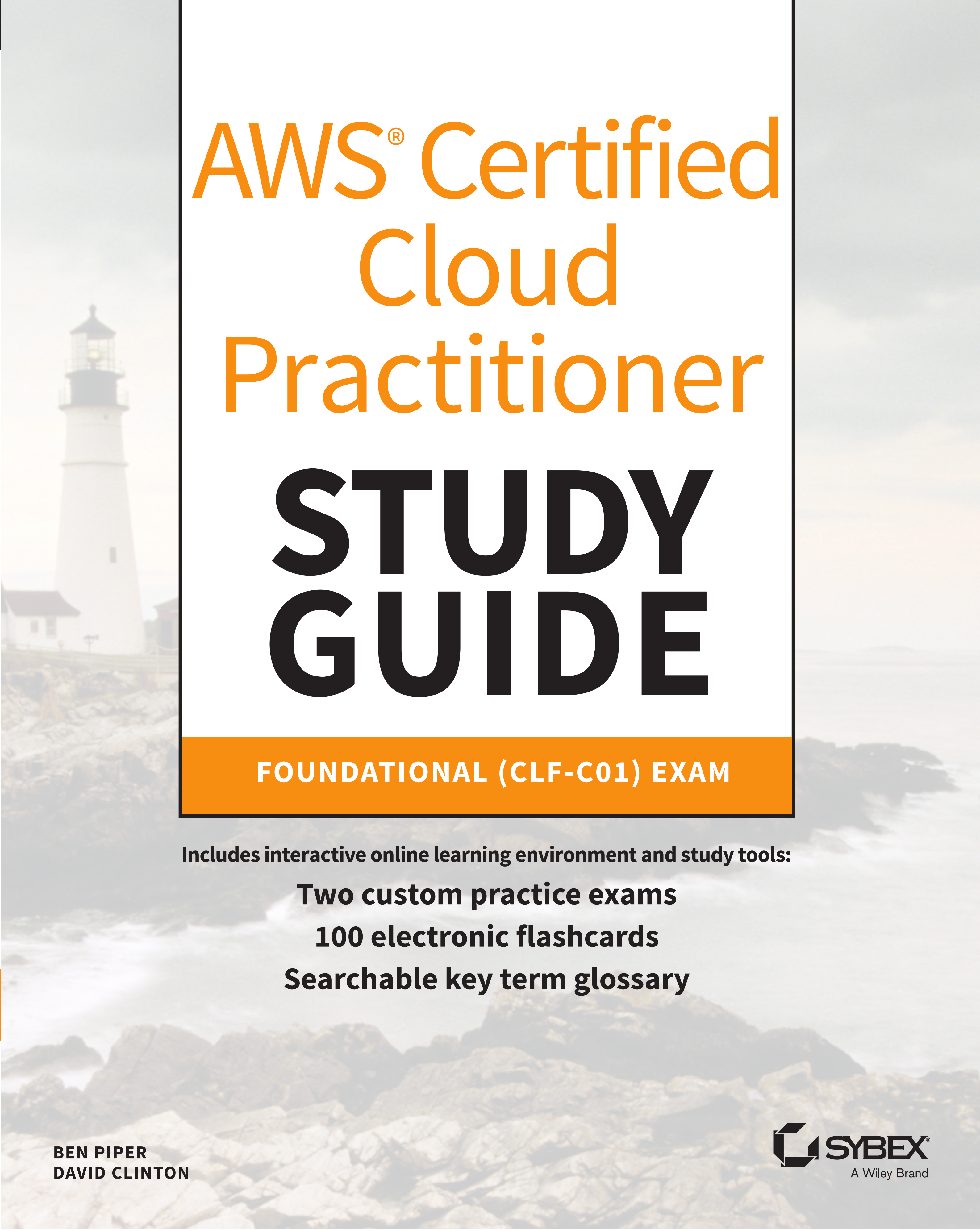


AWS® Certified Cloud Practitioner STUDY GUIDE

FOUNDATIONAL (CLF-C01) EXAM

Includes interactive online learning environment and study tools:

