already know. Chef supports Ruby, but Pulumi offers support for a broad range of general-purpose programming languages (GPLs), including JavaScript, TypeScript, Python, Go, C#, Java, and more.

Moreover, there is another type called a *domain-specific language* (DSL); for example, YAML is used by Ansible and CloudFormation (which also supports JSON), and Terraform uses HCL. Puppet makes use of the Puppet language.

GPL Pros

- If you know the language, you don't need to learn anything extra.
- It has professional mature tooling because of the community and overall level of quality.
- It can be applied to the completion of virtually any programming endeavor.

DSL Pros

- It has organized and structured code.
- The code is easier to understand since it's more straightforward and less wordy.

Paid Version vs. Free Version

There are different IaC options, and which you use depends on the goal the company wants to meet. Most IaC vendors provide two options, either paid or free. For instance, the open-source version of Terraform is free to use on its own, or you can pay for HashiCorp's Terraform Cloud.

I think the free versions of Terraform, Chef, Puppet, and Ansible can be used well for production; commercial services can make these tools much better, but you can get by without them. Alternatively, without the premium service known as Pulumi Service, it is more challenging to utilize Pulumi in production.

Comparison of Tools

Table 6-3 summarizes the main differences between the IaC tools we've discussed so far.

Table 6-3. *IaC comparison based on different aspect*

Factors	Terraform	Ansible	CloudFormation	Pulumi	Chef	Puppet
Free/Paid	Both	Both	Paid	Both	Both	Both
Cloud	All	All	AWS	All	All	All
Category	Provision	Configuration Mgmt	Provision	Provision	Configuration Mgmt	Configuration Mgmt
Infra type	Immutable	Mutable	Immutable	Immutable	Mutable	Mutable
Agent	No	No	No	No	Yes	Yes
Community	Huge	Huge	Large	Small	Large	Large
Maturity	Average	Average	minimal	minimal	High	High
DSL/GPL	DSL	DSL	DSL	GPL	GPL	DSL
Procedural/ Declarative	Declarative	Procedural	Declarative	Declarative	Procedural	Declarative

Terraform

When we are talking about IaC, the first tool that comes to your mind is probably Terraform, which is an open-source IaC tool created by a company called HashiCorp. It is generally used by DevOps engineers to automate infrastructure-related activities. You can use Terraform to create cloud infrastructure, and it works with any cloud providers, including AWS, Google Cloud, Azure, and more.

With Terraform, you can provide a programmatic description of your whole network's setup. Terraform allows you to construct and manage resources in parallel

across providers, even if your servers originate from various providers such as AWS or Azure. Terraform is the glue that holds together your IT infrastructure and the universal language with which you can communicate with all your teams.

Terraform was developed using the Go programming language. The Go source code is compiled into a Terraform binary (or multiple binaries for each supported OS).

You don't need to run any additional hardware to use this program to deploy infrastructure from your laptop, a build server, or any other computer. The Terraform binary automatically performs API calls on your behalf to various cloud providers such as Amazon Web Services, Microsoft Azure, Google Cloud Platform, DigitalOcean, OpenStack, and more. Terraform can use the infrastructure these providers already have for their API servers and the authentication procedures you currently use (e.g., the API keys you already have for AWS).

You next question probably is, where does Terraform get its API call? Terraform configurations are the solution; they detail the architecture you want to build. The "code" in "infrastructure as code" refers to these settings. Let's look at how to set up Terraform; the API used here is for the AWS cloud provider.

While initializing a working directory, the Terraform CLI searches for and installs the necessary providers. It can automatically download providers from a Terraform registry or load them from a local cache or mirror.

The operation of Terraform is accomplished by constructing a graph database that gives operators insight into the relationships between resources. In addition, it produces an execution plan, showing operators the order in which Terraform's activities will be carried out in response to a setting being applied or modified.

```
1. resource "aws_s3_bucket" "example_bucket" {
2.   bucket = "terraform-code-chapter"
3. }
```

Transparent portability across cloud providers is a typical concern for Terraform users because of the wide range of cloud services it supports. In other words, if you use Terraform to specify a set of AWS resources such as servers, databases, and load balancers, can you then use the exact instructions to deploy the same resources in another cloud provider like Azure or Google Cloud?

I find this is a question that leads nowhere. However, since various cloud providers provide different kinds of infrastructure, it is impossible to implement the same infrastructure in multiple cloud providers.

The functionality, configuration, administration, security, scalability, availability, observability, and so on, of AWS's servers, load balancers, and databases differ vastly from those of Azure and Google Cloud. Terraform's approach is to standardize the language, toolset, and IaC techniques behind the scenes while letting you write provider-specific code to use the provider's peculiarities.

Figure 6-7 shows an example of writing code in Terraform to deploy AWS resources (whatever the resources are). It will not work on another cloud provider because the Terraform API will differ.

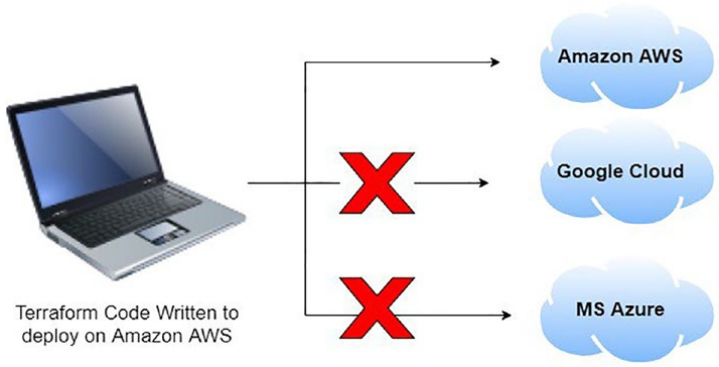

Figure 6-7. *Different Terraform code for each cloud*

The Terraform provider allows you to interact with AWS resources, as follows:

```
1. provider "aws" {
2.   region = "eu-west-1"
3. }
```

I will cover the Terraform basics with AWS. It is as an excellent option for these reasons:

- Amazon AWS was the most common provider with a 32 percent market share as I wrote this book.
- AWS services are mature and include many services that focus on your needs.
- AWS offers different versions including a free tier.

Terraform Concepts

Let's move on to the technical talk; we should start with Terraform state.

Terraform State

Each time you execute Terraform, it makes a Terraform state file that shows what infrastructure it built. Once you run Terraform, by default the name of the state file will be `terraformed.tfstate`. This file has a personalized JSON format that keeps track of the mapping between the Terraform assets in your system settings and how those resources look in the actual world.

The following example is a section from the `terraform.tfstate` file. You can use the following JSON every time you update the code. Terraform will read from that file and check/review these resources to ensure there are no duplicates.

```
1. {
2. "version": 4,
3.   "terraform_version": "1.2.0",
4. "serial": 222,
5. "lineage": "b697390d-f0a4-14dc-7df4-95b5b2cff75f",
6. "outputs": {},
7. "resources": [
8.     {
9. "module": "module.eks",
10. "mode": "data",
11. "type": "aws_availability_zones",
12. "name": "available",
13.       "provider": "provider[\"registry.terraform.io/hashicorp/aws\"]",
14. "instances": [
15.         {
16. "schema_version": 0,
17. "attributes": {
18. "all_availability_zones": null,
19. "exclude_names": null,
20. "exclude_zone_ids": null,
21. "filter": null,
22. "group_names": [
23. "eu-west-1"
```

```
            ],
"id": "eu-west-1",
"names": [
"eu-west-1a",
"eu-west-1b",
"eu-west-1c"
            ],
"state": null,
"timeouts": null,
"zone_ids": [
              "euw1-az2",
              "euw1-az3",
              "euw1-az1"
            ]
          },
"sensitive_attributes": []
        }
      ]
    },
```

When talking about the state, best practices need to be implemented; if you are using the state file for a private project, it will be saved on your local computer within the same directory as Terraform, which is acceptable. But if the project is for a company, it will not work to be local since a couple of issues will happen.

- Shared storage for state files
  - Each team member needs access to the duplicate Terraform state files because each one will need to update the infrastructure, and therefore, these changes should be saved under one state file; otherwise, it will be a messy configuration.
- State file lock
  - Without locking, data inconsistencies, lost updates, and corrupted state files might arise if more than one team member uses Terraform.

- Separate state file
  - A company will have different environments, such as production, QA, UAT, and more; when you update these environments, it is much better to keep the state file separated and not mixed.

As a remote back end, Amazon S3, Amazon's managed file store, is usually your best bet when using Terraform with AWS.

- Since it's a managed service, you don't have to set up and manage any extra hardware.
- It's made to last 99.999999999 percent of the time and be available 99.99 percent of the time, so you don't have to worry much about data loss or outages.
- You can save the state there and be supported by DynamoDB, but why?

  To secure a system and ensure its integrity, the `terraform.tfstate` file is saved in a bucket on Amazon S3. When you or a co-worker uses the `terraform plan` command, Terraform retrieves the file from that location and compares it to your Terraform settings to see if any adjustments need to be made.

