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

through the Secure Shell (SSH) protocol for this demo. Otherwise, acknowledge the consequences of launching an instance without a key pair (something that would normally be a bit crazy), and go ahead with the launch.

7. The blue View Instances button at the bottom of the confirmation screen will take you to the EC2 Instances Dashboard where you should see your instance status. When the Instance State indicator turns green and the word *running* appears next to it, your web server should be live.

8. Making sure this instance is selected, find the IPv4 Public IP entry on the Description tab in the bottom half of the screen. Hover your mouse over the IP address that's listed there, and choose the Copy To Clipboard icon to its right. Open a new tab in your browser, and paste the IP address into the URL field. You should see the default Apache test page.

9. When you're done admiring your handiwork, you can shut down the instance from the Actions menu at the top of the Instances page. With your instance selected, choose Action, then Instance State, and finally Terminate. When you confirm the action, your instance will be shut down, and you won't be billed any further. If your account is still in its first year and you haven't been running similar instances for more than 750 hours this month, this Free Tier instance won't have cost you anything anyway.

Simplified Deployments Through Managed Services

Building and administrating software applications can be complex wherever you deploy them. Whether they're on-premises or in the cloud, you'll face a lot of moving parts existing in a universe where they all have to play nicely together or the whole thing can collapse. To help lower the bar for entry into the cloud, some AWS services will handle much of the underlying infrastructure for you, allowing you to focus on your application needs. The benefits of a managed service are sometimes offset by premium pricing. But it's often well worth it.

One important example of such a managed service is the Relational Database Service (RDS). RDS, as you'll see in Chapter 9, "The Core Database Services," lets you set the basic configuration parameters for the database engine of your choice, gives you an endpoint address through which your applications can connect to the database, and takes care of all the details invisibly. Besides being responsible for making top-level configuration decisions, you won't need to worry about maintaining the database instance hosting the database, software updates, or data replication.

One step beyond a managed service—which handles only *one part* of your deployment stack for you—is a managed *deployment*, where the *whole stack* is taken care of behind the scenes. All you'll need to make things work with one of these services is your application code or, if it's a website you're after, your content. Until Amazon figures out how to secretly

listen in on your organization's planning meetings and then automatically convert your ideas to code without you knowing, things are unlikely to get any simpler than this.

Two AWS services created with managed deployments in mind are Amazon Lightsail and AWS Elastic Beanstalk.

Amazon Lightsail

Lightsail is promoted as a low-stress way to enter the Amazon cloud world. It offers *blueprints* that, when launched, will automatically provision all the compute, storage, database, and network resources needed to make your deployment work. You set the pricing level you're after (currently that'll cost you somewhere between $3.50 and $160 USD each month) and add an optional script that will be run on your instance at startup, and AWS will take over. For context, $3.50 will get you 512 MB of memory, 1 vCPU, a 20 GB solid-state drive (SSD) storage volume, and 1 TB of transfers.

Lightsail uses all the same tools—such as the AMIs and instances you saw earlier in the chapter—to convert your plans to reality. Since it's all AWS from top to bottom, you're also covered should you later decide to move your stack directly to EC2, where you'll have all the added access and flexibility standard to that platform.

Because things are packaged in blueprints, you won't have the unlimited range of tools for your deployments that you'd get from EC2 itself. But, as you can see from these lists, there's still a nice selection:

- **Operating systems:** Amazon Linux, Ubuntu, Debian, FreeBSD, OpenSUSE, and Windows Server
- **Applications:** WordPress, Magento, Drupal, Joomla, Redmine, and Plesk
- **Stacks:** Node.js, GitLab, LAMP, MEAN, and Nginx

In case you're curious, a LAMP stack is a web server built on Linux, Apache, MySQL (or MariaDB), and PHP (or Python). By contrast, MEAN is a JavaScript stack for dynamic websites consisting of MongoDB, Express.js, AngularJS (or Angular), and Node.js.

AWS Elastic Beanstalk

If anything, Elastic Beanstalk is even simpler than Lightsail. All that's expected from you is to define the application platform and then upload your code. That's it. You can choose between preconfigured environments (including Go, .NET, Java Node.js, and PHP) and a number of Docker container environments. The "code" for Docker applications is defined with specially formatted `Dockerrun.aws.json` files.