Two custom practice exams

100 electronic flashcards

Searchable key term glossary

BEN PIPER
DAVID CLINTON

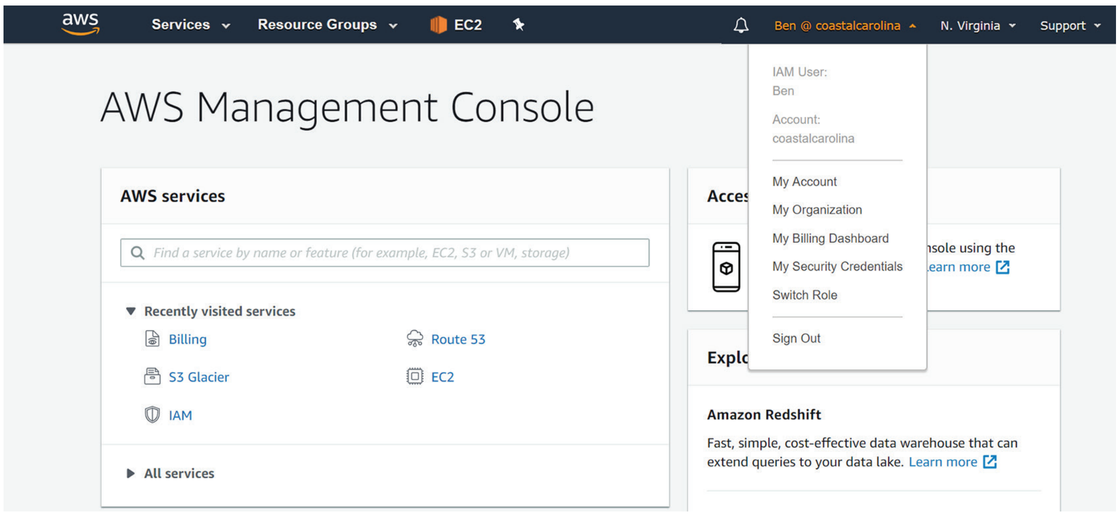
SYBEX®
A Wiley Brand

The Account Name Menu

The account name menu in the navigation bar displays the IAM user and account alias or ID you're logged in as, or the name associated with your AWS account if you're logged in as root. Choosing the menu gives you the following options, as shown in Figure 6.8:

- My Account—Takes you to the Billing service console page that displays your account information
- My Organization—Takes you to the AWS Organizations service console
- My Billing Dashboard—Takes you to the Billing and Cost Management Dashboard
- My Security Credentials—Takes you to the My Password page in the IAM service console where you can change your password
- Switch Role—Lets you assume an IAM role where you can perform tasks as a different principal with different permissions
- Sign Out—Signs you out of the AWS Management Console

FIGURE 6.8 The account name menu when you're logged in as an IAM user



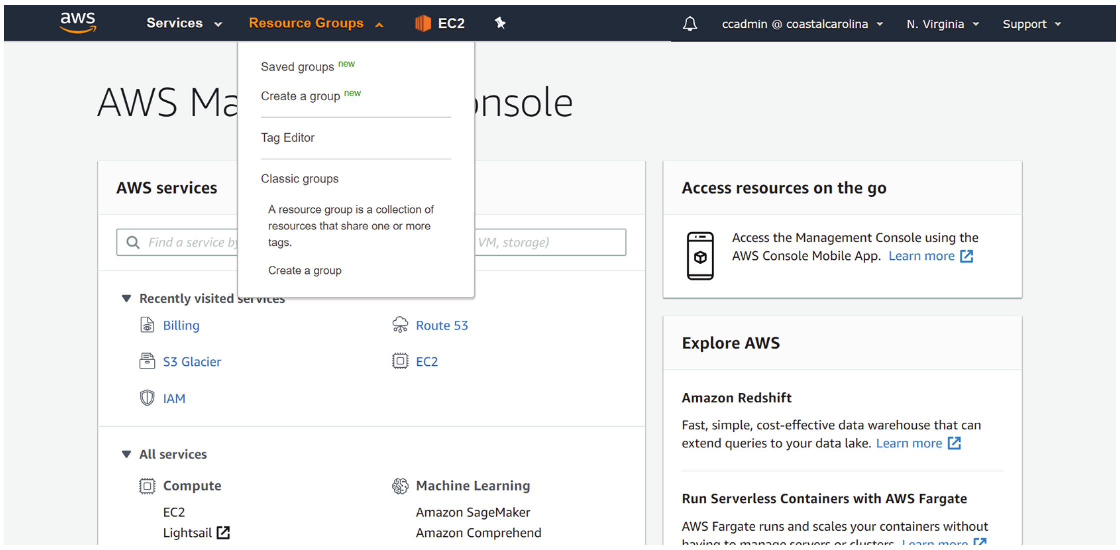
Fast, simple, cost-effective data warehouse that can extend queries to your data lake. Learn more

Resource Groups

Resource groups are one of the options readily available from the AWS Management Console. Choosing the Resource Groups link in the navigation bar, as shown in Figure 6.9, will reveal options to create new resource groups and view existing ones.