  At the same time, the DynamoDB table locks its current state to prevent data corruption, lost information, and conflicts if any co-workers attempt to modify the infrastructure simultaneously.
- It enables versioning very quickly, saving the state in each change you make.
- AWS provides a different tier for S3 depending on the usage.

To use Amazon S3 for remote state storage, you must first make an S3 bucket, and you can do this in two ways, either with the console, with AWS CLI, or with some Terraform code (which should be run first).

If you choose, for example, to create the bucket using Terraform code, you should run that code first to ensure the bucket is created; otherwise, an error will be generated that the bucket will not exist.

Let's do that. The following are the best practices for creating a Terraform structure (see Figure 6-8):

- `Backend.tf` is responsible for defining the state file remote back end.
- `Main.tf` contains your module's main set of settings; if the file is not created, the state file will be on your local machine.
- `Terraform.tfvars` is used to define the variable, especially if you have a different environment so that you can call the file, for example, `prod.tfvars`, `qa.tfvars`, and so on. This is usually used to define the primary variable that can be changed all the time, and when this file used, this file has precedence over the variable file.
- `Variable.tf` is used as parameters to let us change how our deployments work by putting in values at runtime. In the `main.tf` configuration file, you can set up input variables for Terraform.
- `Version.tf` (optional) is used to define what Terraform version this code will work with. If not mentioned, it will assume you are using the latest version.

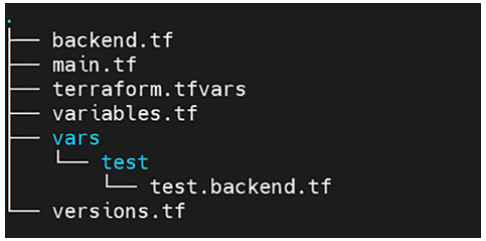

Figure 6-8. *Terraform folder structure*

Let's create our first file, which is `version.tf` or `provider.tf`.

```
terraform {
    required_version = ">= 0.13.1"

    required_providers {
      aws = {
        source  = "hashicorp/aws"
```



```
        version = ">= 3.73"
      }
    }
}
```

Inside `main.tf`, let's create the bucket. The code will do the following:

- Set up code changes on the S3 bucket so that each time a file in the bucket is changed, a new version of that file is created. The configuration lets you see previous models of the file and go back to them at whatever time, which can be helpful if anything goes wrong.
- Server-side encryption should be turned on for all information published to this S3 bucket. This ensures that your state files and any classified information they hold are always encrypted on an S3 disk.
- Block everyone from getting into the S3 bucket. S3 buckets are private by default.

```
 resource "aws_s3_bucket" "terraform_state_file" {
   bucket = "terraform-state-file-book-aws"

    # Prevent accidental deletion of this S3 bucket
    lifecycle {
      prevent_destroy = true
    }

  # Enable bucket versioning.
  resource "aws_s3_bucket_versioning" "bucket_version" {
    bucket = aws_s3_bucket.terraform_state_file
    versioning_configuration {
      status = "Enabled"
    }
 }

# Enable encryption
resource "aws_s3_bucket_server_side_encryption_configuration" "bucket-
encrypt" {
  bucket = aws_s3_bucket. terraform_state_file
```

```
20.
21.     rule {
22.       apply_server_side_encryption_by_default {
23.         sse_algorithm = "AES256"
24.       }
25.     }
26.
27. # Block public access
28. resource "aws_s3_bucket_public_access_block" "bucket_public_access" {
29.   bucket                  = aws_s3_bucket.terraform_state_file
30.   block_public_acls       = true
31.   block_public_policy     = true
32.   ignore_public_acls      = true
33.   restrict_public_buckets = true
34. }
35.
```

That's it. The Terraform state file will be saved into the bucket as the remote back end. However, what if you want to ensure the lock issue is solved? Then you need to use another AWS service, called DynamoDB.

Amazon's key-value store, DynamoDB, has several copies of its data. It provides the ability to do highly consistent readings and conditional writes, which are the only two operations a distributed lock system requires. It is also inexpensive and fully managed, so you do not need to worry about the underlying infrastructure.

To utilize DynamoDB with Terraform's locking functionality, you must construct a DynamoDB table with the primary key named LockID.

```
1. resource "aws_dynamodb_table" "terraform_locks_file" {
2.   name         = "terraform-locks_table"
3.   billing_mode = "PROVISIONED"
4.   hash_key     = "LockID"
5.
6.   attribute {
7.     name = "LockID"
8.     type = "S"
9.   }
10. }
```


- `Billing_mode` (optional): How you are paid for read and write throughput and how capacity is managed are under your control, and there are two values: `PROVISION`, which is the default, or `PAY_PER_REQUEST`.
- `Type` (required): Attribute type. Valid values are `S` (string), `N` (number), and `B` (binary).

Now we create most of the resources needed for the bucket and DymanoDB; in case you want that, the next step is to use the Terraform commands. This is not complicated; you will use three commands here, but there are more if you want to do more.

- `Terraform init`

  After creating a fresh Terraform template or cloning an old one from version control, the first step is to execute this command. It is recommended that you do it immediately.

  This command will conduct multiple initialization stages to get the current working directory ready for usage with Terraform. These procedures can be found in the Terraform documentation. In most circumstances, it is not essential to worry about these specific stages, and more information on them can be found in the following sections.

  It is never dangerous to use this command more than once to bring the working directory up-to-date with modifications made to the configuration. This command will never erase your current settings or state, even though future executions may produce errors.

- `Terraform plan`

  This command generates an implementation strategy that previews the modifications Terraform will apply to your infrastructure.

  This command is optional as well, but it's important so the DevOps engineer can review what will be deployed or destroyed.

- `Terraform apply`

  The command will apply and deploy the changes of the new infrastructure; once you run the order, you need to confirm again by typing `yes`.

- `Terraform destroy`

  This eradicates all distant objects handled by a particular Terraform setup or state.

To run the previous code, deploy using the `terraform apply` command, and then execute `terraform init` to get the provider code. After everything has been deployed, you will have an S3 bucket and a DynamoDB table, but the state of your Terraform configuration will continue to be kept locally. You must add a back-end setting to your Terraform code to set up Terraform to save the state in your S3 bucket (while also encrypting and locking the information). Because this is a configuration for Terraform itself, it must be included inside a `terraform` block and must adhere to the following syntax:

```
1. terraform {
2.   backend "s3" {
3.     bucket = "terraform-state-file-book-aws "
4.     key    = "chapter6/terraform-iac/terraform.tfstate"
5.     region = "eu-west-1"
6.
7. # Replace this with your DynamoDB table name!
8.     dynamodb_table = "terraform-locks_table"
9.     encrypt        = true
10.
11.   }
12. }
```

- `Bucket`: This is the label of the S3 bucket that will be used. Make sure you change the ID of the S3 bucket you established previously in the process.
- `Key`: This is the file's location inside the S3 bucket where the Terraform state file should be saved when created.

- `Region`: This is the region inside AWS where the S3 bucket is stored.
- `dynamodb_table`: This is the DynamoDB table that will be used for the locking operation.
- `Encrypt`: By setting this to `true`, you can be sure that the Terraform state you save in S3 will be secured on disk. Since we previously made encryption the default setting for the S3 bucket, this serves as a secondary layer of protection to guarantee that the information is always protected.

The `terraform init` command is responsible for downloading the provider and setting up the back end, as shown in Figure 6-9.

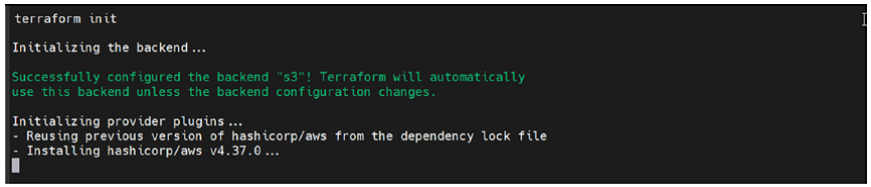

Figure 6-9. *Running the terraform init command to install the providers*

This command will take time, depending on how many providers you are using inside your code, but you should be able to see a message like the one in Figure 6-10.

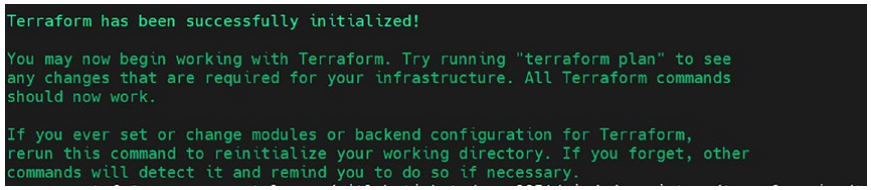

Figure 6-10. *Terraform init output*

If you check from the console, you will see the Terraform screen shown in Figure 6-11.