One key difference between the two services is that while Lightsail bills at a flat rate (between $3.50 and $160 per month, as you saw), Beanstalk generates costs according to how resources are consumed. You don't get to choose how many vCPUs or how much memory you will use. Instead, your application will scale its resource consumption

according to demand. Should, say, your WordPress site go viral and attract millions of viewers one day, AWS will invisibly ramp up the infrastructure to meet demand. As demand falls, your infrastructure will similarly drop. Keep this in mind, as such variations in demand will determine how much you'll be billed each month.

Deploying Container and Serverless Workloads

Even virtualized servers like EC2 instances tend to be resource-hungry. They do, after all, act like discrete, stand-alone machines running on top of a full-stack operating system. That means that having 5 or 10 of those virtual servers on a single physical host involves some serious duplication because each one will require its own OS kernel and device drivers.

Containers

Container technologies such as Docker avoid a lot of that overhead by allowing individual containers to share the Linux kernel with the physical host. They're also able to share common elements (called *layers*) with other containers running on a single host. This makes Docker containers fast to load and execute and also lets you pack many more container workloads on a single hardware platform.

You're always free to fire up one or more EC2 instances, install Docker, and use them to run as many containers as you'd like. But keeping all the bits and pieces talking to each other can get complicated. Instead, you can use either *Amazon Elastic Container Service* (ECS) or *Amazon Elastic Container Service for Kubernetes* (EKS) to orchestrate swarms of Docker containers on AWS using EC2 resources. Both of those services manage the underlying infrastructure for you, allowing you to ignore the messy details and concentrate on administrating Docker itself.

What's the difference between ECS and EKS? Broadly speaking, they both have the same big-picture goals. But EKS gets there by using the popular open source Kubernetes orchestration tool. They are different paths to the same place.

Serverless Functions

The serverless computing model uses a resource footprint that's even smaller than the one left by containers. Not only do *serverless functions* not require their own OS kernel, but they tend to spring into existence, perform some task, and then just as quickly die within minutes, if not seconds.

On the surface, Amazon's serverless service—AWS Lambda—looks a bit like Elastic Beanstalk. You define your function by setting a runtime environment (like Node.js, .NET, or Python) and uploading the code you want the function to run. But, unlike Beanstalk,