Resource groups let you view, manage, and automate tasks on multiple AWS resources at a time. They are ideal for grouping AWS resources that all compose a particular application. Rather than having to remember which resources belong to which application, you can add those resources to a resource group and let AWS remember for you! You can then perform bulk actions against those resources.

FIGURE 6.9 The Resource Groups menu



A resource group is a collection of resources that share one or more tags.

Access the Management Console using the AWS Console Mobile App. Learn more

Fast, simple, cost-effective data warehouse that can extend queries to your data lake. Learn more

AWS Fargate runs and scales your containers without

A resource group is a collection of AWS resources in the same region that match the results of a query. You can create a resource group from a query based on resource tags or from a CloudFormation Stack.

Resource tags *Resource tags* are optional metadata that you can assign to AWS resources. A tag must contain a label called a *key* and may optionally contain a value. For example, you may tag your production resources using a key named `Environment` and a value named `Production`. Tag keys and values can include letters, numbers, spaces, and these characters: `+ - = . _ : / @`. You can assign multiple tags to a resource, up to 50. Tag keys and values are case-sensitive. To create a resource group using tags, you can query based on a tag key only or both a tag key and its value.

AWS CloudFormation stacks CloudFormation, which you'll learn about in Chapter 11, "Automating Your AWS Workloads," lets you programmatically deploy and manage multiple AWS resources as a single unit called a *stack*. You can create resource groups that contain some or all of the resources in a CloudFormation stack. You can choose all resources in the stack or filter by resource type, such as EC2 instance, EC2 security group, or S3 bucket.

Tag Editor

If you want to create a resource group based on tags, you'll first need to tag the resources you want to include in the group. Thankfully the AWS Management Console makes this easy. Choose the Resource Groups link in the navigation bar of the AWS Management Console, and then choose Tag Editor, as shown in Figure 6.9.

To get started with Tag Editor, create a query to find the resources you want to tag. At a minimum, you must select at least one region and one or more resource types, such as EC2 instances, EC2 volumes, VPCs, or S3 buckets. You can select all resources if you want. Optionally, you can list resources that already have an existing tag. For example, you

The Declarative Approach

Using a declarative approach, you write code that declares the desired result of the task, rather than how to carry it out. For example, rather than using the AWS CLI to create a new virtual private cloud (VPC) and subnets, you could write declarative code that simply defines the configuration for the VPC and subnets. This approach naturally requires some intelligent software to figure out the operations required to achieve the desired result. CloudFormation, which is covered in the first part of this chapter, is the most well-known AWS service that takes a declarative approach to building and configuring your AWS infrastructure. In short, you write code containing the specifics of the AWS resources you want and present that code to CloudFormation, and it builds and configures those resources on your behalf in a matter of seconds.

Infrastructure as Code

Using code to define your infrastructure and configurations is commonly called the *infrastructure as code* (IaC) approach. Defining IaC is a fundamental element of automation in the cloud. Because automation relies on code being executed by a machine, it reduces the risk of human error inherent in carrying out the same work manually. This in turn reduces rework and other problems down the line.

In this chapter, we cover the following AWS services that enable automation in different ways:

- CloudFormation—Lets you automate the building and configuration of your AWS resources
- The AWS Developer Tools of CodeCommit, CodeBuild, CodeDeploy, and CodePipeline—Automate the testing, building, and deployment of applications to EC2 and on-premises instances
- EC2 Auto Scaling—Automatically launches, configures, and terminates EC2 instances as needed to meet fluctuating demand
- Systems Manager—Automates common operational tasks such as patching instances and backing up Elastic Block Store (EBS) volumes
- OpsWorks—A collection of three different offerings that help automate instance configuration and application deployments using the popular Chef and Puppet configuration management platforms

CloudFormation

CloudFormation automatically creates and configures your AWS infrastructure from code that defines the resources you want it to create and how you want those resources configured.

Templates

The code that defines your resources is stored in text files called *templates*. Templates use the proprietary CloudFormation language, which can be written in JavaScript Object Notation (JSON) or YAML format.

Templates contain a description of the AWS resources you want CloudFormation to create, so they simultaneously function as infrastructure documentation. Because templates are files, you can store them in a version-controlled system such as an S3 bucket or a Git repository, allowing you to track changes over time.