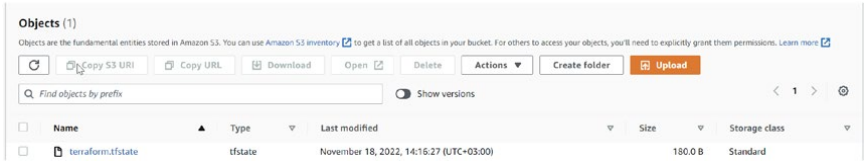

Figure 6-11. *Remote back-end Terraform state*

Sometimes, you need to show some output for the resources, such as database name, public IP address, or private IP address. Terraform allows you to do that with an output, but it's better to do that and configure a file called `output.tf`.

For example, to show the name of the S3 bucket and DynamoDB that was created earlier, I need a file called `output.tf`.

```
output "s3_bucket_arn" {
  value       = aws_s3_bucket.terraform_locks_file.arn
  description = "The ARN of the S3 bucket"
}

output "dynamodb_table_name" {
  value       = aws_dynamodb_table.terraform_locks_file.name
  description = "The name of the DynamoDB table"
}

```

It's hard to cover everything about Terraform in one section, but I gave you the basics of how it works. I also created a different project for Terraform using another cloud provider for free and uploaded it to GitHub (`https://Github.com/Osamaoracle`). Plus, Chapter 11 includes some complete projects.

Let's look at a straightforward Terraform code example. I share the directory structure in Figure 6-12, which will allow you to understand how it will look at the end. The figure shows how Terraform will be set up.

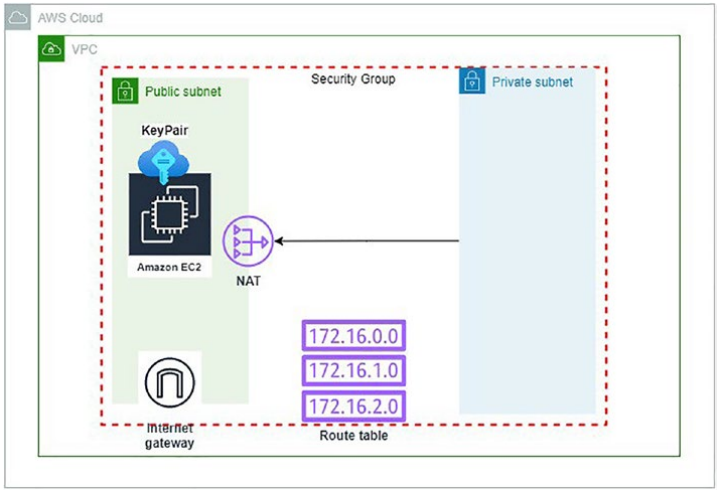

Figure 6-12. *Simple Terraform example to set up AWS resources*

- VPC refers to the virtual private network in AWS.
- VPC subnet, in our case, public subnet that will be expose to the Internet.
- We are creating an Internet gateway associated with the VPC.
- There is a route table inside VPC with a route that directs Internet-bound traffic to the IGW.
- Route table association with our subnet to determine where network traffic from your subnet or gateway is directed.
- Security group to provide protection at the port and protocol access levels.
- Key pair used to access the EC2.
- EC2 instance deployed inside the public subnet.

Let's look at a real-world example of Terraform to show how the structure folder works; Figure 6-13 shows the Terraform folder structure.

```
.
|-- README.md
|-- main.tf
|-- output.tf
|-- provider.tf
`-- variables.tf

0 directories, 5 files
```

Figure 6-13. *Terraform file structure*

The provider.tf

This file will allow Terraform to determine which cloud provider will be used.

```
1. provider "aws" {
2.  access_key = var.access_key
3.  secret_key = var.secret_key
4.  region = var.region
5. }
6.
```

Variable.tf

In this file, we define the variable used inside our Terraform code to make the code more dynamic.

```
 1. variable "access_key" {
 2.   default = "ACCESS_KEY_HERE"
 3. }
 4. variable "secret_key" {
 5.   default = "SECRET_KEY_HERE"
 6. }
 7. variable "region" {
 8.   default = "us-east-2"
 9. }
10. variable "cidr_vpc" {
11.   description = "CIDR block for the VPC"
12.   default     = "10.1.0.0/16"
13. }
```



```
14. variable "cidr_subnet" {
15.   description = "CIDR block for the subnet"
16.   default     = "10.1.0.0/24"
17. }
18. variable "availability_zone" {
19.   description = "availability zone to create subnet"
20.   default     = "us-east-2a"
21. }
22. variable "public_key_location" {
23.   description = "Public key path"
24.   default     = "~/.ssh/id_rsa.pub"
25. }
26. variable "instance_ami" {
27.   description = "EC2 AMI depends on the region"
28.   default     = "ami-0cf31d971a3ca20d6"
29. }
30. variable "instance_type" {
31.   description = "EC2 Instance type"
32.   default     = "t2.micro"
33. }
34. variable "environment_tag" {
35.   description = "Resource tags"
36.   default     = "Production"
37. }
```

Main.tf

After creating the variable needed to be used and the provider, this file will call all the necessary resources to be configured inside the AWS console.

```
1. resource "aws_vpc" "vpc" {
2.   cidr_block = var.cidr_vpc
3.   enable_dns_support   = true
4.   enable_dns_hostnames = true
5.   tags = {
```

```
    "Environment" = var.environment_tag
  }
}

resource "aws_internet_gateway" "igw" {
  vpc_id = aws_vpc.vpc.id
  tags = {
    "Environment" = var.environment_tag
  }
}

resource "aws_subnet" "subnet_public" {
  vpc_id = aws_vpc.vpc.id
  cidr_block = var.cidr_subnet
  map_public_ip_on_launch = "true"
  availability_zone = var.availability_zone
  tags = {
    "Environment" = var.environment_tag
  }
}

resource "aws_route_table" "rt_public" {
  vpc_id = aws_vpc.vpc.id

  route {
      cidr_block = "0.0.0.0/0"
      gateway_id = aws_internet_gateway.igw.id
  }

  tags = {
    "Environment" = var.environment_tag
  }
}

resource "aws_route_table_association" "rt_subnet_public" {
```



```
  subnet_id      = aws_subnet.subnet_public.id
  route_table_id = aws_route_table.rt_public.id
}

resource "aws_security_group" "sg_ssh" {
  name = "sg_ssh"
  vpc_id = aws_vpc.vpc.id

  # SSH access from the VPC
  ingress {
      from_port   = 22
      to_port     = 22
      protocol    = "tcp"
      cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
    "Environment" = var.environment_tag
  }
}

resource "aws_key_pair" "ec2ssh" {
  key_name = "publicKey"
  public_key = file(var.public_key_location)
}

resource "aws_instance" "test_Instance" {
  ami           = var.instance_ami
```

```
76.   instance_type = var.instance_type
77.   subnet_id = aws_subnet.subnet_public.id
78.   vpc_security_group_ids = [aws_security_group.sg_ssh.id]
79.   key_name = aws_key_pair.ec2ssh.key_name
80.
81.   tags = {
82.          "Environment" = var.environment_tag
83.   }
84. }
```

Output.tf

The last file in our folder structure is `output.tf`; this file is responsible for showing the needed output for the user, such as the public IP address, DNS, etc.

```
1. output "vpc_id" {
2.   value = "${aws_vpc.vpc.id}"
3. }
4. output "public_subnet" {
5.   value = ["${aws_subnet.subnet_public.id}"]
6. }
7. output "public_rt_ids" {
8.   value = ["${aws_route_table.rt_public.id}"]
9. }
10. output "public_instance_ip" {
11.   value = ["${aws_instance.test_Instance.public_ip}"]
12. }
```


Figure 6-14 shows the command `terraform init` output.