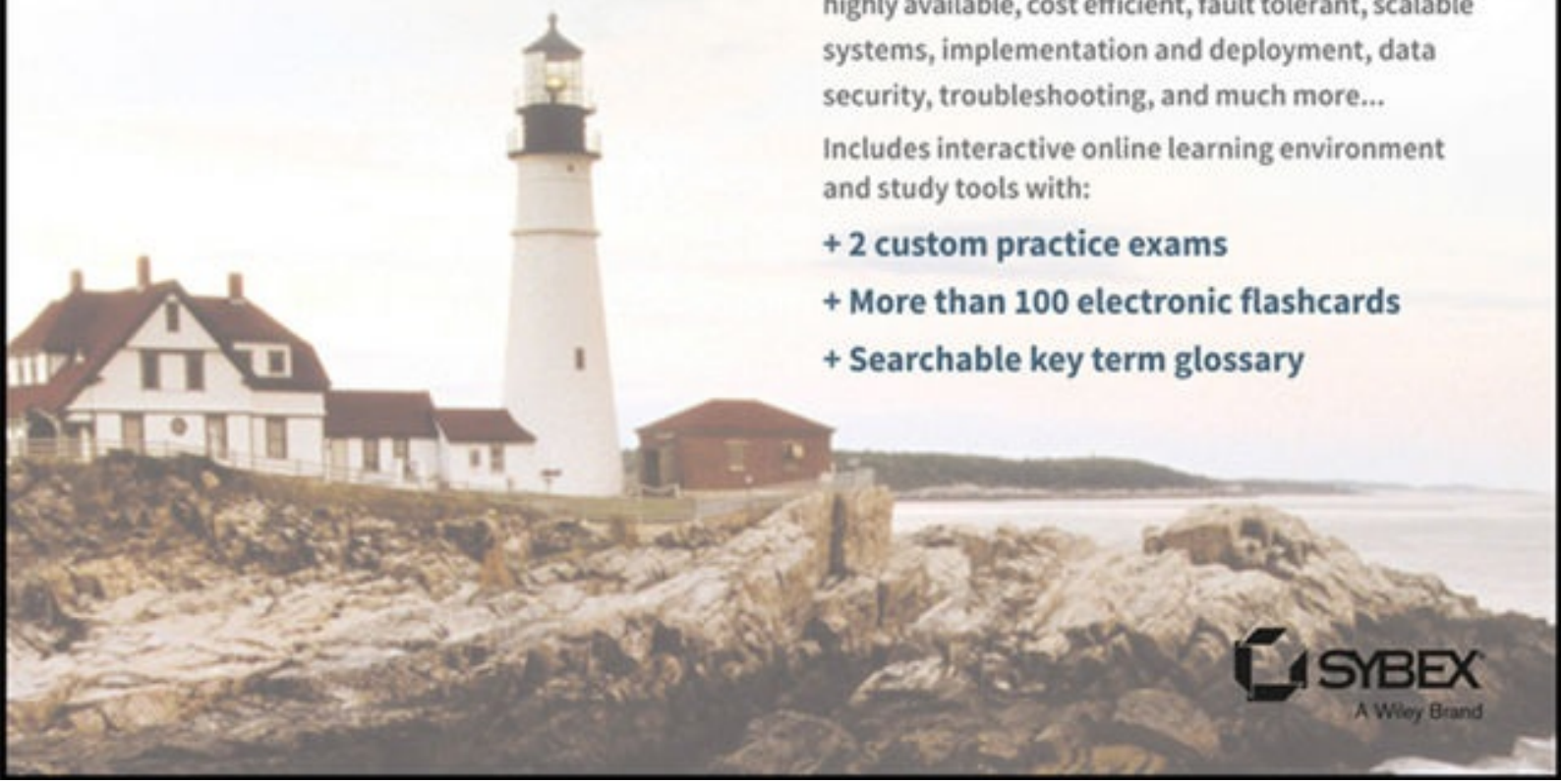
amazon web services

Joe Baron, Hisham Baz, Tim Bixler, Biff Gaut,
Kevin E. Kelly, Sean Senior, John Stamper

AWS Certified Solutions Architect

OFFICIAL STUDY GUIDE

ASSOCIATE EXAM



Covers exam objectives, including designing highly available, cost efficient, fault tolerant, scalable systems, implementation and deployment, data security, troubleshooting, and much more...

Includes interactive online learning environment and study tools with:

+ 2 custom practice exams
+ More than 100 electronic flashcards
+ Searchable key term glossary

SYBEX
A Wiley Brand

After all of the resources have been deleted, AWS CloudFormation signals that your stack has been successfully deleted. If AWS CloudFormation cannot delete a resource, the stack will not be deleted. Any resources that haven't been deleted will remain until you can successfully delete the stack.

Use Case

By allowing you to replicate your entire infrastructure stack easily and quickly, AWS CloudFormation enables a variety of use cases, including, but not limited to:

Quickly Launch New Test Environments AWS CloudFormation lets testing teams quickly create a clean environment to run tests without disturbing ongoing efforts in other environments.

Reliably Replicate Configuration Between Environments Because AWS CloudFormation scripts the entire environment, human error is eliminated when creating new stacks.

Launch Applications in New AWS Regions A single script can be used across multiple regions to launch stacks reliably in different markets.

AWS Elastic Beanstalk

AWS Elastic Beanstalk is the fastest and simplest way to get an application up and running on AWS. Developers can simply upload their application code, and the service automatically handles all of the details, such as resource provisioning, load balancing, Auto Scaling, and monitoring.