You can use the same template to build AWS infrastructure repeatedly, such as for development or testing. You can also use a single template to build many similar environments. For example, you can use one template to create identical resources for both production and test environments. Both environments can have the same infrastructure—VPCs, subnets, application load balancers, and so on—but they'll be separate environments. CloudFormation uses random identifiers when naming resources to ensure the resources it creates don't conflict with each other.

You can write templates to optionally take parameter inputs, allowing you to customize your resources without modifying the original template. For instance, you can write a template that asks for a custom resource tag to apply to all the resources CloudFormation creates. You can make a parameter optional by specifying a default value, or you can require the user to specify a value in order to use the template.

Stacks

To provision resources from a template, you must specify a stack name that's unique within your account. A *stack* is a container that organizes the resources described in the template.

The purpose of a stack is to collectively manage related resources. If you delete a stack, CloudFormation automatically deletes all of the resources in it. This makes CloudFormation perfect for test and development environments that need to be provisioned as pristine and then thrown away when no longer needed. Deleting a stack helps ensure that all resources are deleted and that no forgotten resources are left lingering, running up charges.

NOTE

One of the fundamental principles of good cloud design is to make your test environments mirror your production environments as closely as possible. CloudFormation makes this not only possible but almost trivially easy!

Stack Updates

You can have CloudFormation change individual resources in a stack. Just modify the template to delete, modify, or add resources, and then instruct CloudFormation to perform a stack update using the template. CloudFormation will automatically update the resources accordingly. If any other resources are dependent upon a resource that you've updated, CloudFormation will detect that and make sure those downstream resources are also reconfigured properly.

Alternatively, you can create a change set. Make the desired changes to your template, and then have CloudFormation generate a change set based on the template. CloudFormation will let you review the specific changes it will make, and then you can decide whether to execute those changes or leave everything as is.

CloudFormation makes it easy to update stacks, increasing the possibility that a less skilled user might inadvertently update a stack, making undesired changes to a critical resource. If you're concerned about this possibility, you can create a stack policy to guard against accidental updates. A stack policy is a JSON document, separate from a template, that specifies what resources may be updated. You can use it prevent updates to any or all resources in a stack. If you absolutely need to update a stack, you can temporarily override the policy.

Tracking Stack Changes

Each stack contains a timestamped record of events that occur within the stack, including when resources were created, updated, or deleted. This makes it easy to see all changes made to your stack.

It's important to understand that resources in CloudFormation stacks can be changed manually, and CloudFormation doesn't prevent this. For instance, you can manually delete a VPC that's part of a CloudFormation stack. Drift detection is a feature that monitors your stacks for such changes and alerts you when they occur.

CloudFormation vs. the AWS CLI

You can script AWS CLI commands to create resources fast and automatically, but those resources won't be kept in a stack, so you won't get the same advantages that stacks provide.

When you're using the AWS CLI, you can't update your resources as easily as you can with CloudFormation. With CloudFormation, you simply adjust the template or parameters to change your resources, and CloudFormation figures out how to perform the changes. With the AWS CLI, it's up to you to understand how to change each resource and to ensure that a change you make to one resource doesn't break another.

EXERCISE 11.1

Explore the CloudFormation Designer

In this exercise, you'll use the CloudFormation Designer to look at a sample template that defines a simple environment containing a Linux EC2 instance. In addition to letting you view and edit the template directly, the CloudFormation Designer also visualizes the resources in a template, making it easy to see exactly what CloudFormation will build.

1. Browse to the CloudFormation service console.
2. Choose the Create New Stack button.

EXERCISE 11.1 *(continued)*

3. In the Choose A Template section, choose the Select A Sample Template drop-down, and select LAMP Stack.
4. Choose the View/Edit Template In Designer link.
5. The Designer will show you icons for two resources: an EC2 instance and a security group. Choose the instance icon. The Designer will take you to the section of the template that defines the instance.

AWS Developer Tools

The AWS Developer Tools are a collection of tools designed to help application developers develop, build, test, and deploy their applications onto EC2 and on-premises instances. These tools facilitate and automate the tasks that must take place to get a new application revision released into production.

However, the AWS Developer Tools enable more than just application development. You can use them as part of any IaC approach to automate the deployment and configuration of your AWS infrastructure.

In this section, you'll learn about the following AWS Developer Tools:

- CodeCommit
- CodeBuild
- CodeDeploy
- CodePipeline

CodeCommit

CodeCommit lets you create private Git repositories that easily integrate with other AWS services. Git (`https://git-scm.com`) is a version control system that you can use to store source code, CloudFormation templates, documents, or any arbitrary files—even binary files such as Amazon Machine Images (AMI) and Docker containers. These files are stored in a repository, colloquially known as a *repo*.