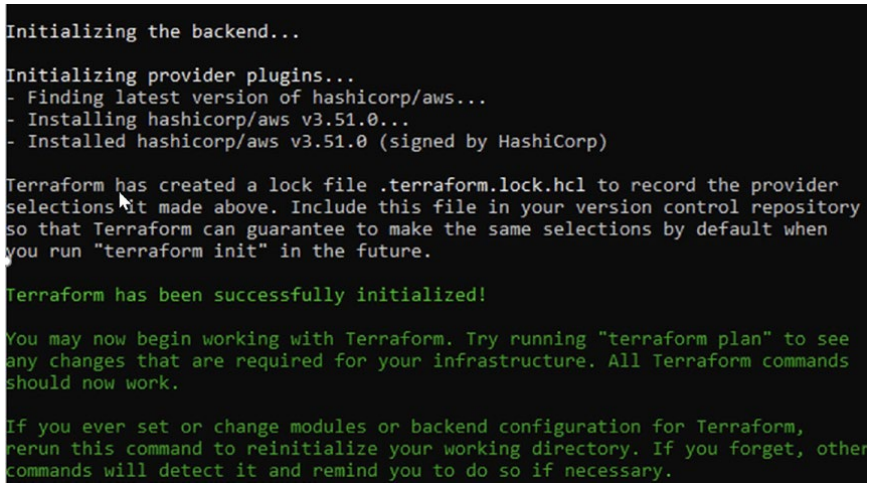

Figure 6-14. *The terraform init output*

After that, check what Terraform provision is on the AWS cloud. This is considered a critical command every time you deploy or change the configuration Terraform plan; see Figure 6-15.

```
# aws_subnet.publicsubnets will be created
+ resource "aws_subnet" "publicsubnets" {
    + arn                                = (known after apply)
    + assign_ipv6_address_on_creation = false
    + availability_zone                  = (known after apply)
    + availability_zone_id               = (known after apply)
    + cidr_block                         = "10.0.0.128/26"
    + id                                 = (known after apply)
    + ipv6_cidr_block_association_id  = (known after apply)
    + map_public_ip_on_launch         = false
    + owner_id                           = (known after apply)
    + tags_all                           = (known after apply)
    + vpc_id                             = (known after apply)
  }

# aws_vpc.Main will be created
+ resource "aws_vpc" "Main" {
    + arn                                = (known after apply)
    + assign_generated_ipv6_cidr_block = false
    + cidr_block                         = "10.0.0.0/24"
    + default_network_acl_id             = (known after apply)
    + default_route_table_id             = (known after apply)
    + default_security_group_id          = (known after apply)
    + dhcp_options_id                    = (known after apply)
    + enable_classiclink                 = (known after apply)
    + enable_classiclink_dns_support    = (known after apply)
    + enable_dns_hostnames               = (known after apply)
    + enable_dns_support                 = true
    + id                                 = (known after apply)
    + instance_tenancy                   = "default"
    + ipv6_association_id                = (known after apply)
    + ipv6_cidr_block                    = (known after apply)
    + main_route_table_id                = (known after apply)
    + owner_id                           = (known after apply)
    + tags_all                           = (known after apply)
  }
```

Figure 6-15. *Terraform plan output*

Terraform Module

A Terraform module can be anything that consists of Terraform configuration files stored in a subdirectory; in other words, a module is a pretty straightforward concept. All of the configurations you have created up to this point have, in a technical sense, been modules; however, they are not especially interesting since you deployed them

directly. If you apply these configuration on a module, that module is referred to as a *root module*. It would be best if you constructed a reusable module, which is a module that is designed to be utilized inside other modules, to get a proper understanding of what the modules are capable of doing.

Figure 6-16 shows the architecture that will create the following resources using Terraform:

- VPC
  - Subnets
  - Internet gateway
  - NAT gateway
- EC2 instance
- RDS databases
- Security group

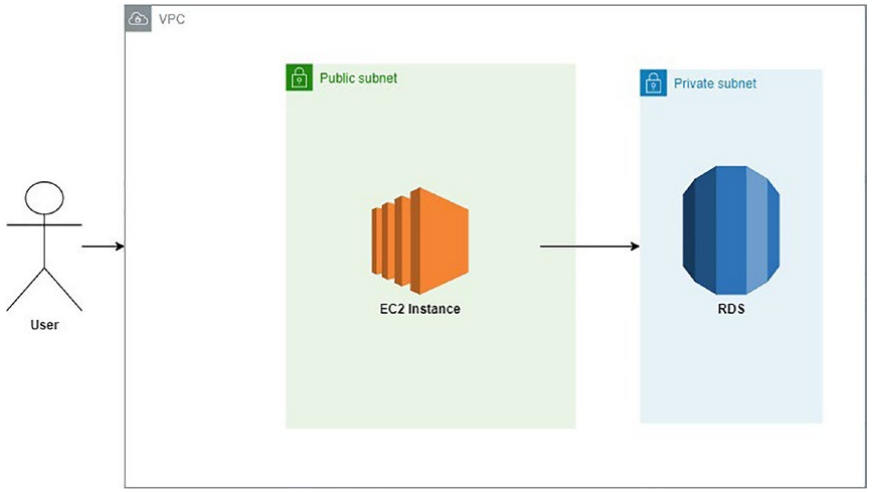

Figure 6-16. *Terraform example architecture*

Figure 6-17 shows how the final folder structure will look.

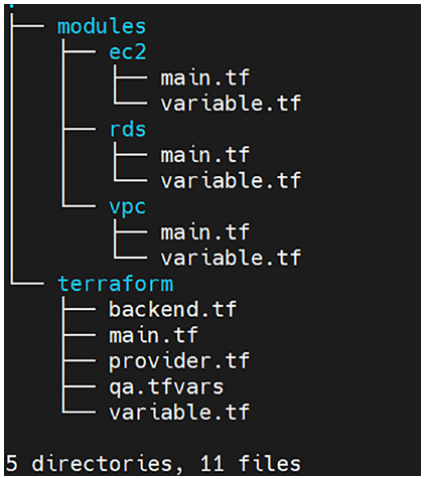

Figure 6-17. *Terraform example folder structure*

VPC has only one route table that links the Internet gateway to the public subnet hosting the EC2 instance.

A private subnet is organized into a single subnet group, each hosting a single RDS server.

VPC Code

Create a folder called `vpc.tf` located inside the module directory.

```
# VPC
resource "aws_vpc" "iac-chapter" {
  cidr_block = var.vpc_cidr

  enable_dns_support   = var.enable_dns_support
  enable_dns_hostnames = var.enable_dns_hostnames
}
# Internet Gateway
resource "aws_internet_gateway" "iac-chapter" {
  vpc_id = aws_vpc.iac-chapter.id
}
```



```

# Route Table
resource "aws_route_table" "iac-chapter" {
  vpc_id = aws_vpc.iac-chapter.id

  dynamic "route" {
    for_each = var.route

    content {
      cidr_block     = route.value.cidr_block
      gateway_id     = route.value.gateway_id
      instance_id    = route.value.instance_id
      nat_gateway_id = route.value.nat_gateway_id
    }
  }
}

# associate route table with subnet.
resource "aws_route_table_association" "iac-chapter" {
  count          = length(var.subnet_ids)

  subnet_id      = element(var.subnet_ids, count.index)
  route_table_id = aws_route_table.iac-chapter.id
}

```

EC2

To initiate the creation of the EC2 instance, all that is required is to configure the machine we want and position it inside the subnet containing our route table.

- An instance of AWS EC2
- In conjunction with an elastic IP address being attached to that instance
- A public-private key pair, also known as a PEM key, to use when connecting to the model via SSH

```
locals {
  resource_name_prefix = "${var.namespace}-${var.resource_tag_name}"
}

resource "aws_instance" "iac-chapter" {
  ami                         = var.ami
  instance_type               = var.instance_type
  user_data                   = var.user_data
  subnet_id                   = var.subnet_id
  associate_public_ip_address = var.associate_public_ip_address
  key_name                    = aws_key_pair.iac-chapter.key_name
  vpc_security_group_ids      = var.vpc_security_group_ids

  iam_instance_profile = var.iam_instance_profile
}

resource "aws_eip" "iac-chapter" {
  vpc      = true
  instance = aws_instance.iac-chapter.id
}

resource "tls_private_key" "iac-chapter" {
  algorithm = "RSA"
  rsa_bits  = 4096
}

resource "aws_key_pair" "iac-chapter" {
  key_name   = var.key_name
  public_key = tls_private_key.iac-chapter.public_key_openssh
}

```


EC2 Security Group

A security group acts like a firewall, controlling the network that goes in (ingress) and out (egress) of your network.

Ingress can describe either the act of coming or an entrance. The term *egress* represents either the act of leaving or a physical door or window that allows one to do so.

We permit traffic to enter our network via ports 22 (SSH), 80 (HTTP), and 443 (HTTPS), and we let all traffic leave our network through all ports. To make this even more secure, you should profile your application's ports for outgoing traffic.

```
resource "aws_security_group" "ec2" {
  name = "${local.resource_name_prefix}-ec2-sg"

  description = "EC2 security group (terraform-managed)"
  vpc_id      = module.vpc.id

  ingress {
    from_port   = var.rds_port
    to_port     = var.rds_port
    protocol    = "tcp"
    description = "MySQL"
    cidr_blocks = local.rds_cidr_blocks
  }

  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    description = "Telnet"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
```

```
    description = "HTTP"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    description = "HTTPS"
    cidr_blocks = ["0.0.0.0/0"]
  }

  # Allow all outbound traffic.
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

```

RDS

Now, it's time to set up the database; in my case, I chose MySQL.

```
locals {
  resource_name_prefix = "${var.namespace}-${var.resource_tag_name}"
}

resource "aws_db_subnet_group" "iac-chapter" {
  name       = "${local.resource_name_prefix}-${var.identifier}-
  subnet-group"
  subnet_ids = var.subnet_ids
}

```



```
resource "aws_db_instance" "iac-chapter" {
  identifier = "${local.resource_name_prefix}-${var.identifier}"

  allocated_storage       = var.allocated_storage
  backup_retention_period = var.backup_retention_period
  backup_window           = var.backup_window
  maintenance_window      = var.maintenance_window
  db_subnet_group_name    = aws_db_subnet_group.iac-chapter.id
  engine                  = var.engine
  engine_version          = var.engine_version
  instance_class          = var.instance_class
  multi_az                = var.multi_az
  name                    = var.name
  username                = var.username
  password                = var.password
  port                    = var.port
  publicly_accessible     = var.publicly_accessible
  storage_encrypted       = var.storage_encrypted
  storage_type            = var.storage_type

  vpc_security_group_ids = ["${aws_security_group.iac-chapter.id}"]

  allow_major_version_upgrade = var.allow_major_version_upgrade
  auto_minor_version_upgrade  = var.auto_minor_version_upgrade

  final_snapshot_identifier = var.final_snapshot_identifier
  snapshot_identifier       = var.snapshot_identifier
  skip_final_snapshot       = var.skip_final_snapshot

  performance_insights_enabled = var.performance_insights_enabled
}

```

As you can see from the previous code, different variables need to be filled in by DevOps, such as the username, password, instance type, name of the instance, and storage.