Overview

AWS comprises dozens of building block services, each of which exposes an area of functionality. While the variety of services offers flexibility for how organizations want to manage their AWS infrastructure, it can be challenging to figure out which services to use and how to provision them. With AWS Elastic Beanstalk, you can quickly deploy and manage applications on the AWS cloud without worrying about the infrastructure that runs those applications. AWS Elastic Beanstalk reduces management complexity without restricting choice or control.

There are key components that comprise AWS Elastic Beanstalk and work together to provide the necessary services to deploy and manage applications easily in the cloud. An *AWS Elastic Beanstalk application* is the logical collection of these AWS Elastic Beanstalk components, which includes environments, versions, and environment configurations. In AWS Elastic Beanstalk, an application is conceptually similar to a folder.

An *application version* refers to a specific, labeled iteration of deployable code for a web application. An application version points to an Amazon S3 object that contains the deployable code. Applications can have many versions and each application version is unique. In a running environment, organizations can deploy any application version they already uploaded to the application, or they can upload and immediately deploy a new application version. Organizations might upload multiple application versions to test differences between one version of their web application and another.

An *environment* is an application version that is deployed onto AWS resources. Each environment runs only a single application version at a time; however, the same version or different versions can run in as many environments at the same time as needed. When an environment is created, AWS Elastic Beanstalk provisions the resources needed to run the application version that is specified.

An *environment configuration* identifies a collection of parameters and settings that define how an environment and its associated resources behave. When an environment's configuration settings are updated, AWS Elastic Beanstalk automatically applies the changes to existing resources or deletes and deploys new resources depending on the type of change.

When an AWS Elastic Beanstalk environment is launched, the environment tier, platform, and environment type are specified. The environment tier that is chosen determines whether AWS Elastic Beanstalk provisions resources to support a web application that handles HTTP(S) requests or an application that handles background-processing tasks. An environment tier whose web application processes web requests is known as a *web server tier*. An environment tier whose application runs background jobs is known as a *worker tier*.

At the time of this writing, AWS Elastic Beanstalk provides platform support for the programming languages Java, Node.js, PHP, Python, Ruby, and Go with support for the web containers Tomcat, Passenger, Puma, and Docker.

Use Cases

A company provides a website for prospective home buyers, sellers, and renters to browse home and apartment listings for more than 110 million homes. The website processes more than three million new images daily. It receives more than 17,000 image requests per second on its website during peak traffic from both desktop and mobile clients.

The company was looking for ways to be more agile with deployments and empower its developers to focus more on writing code instead of spending time managing and configuring servers, databases, load balancers, firewalls, and networks. It began using AWS Elastic Beanstalk as the service for deploying and scaling the web applications and services. Developers were empowered to upload code to AWS Elastic Beanstalk, which then automatically handled the deployment, from capacity provisioning, load balancing, and Auto Scaling, to application health monitoring.

Because the company ingests data in a haphazard way, running feeds that dump a ton of work into the image processing system all at once, it needs to scale up its image converter fleet to meet peak demand. The company determined that an AWS Elastic Beanstalk worker fleet to run a Python Imaging Library with custom code was the simplest way to meet the requirement. This eliminated the need to have a number of static instances or, worse, trying to write their own Auto Scaling configuration.

By making the move to AWS Elastic Beanstalk, the company was able to reduce operating costs while increasing agility and scalability for its image processing and delivery system.

Key Features

AWS Elastic Beanstalk provides several management features that ease deployment and management of applications on AWS. Organizations have access to built-in Amazon CloudWatch monitoring metrics such as average CPU utilization, request count, and average

latency. They can receive email notifications through Amazon SNS when application health changes or application servers are added or removed. Server logs for the application servers can be accessed without needing to log in. Organizations can even elect to have updates applied automatically to the underlying platform running the application such as the AMI, operating system, language and framework, and application or proxy server.

Additionally, developers retain full control over the AWS resources powering their application and can perform a variety of functions by simply adjusting the configuration settings. These include settings such as:

- Selecting the most appropriate Amazon EC2 instance type that matches the CPU and memory requirements of their application
- Choosing the right database and storage options such as Amazon RDS, Amazon DynamoDB, Microsoft SQL Server, and Oracle
- Enabling login access to Amazon EC2 instances for immediate and direct troubleshooting
- Enhancing application security by enabling HTTPS protocol on the load balancer
- Adjusting application server settings (for example, JVM settings) and passing environment variables
- Adjust Auto Scaling settings to control the metrics and thresholds used to determine when to add or remove instances from an environment

With AWS Elastic Beanstalk, organizations can deploy an application quickly while retaining as much control as they want to have over the underlying infrastructure.