Git uses a process called *versioning*, where all changes or commits to a repository are retained indefinitely, so you can always revert to an old version of a file if you need it.

CodeCommit is useful for teams of people who need to collaborate on the same set of files, such as developers who collaborate on a shared source code base for an application. CodeCommit allows users to check out code by copying or cloning it locally to their machine. They make changes to it locally and then check it back in to the repository.

Command Documents

Command documents are scripts that run once or periodically that get the system into the state you want.

Using Command documents, you can install software on an instance, install the latest security patches, or take inventory of all software on an instance. You can use the same Bash or PowerShell commands you'd use with a Linux or Windows instance. Systems Manager can run these periodically, or on a trigger, such as a new instance launch. Systems Manager requires an agent to be installed on the instances that you want it to manage.

Configuration Compliance

Configuration Compliance is a feature of Systems Manager that can show you whether instances are in compliance with your desired configuration state—whether it's having a certain version of an application installed or being up-to-date on the latest operating system security patches.

Automation Documents

In addition to providing configuration management for instances, Systems Manager lets you perform many administrative AWS operations you would otherwise perform using the AWS Management Console or the AWS CLI. These operations are defined using Automation documents. For example, you can use Systems Manager to automatically create a snapshot of an Elastic Block Store (EBS) volume, launch or terminate an instance, create a CloudFormation stack, or even create an AMI from an existing EBS volume.

Distributor

Using Systems Manager Distributor, you can deploy installable software packages to your instances. You create a .zip archive containing installable or executable software packages that your operating system recognizes, put the archive in an S3 bucket, and tell Distributor where to find it. Distributor takes care of deploying and installing the software. Distributor is especially useful for deploying a standard set of software packages that already come as installers or executables.

OpsWorks

OpsWorks is a set of three different services that let you take a declarative approach to configuration management. As explained earlier in this chapter, using a declarative approach requires some intelligence to translate declarative code into imperative operations. OpsWorks uses two popular configuration management platforms that fulfill this requirement: Chef (`https://www.chef.io`) and Puppet Enterprise (`https://puppet.com`).

Puppet and Chef are popular configuration management platforms that can configure operating systems, deploy applications, create databases, and perform just about any configuration task you can dream of, all using code. Both Puppet and Chef are widely used for on-premises deployments, but OpsWorks brings their power to AWS as the OpsWorks managed service.

Just as with automation in general, configuration management is not an all-or-nothing decision. Thankfully, OpsWorks comes in three different flavors to meet any configuration management appetite.

AWS OpsWorks for Puppet Enterprise and AWS OpsWorks for Chef Automate are robust and scalable options that let you run managed Puppet or Chef servers on AWS. This is good if you want to use configuration management across all your instances.

AWS OpsWorks Stacks provides a simple and flexible approach to using configuration management just for deploying applications. Instead of going all-in on configuration management, you can just use it for deploying and configuring applications. OpsWorks Stacks takes care of setting up the supporting infrastructure.

AWS OpsWorks for Puppet Enterprise and AWS OpsWorks for Chef Automate

The high-level architectures of AWS OpsWorks for Puppet Enterprise and Chef Automate are similar. They consist of at least one Puppet master server or Chef server to communicate with your managed nodes—EC2 or on-premises instances—using an installed agent.

You define the configuration state of your instances—such as operating system configurations applications—using Puppet modules or Chef recipes. These contain declarative code, written in the platform's domain-specific language, that specifies the resources to provision. The code is stored in a central location, such as a Git repository like CodeCommit or an S3 bucket.

OpsWorks manages the servers, but you're responsible for understanding and operating the Puppet or Chef software, so you'll need to know how they work and how to manage them. No worries, though, because both Chef and Puppet have large ecosystems brimming with off-the-shelf code that can be used for a variety of common scenarios.

AWS OpsWorks Stacks

If you like the IaC concept for your applications, but you aren't familiar with managing Puppet or Chef servers, you can use AWS OpsWorks Stacks.

OpsWorks Stacks lets you build your application infrastructure in stacks. A *stack* is a collection of all the resources your application needs: EC2 instances, databases, application load balancers, and so on. Each stack contains at least one layer, which is a container for some component of your application.

To understand how you might use OpsWorks Stacks, consider a typical database-backed application that consists of three layers:

- An application layer containing EC2 instances or containers
- A database layer consisting of self-hosted or relational database service (RDS) database instances
- An application load balancer that distributes traffic to the application layer

There are two basic types of layers that OpsWorks uses: OpsWorks layers and service layers.