To make our job much easier in the future, we will create `qa.tfvars`. Depending on the environment name, which contains the primary variable used by the RDS, the file will allow DevOps to change database settings easily.

```
# RDS
rds_identifier        = "mysql"
rds_engine            = "mysql"
rds_engine_version    = "8.0.15"
rds_instance_class    = "db.t2.micro"
rds_allocated_storage = 10
rds_storage_encrypted = false
rds_name              = ""
rds_username          = "admin"
rds_port                    = 3306
rds_maintenance_window      = "Sun:05:00-Fri:06:00"
rds_backup_window           = "12:46-13:16"
rds_backup_retention_period = 1
rds_publicly_accessible     = false
rds_final_snapshot_identifier = "db-snapshot"
rds_snapshot_identifier       = null
rds_performance_insights_enabled  = true
```

RDS Security Group

A security group opens the MySQL port for incoming connections.

```
resource "aws_security_group" "db-sg" {
  name = "${local.resource_name_prefix}-rds-sg"

  description = "RDS (terraform-managed)"
  vpc_id      = var.rds_vpc_id

  ingress {
    from_port   = var.port
    to_port     = var.port
    protocol    = "tcp"
```



```
    cidr_blocks = var.sg_ingress_cidr_block
  }

  # Allow all outbound traffic.
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = var.sg_egress_cidr_block
  }
}
```

Now, we create the module folders, which include all our main code, but we need to call this code. We will call the module inside another directory, as mentioned in Figure 6-13.

`main.tf` will look like the following:

```
module "vpc" {
  source = "../../modules/vpc"

  resource_tag_name = var.resource_tag_name
  namespace         = var.namespace
  region            = var.region

  vpc_cidr = "10.0.0.0/16"

  route = [
    {
      cidr_block     = "0.0.0.0/0"
      gateway_id     = module.vpc.gateway_id
      instance_id    = null
      nat_gateway_id = null
    }
  ]

  subnet_ids = module.subnet_ec2.ids
}
```

```
21.
22.
23. module "ec2" {
24.   source = "../../modules/ec2"
25.
26.   resource_tag_name = var.resource_tag_name
27.   namespace         = var.namespace
28.   region            = var.region
29.
30.   ami           = "ami-1212sdasdas" # Choose ami depends on the region
31.   key_name      = "${local.resource_name_prefix}-ec2-key"
32.   instance_type = var.instance_type
33.   subnet_id     = module.subnet_ec2.ids[0]
34.
35.   vpc_security_group_ids = [aws_security_group.ec2.id]
36.
37.   vpc_id = module.vpc.id
38. }
```

Terraform Tips and Tricks

As with any other language, Terraform supplies some basic building blocks to help the DevOps engineer write the best code for the IaC; we will discuss the following in this section:

- Loops
- Conditionals
- Zero-downtime deployment

Loops

Terraform provides several distinct looping constructions, each of which is designed to be used in a somewhat unique circumstance.

- `Count`: This parameter will be used to loop the resource; for example, if you want to create 10 EC2 instances, it does not make sense to repeat the code 10 times, so we use the `count` parameter to avoid this.
- `For_each`: This can be used in the same way as shown earlier.
- `For`: This is usually used to loop with the list and map variable.

I will give a short example to show the power of Terraform loops.

```
1. resource "aws_instance" "web-ec2" {
2.   count = 2 # generate two similar EC2 instances
3.   ami           = "ami-134324321"
4.   instance_type = "t2.medium"
5.   tags = {
6.     Name = "web-ec2-${count.index}"
7.     Owner = "Osama"
8.   }
9. }
```

This will create two similar EC2 instances, which for some resources can be an issue like the following:

```
1. resource "aws_iam_user" "count_example" {
2.   count = 2
3.   name  = "Osama" }
4.
```

All three users of IAM would have the same name, which would result in an error since the usernames are supposed to be different.

To solve this, we need to use an index, which will create a unique IAM like the following:

```
1. resource "aws_iam_user" " count_example " {
2.   count = 2
3.   name  = "Osama.${count.index}"
4. }
```

For_each works in a different way, which is usually used by mapping and listing variables, and if you don't do that, the error will be clear like the following:

```
The given "for_each" argument value is unsuitable: the "for_each" argument
must be a map, or set of strings, and you have provided a value of
type tuple.
```

To use the previous example with for_each, I need to define an array like the one shown here:

```
1. locals {
2.   IAM_USER_NAME = {
3.     "Osama"           = "Chapter_6"
4.     "Amazon"          = "AWS"
5.     "Test"            = "working"
6.   }
7. }
```

After that, I can create this as follows:

```
1. resource "aws_iam_user" "examples" {
2.   for_each = local.IAM_USER_NAME
3.   triggers = {
4.     name  = each.key
5.     Middle = each.value
6.   }
7. }
```

There are lots of uses for the loop conditions in Terraform. Still, to explain it entirely, I probably need another chapter to do that, so I tried to show the power of looping in Terraform and how to take advantage of it to make your code follow best practices and make it easy to understand.

Conditionals

Terraform has several distinct options for making loops. It also provides many specific opportunities for performing conditionals; each option is designed for use in a somewhat different setting.

- Count parameter: Can be used for conditional statements
- For_each: Can be used for conditional statements and within the inline blocks
- If statement: The conditional statement

Let's assume I want to enable EC2 autoscaling, but depending on the situation and variable being set, this can be done by using a conditional statement like the following example.

In the first step, you need to define the Boolean variable.

```
1. variable "autoscaling_example" {
2.   description = "If set to true, enable auto-scaling"
3.   type        = bool
4. }
```

Inside the Terraform code, my code will look like the following one:

```
 1. resource "aws_autoscaling_schedule" "scaling-ec2-out" {
 2.   count = var.autoscaling_example ? 1 : 0
 3.
 4.   scheduled_action_name  = "${var.cluster_name}-scaling-ec2"
 5.   min_size               = 2
 6.   max_size               = 5
 7.   desired_capacity       = 5
 8.   recurrence             = "0 5 * * *"
 9.   autoscaling_group_name = aws_autoscaling_group.example.name
10. }
11.
12. resource "aws_autoscaling_schedule" "scaling-ec2-in" {
13.   count = var.enable_autoscaling ? 1 : 0
14.
15.   scheduled_action_name  = "${var.cluster_name}-scaling-ec2-in"
16.   min_size               = 2
17.   max_size               = 5
18.   desired_capacity       = 2
19.   recurrence             = "0 23 * * *"
```

```
20.   autoscaling_group_name = aws_autoscaling_group.example.name
21. }
22.
```

What is happening in the example? To explain, the `count` parameter for each AWS autoscaling schedule resource will have a value of 1 assigned to it if `var.autoscaling_example` is configured to be true.

This will result in the creation of one instance of each resource. If `var.autoscaling_example` is false, the `count` parameter for each AWS autoscaling schedule resource will be set to 0, meaning that none will be produced.

AWS CloudFormation

Each cloud provider has its own built-in IaC, allowing the customer to use it if they want to deploy it to that cloud and try to make it as easy as possible.

CloudFormation is a service offered by Amazon Web Services that helps simplify constructing AWS infrastructure using template files. AWS supplies CloudFormation; you can automate the setup of workloads run on the most common AWS services using CloudFormation.

The lack of code for CloudFormation templates written in YAML is an understatement. They usually include chunks of complicated programs as well as JSON specifications. The idea of having a template is moot since they are often encoded via string processing instructions.

To understand CloudFormation more, let's see the advantage and disadvantages, starting with the advantages.

- A different template is already available from AWS and can be used.
- It is easy to use with Amazon AWS.
- If you build the AWS infrastructure manual, a tool can convert the infrastructure to CF templates.
- It can be integrated with CI/CD.
- YAML is widely used, so it's always good to learn it.