AWS Trusted Advisor

AWS Trusted Advisor draws upon best practices learned from the aggregated operational history of serving over a million AWS customers. AWS Trusted Advisor inspects your AWS environment and makes recommendations when opportunities exist to save money, improve system availability and performance, or help close security gaps. You can view the overall status of your AWS resources and savings estimations on the AWS Trusted Advisor dashboard.

AWS Trusted Advisor is accessed in the AWS Management Console. Additionally, programmatic access to AWS Trusted Advisor is available with the AWS Support API.

AWS Trusted Advisor provides best practices in four categories: cost optimization, security, fault tolerance, and performance improvement. The status of the check is shown by using color coding on the dashboard page, as depicted in Figure 11.10.

AWS®

Certified Developer

Official Study Guide

Associate (DVA-C01) Exam

which affect your deployment implementation. Consider the types of deployments that you will perform so that you can implement the most appropriate services into your seamless chain of events or a pipeline. This seamless or "continuous" chain of events on the AWS Cloud is the continuous integration/continuous deployment (CI/CD) pipeline.

Continuous Integration/Continuous Deployment

The CI/CD pipeline helps developers implement continuous builds, tests, and code deployments with multiple AWS resources and a continuous integration server. You can integrate AWS Elastic Beanstalk with the CI/CD pipeline as one of the deployment resources. You can also use AWS CodeCommit as a CI/CD resource paired with a Git repository, from which Elastic Beanstalk can extract and deploy code.

Continuous integration (CI) is the software development practice in which you continuously integrate (or check in) all code changes into a main branch of a central repository. This practice enables you to verify your code changes early and often with an automated build and test process. Whenever you check in code changes, engineers can automate various processes, such as building assets and testing code syntax. By implementing continuous integration practices, teams become more productive and develop new features more quickly. Teams also write scripts to validate the functionality and improve the quality of the software being released.

Continuous delivery (CD) is the software development practice in which all code changes are automatically prepared and always deployable (ready to go into production) at a single step.

Continuous delivery extends continuous integration to include testing production-like stages and running verification testing against those deployments. Although continuous delivery can extend to a production deployment, it requires manual intervention between a code check-in and when that code is available for customer use.

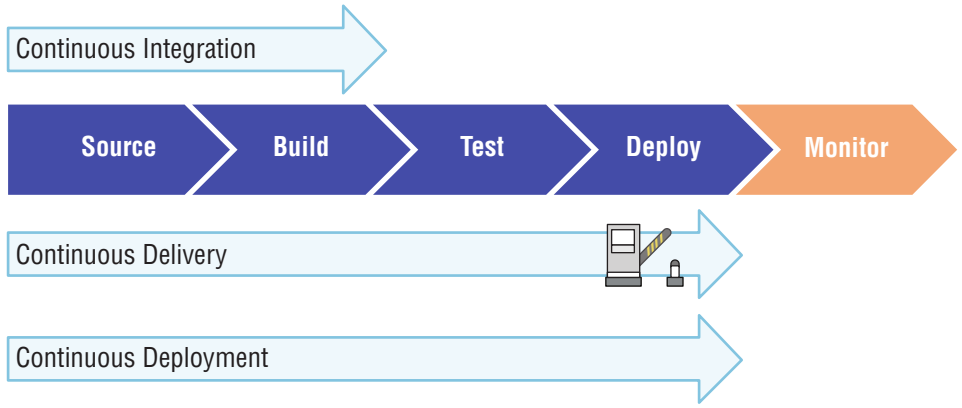
Practicing continuous delivery means that teams gain a greater level of certainty that their software will work in production.

Continuous deployment extends continuous delivery and is the automated release of software to customers, from check-in through production, without human intervention. Continuous deployment helps customers gain value quickly from the code base, with the development team getting faster feedback on the changes made.