NOTE Despite the name, an OpsWorks stack is not the same as a CloudFormation stack and doesn't use CloudFormation templates.

OpsWorks layers An OpsWorks layer is a template for a set of instances. It specifies instance-level settings such as security groups and whether to use public IP addresses. It also includes an auto-healing option that automatically re-creates your instances if they fail. OpsWorks can also perform load-based or time-based auto scaling, adding more EC2 instances as needed to meet demand.

An OpsWorks layer can provision Linux or Windows EC2 instances, or you can add existing Linux EC2 or on-premises instances to a stack. OpsWorks Stacks supports Amazon Linux, Ubuntu Server, CentOS, and Red Hat Enterprise Linux.

To configure your instances and deploy applications, OpsWorks uses the same declarative Chef recipes as the Chef Automate platform, but it doesn't provision a Chef server. Instead, OpsWorks Stacks performs configuration management tasks using the Chef Solo client that it installs on your instances automatically.

Service layers A stack can also include service layers to extend the functionality of your stack to include other AWS services. Service layers include the following layers:

Relational Database Service (RDS) Using an RDS service layer, you can integrate your application with an existing RDS instance.

Elastic Load Balancing (ELB) If you have multiple instances in a stack, you can create an application load balancer to distribute traffic to them and provide high availability.

Elastic Container Service (ECS) Cluster If you prefer to deploy your application to containers instead of EC2 instances, you can create an ECS cluster layer that connects your OpsWorks stack to an existing ECS cluster.

Summary

If there's one thing that should be clear, it's that AWS provides many different ways to automate the same task. The specific services and approaches you should use are architectural decisions beyond the scope of this book, but you should at least understand how each of the different AWS services covered in this chapter can enable automation.

Fundamentally, automation entails defining a task as code that a system carries out. This code can be written as imperative commands that specify the exact steps to perform the task. The most familiar type of example is the Bash or PowerShell script system administrators write to perform routine tasks. AWS Systems Manager, CodeBuild, and CodeDeploy use an imperative approach. Even the Userdata scripts that you use with EC2 Auto Scaling are imperative.

Code can also be written in a more abstract, declarative form, where you specify the end result of the task. The service providing the automation translates those declarative statements into step-by-step operations that it carries out. CloudFormation, OpsWorks for Puppet Enterprise, OpsWorks for Chef Automate, and OpsWorks Stacks use declarative languages to carry out automation.

At the end of the day, both the imperative and declarative approaches result in nothing more than a set of commands that must be carried out sequentially. It's important to understand that the imperative and declarative approaches are not opposites. Rather, the declarative approach abstracts away from you the step-by-step details in favor of a more results-oriented, user-friendly paradigm. The difference comes down to what approach you prefer and what the service requires.

Configuration management is a form of automation that emphasizes configuration consistency and compliance. Naturally, the focus on achieving and maintaining a certain state lends itself to a declarative approach, so many configuration management platforms such as Puppet and Chef use declarative language. However, AWS Systems Manager also provides configuration management, albeit by using an imperative approach.

Exam Essentials

Know what specific tasks AWS services can automate. CloudFormation can automatically deploy, change, and even delete AWS resources in one fell swoop.

The AWS Developer Tools—CodeCommit, CodeBuild, CodeDeploy, and CodePipeline—can help automate some or all of the software development, testing, and deployment process.

EC2 Auto Scaling automatically provisions a set number of EC2 instances. You can optionally have it scale in or out according to demand or a schedule.

Systems Manager Command documents let you automate tasks against your instance operating systems, such as patching, installing software, enforcing configuration settings, and collecting inventory. Automation documents let you automate many administrative AWS tasks that would otherwise require using the management console or CLI.

OpsWorks for Puppet Enterprise and OpsWorks for Chef Automate also let you configure your instances and deploy software but do so using the declarative language of Puppet modules or Chef recipes.

OpsWorks Stacks can automate the build and deployment of an application and its supporting infrastructure.

Understand the benefits of automation and infrastructure as code. Automation allows common, repetitive tasks to be executed faster than doing them manually and reduces the risk of human error. When you automate infrastructure builds using code, the code simultaneously serves as *de facto* documentation. Code can be placed into version control, making it easy to track changes and even roll back when necessary.

Be able to explain the concepts of continuous integration and continuous delivery. The practice of continuous integration involves developers regularly checking in code as they create or change it. An automated process performs build and test actions against it. This immediate feedback loop allows developers to fix problems quickly and early.