Disadvantages

The following are the disadvantages:

- There is a lack of instructions on maintaining a clean and clear code base for CloudFormation.
- It's not the best tool to use to avoid security threats.
- The CloudFormation community is not robust.
- It's hard to troubleshoot the broken code, especially if it's a lot of code.
- Sometimes the stack will get stuck without displaying an error, and it will continue to be stuck until it fails.
- It might be a long process to generate and update content. Sometimes the deletion operation fails, and you must do it by hand.

In my case, I prefer YAML for a different reason; as DevOps engineers, YAML is widely used, and we learn it technically by learning another tool implicitly such as Ansible or a Kubernetes manifest.

To understand it more, let's take a look at the following example, which is creating a VPC:

- Route table
- Two private subnets

```
 1. Parameters:
 2.   Tag:
 3.     Type: String
 4.
 5. Resources:
 6.   VPC:
 7.     Type: "AWS::EC2::VPC"
 8.     Properties:
 9.       CidrBlock: "10.0.0.0/16"
10.       Tags:
11.       - Key: "Name"
12.         Value: !Ref "Tag"
```

```
13.
14.   privatesubnet1:
15.     Type: "AWS::EC2::Subnet"
16.     Properties:
17.       AvailabilityZone: !Select
18.         - 0
19.         - !GetAZs
20.           Ref: 'AWS::Region'
21.       VpcId: !Ref "VPC"
22.       CidrBlock: "10.0.0.0/24"
23.
24.   privatesubnet2:
25.     Type: "AWS::EC2::Subnet"
26.     Properties:
27.       AvailabilityZone: !Select
28.         - 1
29.         - !GetAZs
30.           Ref: 'AWS::Region'
31.       VpcId: !Ref "VPC"
32.       CidrBlock: "10.0.1.0/24"
33.
34.   RouteTable:
35.     Type: "AWS::EC2::RouteTable"
36.     Properties:
37.       VpcId: !Ref "VPC"
38.
39. Outputs:
40.   VpcId:
41.     Description: The VPC ID
42.     Value: !Ref VPC
```


The previous code is a sample of CloudFormation; the VPC name will depend on the tag, as you can see from the following section:

```
1. Parameters:
2.   Tag:
3.     Type: String
```

We have two different CICR sections, which will create two private subnets. The first will be 10.0.0.0/24, and the second will be 10.0.1.0/24.

Finally, to connect these to subnets, we need to define the routing table, which allows connectivity between them.

```
1.   RouteTable:
2.     Type: "AWS::EC2::RouteTable"
3.     Properties:
4.       VpcId: !Ref "VPC"
```

The last section will show the VPC ID after creation; another example of CloudFormation will show you how to create the EC2 instance. Notice that if the code gets longer, it will be hard to troubleshoot if you have an issue.

```
 1. ---
 2. AWSTemplateFormatVersion: '2022-12-16'
 3. Description: A simple example for EC2.
 4. Parameters:
 5.   VpcId:
 6.     Type: String
 7.   SubnetId:
 8.     Type: String
 9.   InstanceName:
10.     Type: String
11.   allowsshcidr:
12.     Type: String
13.     Description: additional security layer to allow certain IP
        for ssh.
14.     MinLength: '7'
15.     MaxLength: '20'
```

```
    AllowedPattern: "(\\d{1,3})\\.(\\d{1,3})\\.(\\d{1,3})\\.
    (\\d{1,3})/(\\d{1,2})"
  InstanceType:
    Description: EC2 instance type
    Type: String
    Default: t2.micro
    AllowedValues:
    - t2.nano
    - t2.micro
    - t2.small
    - t2.medium
    - t2.large
    - m4.large
    - m4.xlarge
    - m4.2xlarge
    - m4.4xlarge
    - m4.10xlarge
    - m3.medium
    - m3.large
    - m3.xlarge
    - m3.2xlarge
    - c4.large
    - c4.xlarge
    - c4.2xlarge
    - c4.4xlarge
    - c4.8xlarge
    - c3.large
    - c3.xlarge
    - c3.2xlarge
    - c3.4xlarge
    - c3.8xlarge
    ConstraintDescription: choose from the list above
  KeyName:
    Description: SSH Key pair
    Type: AWS::EC2::KeyPair::KeyName
```



```
    ConstraintDescription: the name of the key-pair
Mappings:
  AMI2RegionMap:
    eu-west-1:
      '64': ami-1234453
    eu-central-1:
      '64': ami-1312312
Resources:

  InstanceProfile:
    Type: AWS::IAM::InstanceProfile
    Properties:
      Path: "/"
      Roles:
      - Ref: InstanceIAMRole
  InstanceIAMRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: '2022-12-16'
        Statement:
        - Effect: Allow
          Principal:
            Service:
            - ec2.amazonaws.com
          Action:
          - sts:AssumeRole
      Path: "/"
      Policies:
      - PolicyName: s3
        PolicyDocument:
          Version: '2022-12-16'
          Statement:
          - Effect: Allow
            Action:
```

```
            - s3:*
            Resource:
            - Fn::Join:
              -"
              - - 'arn:aws:s3:::'
                - "*"
  AutoScalingGroup:
    Type: AWS::AutoScaling::AutoScalingGroup
    Properties:
      Tags:
      - Key: Name
        Value:
          !Ref InstanceName
        PropagateAtLaunch: 'true'
      LaunchConfigurationName:
        Ref: LaunchConfiguration
      MinSize: 1
      MaxSize: 2
      VPCZoneIdentifier:
        - !Ref SubnetId
  LaunchConfiguration:
    Type: AWS::AutoScaling::LaunchConfiguration
    Properties:
      IamInstanceProfile: !Ref InstanceProfile
      KeyName:
        Ref: KeyName
      ImageId:
        Fn::FindInMap:
        - AMI2RegionMap
        - Ref: AWS::Region
        - '64'
      SecurityGroups:
      - Ref: InstanceSecurityGroup
      - Ref: SSHSecurityGroup
      InstanceType:
```



```
        Ref: InstanceType
      UserData:
        Fn::Base64:
          !Sub |
            #!/bin/bash -x
            apt-get update
            apt-get install --yes awscli
  InstanceSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      VpcId:
        !Ref VpcId
      GroupDescription: Enable HTTP and HTTPS
      SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: '80'
        ToPort: '80'
        CidrIp: 0.0.0.0/0
      - IpProtocol: tcp
        FromPort: '443'
        ToPort: '443'
        CidrIp: 0.0.0.0/0

  SSHSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      VpcId:
        !Ref VpcId
      GroupDescription: SSH and HTTP enabled
      SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: '22'
        ToPort: '22'
        CidrIp:
          Ref: allowsshcidr
```

```
155. Outputs:
156.   InstanceSecurityGroup:
157.     Description: ec2-Security-group
158.     Value:
159.       Fn::GetAtt:
160.       - InstanceSecurityGroup
161.       - GroupId
162.
```

The previous code will create a security group that allows SSH, HTTP, and HTTPS plus EC2. We define the EC2 instance type so the user can choose from the list instead of guessing. This EC2 instance will be attached to the IAM role that allows the user to check the S3 services.

Pulumi

Pulumi is a cutting-edge platform for managing infrastructure as code. Connecting with cloud resources via the Pulumi Software Development Kit (SDK) uses pre-existing programming languages and their respective native ecosystems, such as TypeScript, JavaScript, Python, Go, .NET, Java, and markup languages like YAML. A command-line interface (CLI) that can be downloaded, a runtime environment, modules, and a managed service all collaborate to provide a powerful method of creating, updating, and maintaining cloud infrastructure.

Figure 6-18 shows the purpose of Pulumi; like any other IaC tool, each project will have different resources, and the same code will be used to deploy to different environments such as production, QA, and UAT.

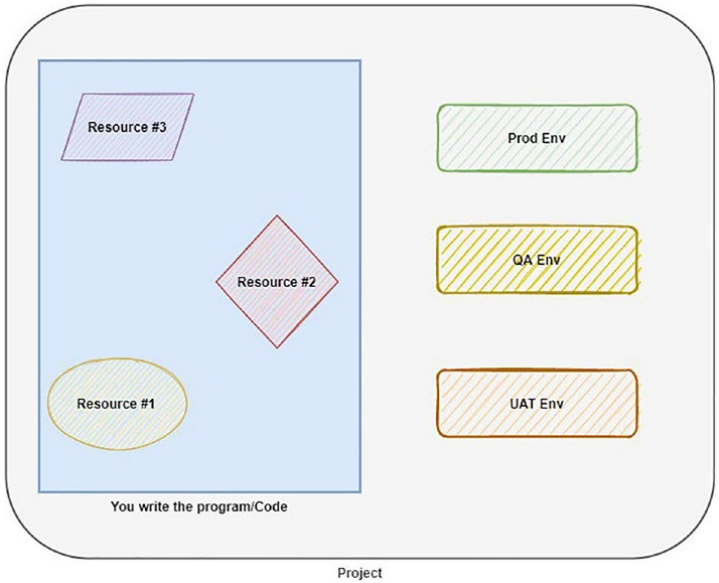

Figure 6-18. *Pulumi*

Pulumi programs deployed using a programming language create and specify your desired cloud infrastructure's architecture. You are allocating resource objects whose characteristics match your infrastructure's intended state, which allows you to define new infrastructure in your application. These attributes are exposed externally to the stack and are used internally to manage dependencies between resources.