An important distinction between continuous delivery and continuous deployment is that in continuous deployment, changes are automatically released to production after build/test stages; there is no manual approval step.

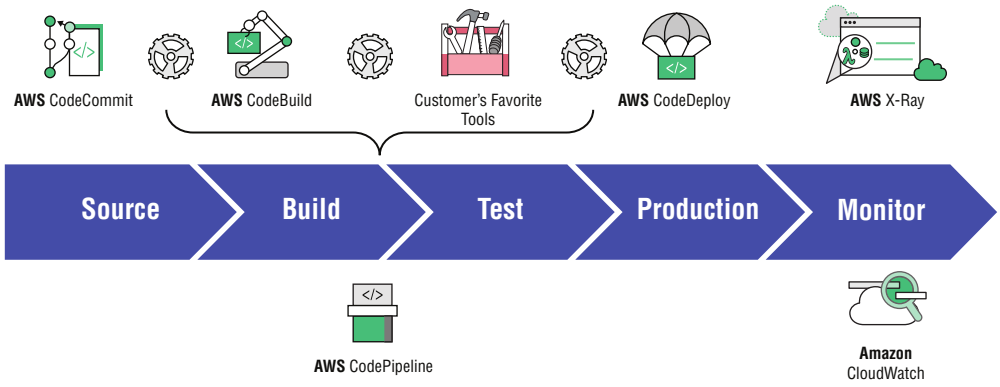
Figure 6.2 displays the CI/DI pipeline.

FIGURE 6.2 CI/DI pipeline



The CI/DI pipeline integrates with other AWS Code services, as illustrated in Figure 6.3.

FIGURE 6.3 AWS Code services



AWS CodePipeline *AWS CodePipeline* is a service for fast and reliable application updates. You can model and visualize the software release process. To build, test, and deploy your code every time there is a code change, integrate this service with third-party tools and AWS.

AWS CodeCommit *AWS CodeCommit* is a secure, highly scalable, managed source-control service that hosts private Git repositories. It enables you to store and manage assets (such as documents, source code, and binary files) privately in the AWS Cloud.

AWS CodeBuild *AWS CodeBuild* compiles source code, runs tests, and produces ready-to-deploy software packages. There is no need to manage build servers.

AWS CodeDeploy *AWS CodeDeploy* automates code deployments to any instance. It handles the complexity of updating your applications, which avoids downtime during application deployment. It deploys to Amazon EC2 or on-premises servers, in any language and on any operating system. It also integrates with third-party tools and AWS.

Deploying Highly Available and Scalable Applications

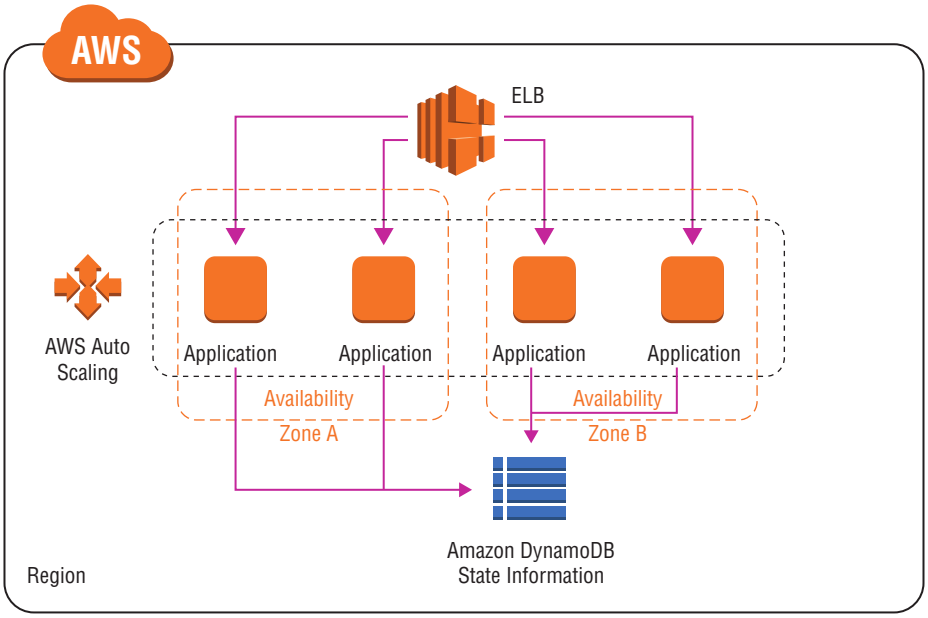
Load balancing is an integral part to directing and managing traffic among your instances. As you launch applications in your environments, you will want them to have high performance and high availability for your users. To enable both of these features, a load balancer will be necessary.