Continuous delivery expands upon continuous integration but includes deploying the application to production after a manual approval. This effectively enables push-button deployment of an application to production.

Review Questions

1. Which of the following is an advantage of using CloudFormation?
   - **A.** It uses the popular Python programming language.
   - **B.** It prevents unauthorized manual changes to resources.
   - **C.** It lets you create multiple separate AWS environments using a single template.
   - **D.** It can create resources outside of AWS.
2. What formats do CloudFormation templates support? (Select TWO.)
   - **A.** XML
   - **B.** YAML
   - **C.** HTML
   - **D.** JSON
3. What's an advantage of using parameters in a CloudFormation template?
   - **A.** Allow customizing a stack without changing the template.
   - **B.** Prevent unauthorized users from using a template.
   - **C.** Prevent stack updates.
   - **D.** Allow multiple stacks to be created from the same template.
4. Why would you use CloudFormation to automatically create resources for a development environment instead of creating them using AWS CLI commands? (Select TWO.)
   - **A.** Resources CloudFormation creates are organized into stacks and can be managed as a single unit.
   - **B.** CloudFormation stack updates help ensure that changes to one resource won't break another.
   - **C.** Resources created by CloudFormation always work as expected.
   - **D.** CloudFormation can provision resources faster than the AWS CLI.
5. What are two features of CodeCommit? (Select TWO.)
   - **A.** Versioning
   - **B.** Automatic deployment
   - **C.** Differencing
   - **D.** Manual deployment
6. In the context of CodeCommit, what can differencing accomplish?
   - **A.** Allowing reverting to an older version of a file
   - **B.** Understanding what code change introduced a bug
   - **C.** Deleting duplicate lines of code
   - **D.** Seeing when an application was last deployed

7. What software development practice regularly tests new code for bugs but doesn't do anything else?
   - A. Differencing
   - B. Continuous deployment
   - C. Continuous delivery
   - D. Continuous integration

8. Which CodeBuild build environment compute types support Windows operating systems? (Select TWO.)
   - A. build.general2.large
   - B. build.general1.medium
   - C. build.general1.small
   - D. build.general1.large
   - E. build.windows1.small

9. What does a CodeBuild environment always contain? (Select TWO.)
   - A. An operating system
   - B. A Docker image
   - C. The Python programming language
   - D. .NET Core
   - E. The PHP programming language

10. Which of the following can CodeDeploy do? (Select THREE.)
    - A. Deploy an application to an on-premises Windows instance.
    - B. Deploy a Docker container to the Elastic Container Service.
    - C. Upgrade an application on an EC2 instance running Red Hat Enterprise Linux.
    - D. Deploy an application to an Android smartphone.
    - E. Deploy a website to an S3 bucket.

11. What is the minimum number of actions in a CodePipeline pipeline?
    - A. 1
    - B. 2
    - C. 3
    - D. 4
    - E. 0

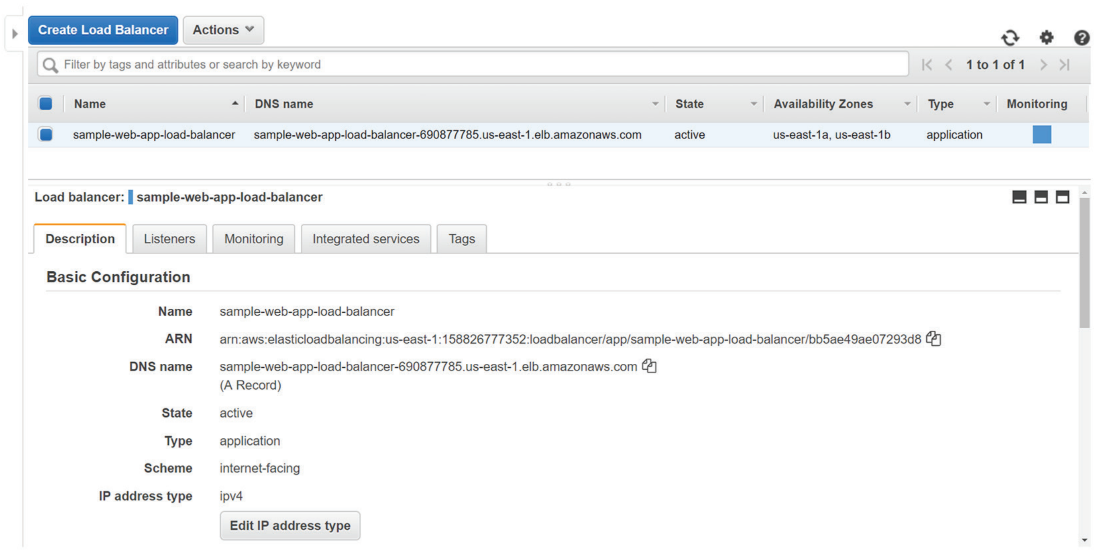
12. You want to predefine the configuration of EC2 instances that you plan to launch manually and using Auto Scaling. What resource must you use?
    - A. CloudFormation template
    - B. Instance role
    - C. Launch configuration
    - D. Launch template