A project is a directory containing the program's source code and information about how to launch the application. To compile your code, use the `pulumi up` command from the Pulumi CLI. A stack is a stand-alone instance of your application that can be customized with the help of this command. Stacks can be considered analogous to the various deployment environments when testing and releasing software updates. For instance, you can build and test against separate stacks for development, staging, and production.

Pulumi Concepts

The following are some Pulumi concepts:

- **Project**

  This is any directory where the `Pulumi.yaml` file is located. To detect the active project while working in a subdirectory, Pulumi looks for a `Pulumi.yaml` file in the parent folder. With `pulumi fresh`, you can start a new project. During deployments, a project indicates the runtime to use and where to find the executable application. The following runtime environments are supported: NodeJs, Python, .NET, Go, Java, and YAML.

```
1. name: Nginx-server
2. runtime: Nginx
3. description: Basic example pulumi.yaml project.
4.
```

- **Stacks**

  When you deploy anything using Pulumi, you deploy it to the stack; stacks in Pulumi are individual instances of programs that may be customized separately.

  For example, to create a stack for production, just run the following command:

```
1.  pulumi stack init staging
```

And to list the stack in Pulumi, use this:

```
1. pulumi stack ls
```

The output for the listing will be like the following one:

```
1. NAME                          LAST UPDATE          RESOURCE COUNT
2. Development                   1 hour ago           70
3. production                    6 hours ago          150
4. UAT                           3 weeks ago          90
```


Resources

Any cloud resources such as compute instances, storage buckets, and Kubernetes clusters are all examples of resources, and there are two different types.

- *Custom resource*: This is a resource managed by the cloud provider such as AWS, Azure, and GCP.
- *Component resource*: For example, a VPC in AWS, with Pulumi, has a built-in module to follow the best practice.

State and Back Ends

Like any other IaC, the state is the metadata to allow the tools to be deployed through it. The state is how Pulumi understands when and how to generate, read, delete, or update cloud resources, and each stack has its state.

Having a back end that you control yourself, just run the following command:

```
1. pulumi login
```

This disconnects you from the current back end.

```
1. pulumi logout
```

Because of this, all other stack or state operations will need a new login since all credentials data will be removed from `/.pulumi/credentials.json`.

Provide the URL for the desired back end to access it.

```
1. pulumi login s3://<bucket-name> # General Syntax how to store backend in s3
2. pulumi login 's3://chapter6-bucket-Osama?region=eu-west-1&awssdk=v2&prof
   ile=AWS-book'
```

Inputs and Outputs

You can tell by the name what the files are used for. Input allows a raw value of the specified type (such as string, integer, Boolean, list, map, and so on), and output reads from another source.

Pulumi Example

I prefer Python since it's easy to learn and has all the modules you need, so I will create an S3 bucket in this example.

Pulumi.yaml

```
name: aws-s3-bucket
runtime:
  name: Python
  options:
    virtualenv: venv
description: this is the pulumi s3 IaC
```

Save the code in __main__.py, the top-level code environment in Python. The following example will create a bucket called s3-chapter-bucket; define a policy for that bucket as read-only.

```
import json
import mimetypes
import os

from pulumi import export, FileAsset
from pulumi_aws import s3

web_bucket = s3.Bucket('s3-chapter-6-bucket',
    website=s3.BucketWebsiteArgs(
        index_document= "index.html",
    ))

content_dir = "www"
for file in os.listdir(content_dir):
    filepath = os.path.join(content_dir, file)
    mime_type, _ = mimetypes.guess_type(filepath)
    obj = s3.BucketObject(file,
        bucket=web_bucket.id,
        source=FileAsset(filepath),
        content_type=mime_type)

```



```
def public_read_policy_for_bucket(bucket_name):
    return json.dumps({
"Version": "2012-10-17",
"Statement": [{
"Effect": "Allow",
"Principal": "*",
"Action": [
                "s3:GetObject"
            ],
"Resource": [
                f"arn:aws:s3:::{bucket_name}/*",
            ]
        }]
    })

bucket_name = web_bucket.id
bucket_policy = s3.BucketPolicy("bucket-policy",
    bucket=bucket_name,
    policy=bucket_name.apply(public_read_policy_for_bucket))

# Export the name of the bucket
export('bucket_name', web_bucket.id)
export('website_url', web_bucket.website_endpoint)

```

As you can see, each IaC tool has advantages and disadvantages, but in the end, they each will lead to automation and infrastructure as code.

The tools can even be mixed. Some companies don't like sticking to one IaC, so they try to combine Terraform, CloudFormation, Pulumi, or Ansible, but you decide which one you want to learn and use.

Table 6-4 compares Terraform, CloudFormation, and Pulumi to each other.

Table 6-4. *Comparisons*

Factor	Terraform	CloudFormation	Pulumi
Version	Open-source and enterprise versions	Available only when you use AWS	Open-source and enterprise versions
Language	HCL	YAML or JSON	Python, Java, Node.js, and .NET
Role-based access control	No	Yes	No
User interface	Yes, third-party solution	Yes	No
Community	Massive	Not good	Not good
Multicloud provider	Yes	No	Yes
Integration with other cloud tools	Yes	No	Yes

Ansible

Ansible is similar to Puppet or Chef but simpler to use than any other configuration management tool.

Ansible is a tool that can be used for different purposes; imagine a scenario where you have 10 databases and have been asked to upgrade or patch them. Or say you want to install an application on a different operating system or server, create users, install packages, and more. Ansible will automate this by writing a custom playbook. Ansible is also an easy tool to learn; it already has predefined modules that will make your life easier.

Figure 6-19 shows the high-level design of how Ansible works and its benefits; the DevOps engineer has been requested to install Nginx on three servers, and they already created a playbook called, for example, `webserver-Nginx.yaml`, which will be responsible for doing the following:

1. Install NGINX.
2. Generate an NGINX configuration file.
3. Launch the service using NGINX.

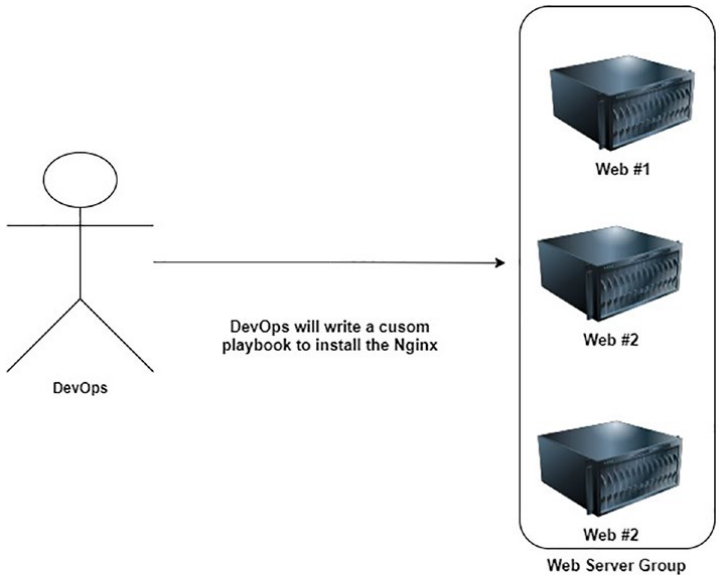

Figure 6-19. *How Ansible works*

Ansible will simultaneously establish SSH connections to servers in parallel. After that, it will simultaneously carry out the first job on the list across all three hosts.

Note the following:

- Each job is executed simultaneously across all hosts when using Ansible.
- Before continuing to the next job, Ansible waits until all hosts have finished the previous task.
- Ansible will perform the functions in the sequence that you specify for them.

Ansible is only one of numerous open-source configuration management technologies available; why should one use it instead of another? I will give different reasons.

- *Easy to learn and use*: Ansible's developers intended for it to be easily installed with no learning curve.

- *Almost nothing to install on the remote server*: Ansible server management requires SSH and Python on Linux servers and WinRM on Windows systems. Since Ansible for Windows utilizes PowerShell rather than Python, no prerequisite host software installation is required.
- *Predefined modules*: Ansible has predefined modules that make your life much easier.
- *Compatibility*: Ansible is compatible with various packaging, database, cloud, notification, monitor tools, etc.

Figure 6-20 shows how to connect to your nodes. Ansible then distributes little programs (known as *Ansible modules*) to them. After that, Ansible will run these modules (through SSH by default) and then delete them when done. You can store your modules on your computer; no servers, daemons, or databases are necessary.