EXERCISE 12.2 *(continued)*

17. For Protocol and Port, select HTTP and 80, respectively.
18. Under Health Checks, make sure Protocol and Path are HTTP and /, respectively.
19. Select the button titled Next: Register Targets.
20. The Auto Scaling group that you'll create will add instances to the target group, so there's no need to do that manually here. Select the button titled Next: Review.
21. Review your settings, and select the Create button. It may take a few minutes for your load balancer to become active.

Once AWS has provisioned the load balancer, you should be able to view its publicly resolvable DNS name and other details, as shown in Figure 12.5. Make a note of the DNS name because you'll need it later.

FIGURE 12.5 Application load balancer details



	Name	DNS name	State	Availability Zones	Type	Monitoring
■	sample-web-app-load-balancer	sample-web-app-load-balancer-690877785.us-east-1.elb.amazonaws.com	active	us-east-1a, us-east-1b	application	■

Load balancer: sample-web-app-load-balancer

Basic Configuration

Name	sample-web-app-load-balancer
ARN	arn:aws:elasticloadbalancing:us-east-1:158826777352:loadbalancer/app/sample-web-app-load-balancer/bb5ae49ae07293d8
DNS name	sample-web-app-load-balancer-690877785.us-east-1.elb.amazonaws.com (A Record)
State	active
Type	application
Scheme	internet-facing
IP address type	ipv4

Creating a Launch Template

Before creating the Auto Scaling group, you need to create a launch template that Auto Scaling will use to launch the instances and install and configure the Apache web server software on them when they're launched. Because creating the launch template by hand would be cumbersome, you'll instead let CloudFormation create it by deploying the CloudFormation template at `https://s3.amazonaws.com/aws-ccp/launch-template.yaml`. The launch template that CloudFormation will create will install Apache on each instance that Auto Scaling provisions. You can also create a custom launch template for your own application. Complete Exercise 12.3 to get some practice with CloudFormation and create the launch template.

EXERCISE 12.3

Create a Launch Template

In this exercise, you'll use CloudFormation to deploy an EC2 launch template that Auto Scaling will use to launch new instances.

1. Browse to the CloudFormation service console. Make sure you're in the AWS Region where you want your instances created.
2. Select the Create Stack button.
3. Under Choose A Template, select the radio button titled Specify An Amazon S3 Template URL.
4. In the text field, enter **https://s3.amazonaws.com/aws-ccp/launch-template.yaml**.
5. Select the Next button.
6. In the Stack Name field, enter **sample-app-launch-template**.
7. From the drop-down list, select the instance type you want to use, or stick with the default t2.micro instance type.
8. Select the Next button.
9. On the Options screen, stick with the defaults and select the Next button.
10. Review your settings, and select the Create button.

CloudFormation will take you to the Stacks view screen. Once the stack is created, the status of the sample-app-launch-template stack will show as `CREATE_COMPLETE`, indicating that the EC2 launch template has been created successfully.

Creating an Auto Scaling Group

EC2 Auto Scaling is responsible for provisioning and maintaining a certain number of healthy instances. By default, Auto Scaling provides reliability by automatically replacing instances that fail their EC2 health check. You'll reconfigure Auto Scaling to monitor the ELB health check and replace any instances on which the application has failed.

To achieve performance efficiency and make this configuration cost-effective, you'll create a dynamic scaling policy that will scale the size of the Auto Scaling group in or out between one and three instances, depending on the average aggregate CPU utilization of the instances. If the CPU utilization is greater than 50 percent, it indicates a heavier load, and Auto Scaling will scale out by adding another instance. On the other hand, if the utilization drops below 50 percent, it indicates that you have more instances than you need, so Auto Scaling will scale in. This is called a *target tracking policy*.

NOTE

EC2 reports these metrics to CloudWatch, where you can graph them to analyze your usage patterns over time. CloudWatch stores metrics for up to 15 months.