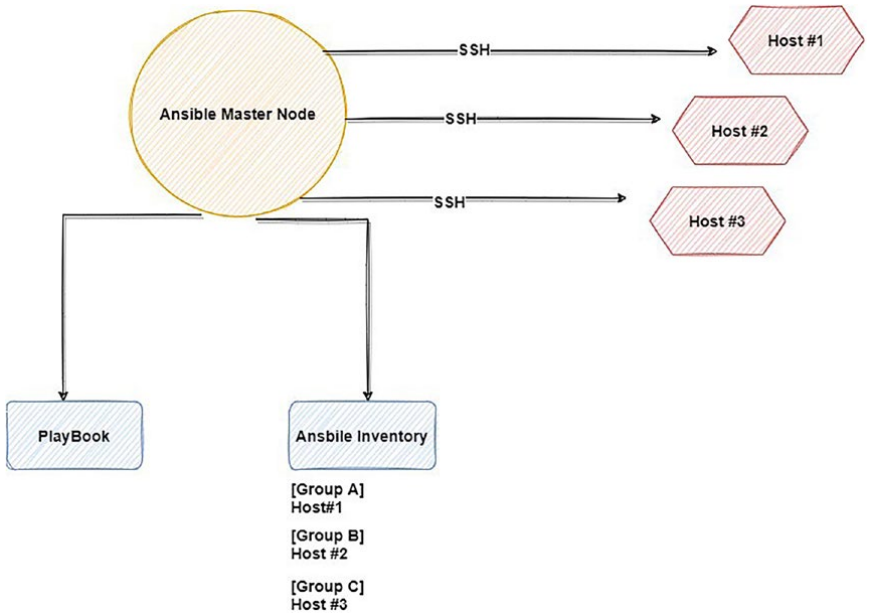

Figure 6-20. *How Ansible works*

Python is used to develop Ansible and runs on Linux, macOS, and BSD. As long as Python is present on Linux, macOS, and BSD systems and PowerShell is present on Windows PCs, you can use it to target any method you choose. Therefore, it is recommended that you set up Ansible on your machine. You should use Python 3.8 on the computer where Ansible is installed.

Installation is pretty simple and doesn't need extra configuration from your side. You can install it in different ways.

RHEL/CentOS Linux

```
1. sudo yum install ansible
```

Debian/Ubuntu Linux

```
1. sudo apt-get install software-properties-common
2. sudo apt-add-repository ppa:ansible/ansible
3. sudo apt-get update
4. sudo apt-get install ansible
```

Install Ansible using `pip` as follows:

```
1. sudo pip install ansible
```

Or the last option is to install it from the source (GitHub):

```
1. git clone git://github.com/ansible/ansible.git
2. cd ./ansible
3. source ./hacking/env-setup
```

To work with Ansible, you need to understand these concepts:

- *Ansible inventory*: This is a file located under `/etc/ansible/hosts`, where you define the servers you want the master to communicate with. It makes your life much easier and allows a playbook's file to specify the hosts and host groups that will be used to execute the playbook's commands, modules, and tasks. According to your Ansible setup and installed plugins, the file might be in various forms.

- *Playbook*: This is a file written by YAML, and it allows you to configure and customize what you need to do with the tool, which is the instruction of what the tool will do and need to implement.
- *Facts*: This refers to the data Ansible will get from the hosts it manages throughout execution. The scripts may then make use of those variables. Data includes everything about a host you require, including its IP address, network interface card (NIC), devices, and so on.
- *Roles*: These are reusable organizational units that facilitate the distribution of automation code between users.
- *Modules*: You can store your modules on your computer; no servers, daemons, or databases are necessary. Generally, you'll need a text editor, a terminal application, and a version control system to keep track of the many iterations of your work.
- *Security*: Ansible can be used with passwords, although SSH keys and ssh-agent are among the most secure options. However, Kerberos may also be used successfully. There are multiple choices! Rather than using the root account, you can join using any other user and switch identities using `su` or `sudo`.
- *CLI*: The so-called ad hoc command-line utility is available for usage. This utility will let you manage OS users across numerous servers with a single command line.

Let's talk about the ad hoc commands and their purpose. When it comes to seldom-used tasks, ad hoc commands shine. To turn off all the lab equipment before a holiday, for instance, a simple one-liner in Ansible would suffice; no playbook would be necessary.

Here is how to reboot the server

```
1. ansible group-web-server -a "/sbin/reboot"
```

- Here, `group-web-server` is a group. We defined it inside the Ansible inventory to indicate to the web server, and maybe it has 10 servers or more.
- `/sbin/reboot` is the Linux command.

Here is how to get the uptime for a group of servers:

```
1. ansible all -m shell -a uptime
```

Here is how to get the disk size:

```
1. ansible all -m shell -a df -h
```

You need to understand that covering these tools in one chapter is a tough job; each one needs a book by itself. Here I am giving you a general idea of how these tools work.

Moving to the playbook and how it works, I will provide you with a simple example.

Simple Playbook Example

Ansible installation on your local workstation and access to a remote Ubuntu server are prerequisites for this tutorial.

Installing Apache Server is one of the most straightforward examples that can be given, but be careful. YAML is an indention language; spaces are essential here, so use a good integrated development environment (IDE).

`apache-installation.yaml`

```
 1. ---
 2.   - name: Playbook
 3.     hosts: webservers #should be defined inside an inventory (/etc/
        ansible/hosts)
 4.     become: yes
 5.     become_user: root
 6.     tasks:
 7.       - name: download the latest version
 8.         yum:
 9.           name: httpd
10.           state: latest
11.       - name: make sure apache services are up and running
12.         service:
13.           name: httpd
14.           state: started
15.
```

Here, note the following:

- `name` is your playbook name.
- `hosts` should be defined inside an inventory (`/etc/ansible/hosts`).
- `become` tells Ansible that we will switch to a higher user.
- `Become_user` is our high-privilege user.
- `tasks` says what the playbook will do.

In our playbook, we have two tasks; the first will install Apache, and the second will ensure the Apache services are running and then run this custom playbook.

```
1. ansible-playbook apache-installation.yaml
```

We could make the previous playbook more usable by using a variable that allows you to read a different value, and you can change it when you want.

```
 1. ---
 2.   - name: Playbook
 3.     hosts: webservers #should be defined inside an inventory (/etc/
        ansible/hosts)
 4.     become: yes
 5.     become_user: root
 6.     vars:
 7.        key_file:  /etc/apache/ssl/DevOpsBook.key
 8.        cert_file: /etc/apache/ssl/DevOpsBook.cert
 9.        server_name: www.DevOpsBook.com
10.     tasks:
11.       - name: download the latest version
12.         yum:
13.           name: httpd
14.           state: latest
15.       - name: make sure apache services are up and running
16.         service:
17.           name: httpd
18.           state: started
19.       - name: copy certificate to another place, for example
20.         copy:
21.           src: {{key_file}}
22.           dest: {{cert_file}}
```


For another example, let's assume you have a group defined inside the inventory that contains, for instance, 15 servers. You need to copy files, create a user, and upgrade the packages on all the servers. What will you do? There is no time to do them one by one.

```
 1. ---
 2. - name: Another Custom playbook
 3.   hosts: all
 4.
 5.   tasks:
 6.   - name: Copy files and dont forget the permission
 7.     ansible.builtin.copy:
 8.       src: ./hosts
 9.       dest: /tmp/hosts_backup
10.       mode: '0655'
11.   - name: Add the user 'Osama'
12.     ansible.builtin.user:
13.       name: Osama
14.     become: yes
15.     become_method: sudo
16.   - name: Upgrade packages
17.     apt:
18.       force_apt_get: yes
19.       upgrade: dist
20.     become: yes
21.
```

Like the earlier example, you define the name of your playbook and which servers you need to run this playbook.

- `Ansible.builtin.copy` is a predefined module to allow copying.
- `ansible.builtin.user` is a predefined module to create a user.

Ansible is one of the most common tools in use, and it has a lot of other features. Here are some examples:

- *Loops*: A loop allows you to operate repeatedly on each item in a list. In a loop, you repeat the process while changing the item's value each time.
- *Handlers*: One of the conditional forms that Ansible offers is a handler. A handler acts like a task but executes only when another job triggers it. If Ansible determines that a task has altered the system state, the task will issue the notice.

Summary

In this chapter, you learned about IaC. Even after putting in so much time and effort on these pages, I can scarcely expect you to be able to write a complete IaC program, but at least you will have an idea of where to start.

There is no right or good tool to use; it all depends on your use case and what you feel comfortable with.

Using Terraform as the basis for your infrastructure may get you a long way and simplify management. If you don't want to invest time and learn a programing language or something so complicated, then Terraform is for you; if you already know one of the programming languages, in that case, you could use Pulumi.

CloudFormation is a valuable tool if you currently use the AWS cloud and do not intend to switch to another cloud provider or distribute your workloads over several clouds in the foreseeable future.

In the next chapter, I will cover two of the most critical aspects of DevOps and SRE, monitoring and observability; the chapter will cover the terms related to these two concepts and how you can troubleshoot them. In addition, we will discuss the Amazon AWS services that allow you to keep an eye on the application, infrastructure, and security.