Elastic Load Balancing (ELB) supports three types of load balancers: Application Load Balancers, Network Load Balancers, and Classic Load Balancers. You select a load balancer based on your application needs.

- The *Application Load Balancer* provides advanced request routing targeted at delivery of modern application architectures, including microservices and container-based applications. It simplifies and improves the security of your application by ensuring that the latest Secure Sockets Layer (SSL)/Transport Layer Security (TLS) ciphers and protocols are used at all times. The Application Load Balancer operates at the request level (Layer 7) to route HTTP/HTTPS traffic to its targets: Amazon EC2 instances, containers, and IP addresses based on the content of the request. It is ideal for advanced load balancing of HTTP and HTTPS traffic.
- The *Network Load Balancer* operates at the connection level (Layer 4) to route TCP traffic to targets: Amazon EC2 instances, containers, and IP addresses based on IP protocol data. It is the best option for load balancing of TCP traffic because it's capable of handling millions of requests per second while maintaining ultra-low latencies. Network Load Balancer is optimized to handle sudden and volatile traffic patterns while using a single static IP address per Availability Zone. It is integrated with other popular AWS services, such as AWS Auto Scaling, Amazon Elastic Container Service (Amazon ECS), and AWS CloudFormation. Amazon ECS provides management for deployment, scheduling, and scaling, and management of containerized applications.
- The *Classic Load Balancer* provides basic load balancing across multiple Amazon EC2 instances and operates at both the request level and the connection level. The Classic Load Balancer is intended for applications that were built within the EC2-Classic network. When you're using Amazon Virtual Private Cloud (Amazon VPC), AWS recommends the Application Load Balancer for Layer 7 and Network Load Balancer for Layer 4).

Figure 6.4 displays the flow for deploying highly available and scalable applications.

FIGURE 6.4 Deploying highly available and scalable applications



The flow for deploying highly available and scalable applications includes the following components:

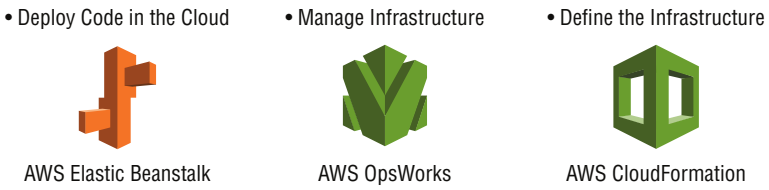
- Multiple Availability Zones and AWS Regions.
- Health check and failover mechanism.
- Stateless application that stores the session state in a cache server or database.
- AWS services that help you to achieve your goal. For example, Auto Scaling helps you maintain high availability and scalability.

Elastic Load Balancing and Auto Scaling are designed to work together.

Deploying and Maintaining Applications

AWS provides several services to manage your application and resources, as shown in Figure 6.5.

FIGURE 6.5 Deployment and maintenance services



With AWS Elastic Beanstalk, you do not have to worry about managing the infrastructure for your application. You deploy your application, such as a Ruby application, in a Ruby container, and Elastic Beanstalk takes care of scaling and managing it.

AWS OpsWorks is a configuration and deployment management tool for your Chef or Puppet resource stacks. Specifically, *OpsWorks for Chef Automate* enables you to manage the lifecycle of your application in layers with Chef recipes. It provides custom Chef cookbooks for managing many different types of layers so that you can write custom Chef recipes to manage any layer that AWS does not support.

AWS CloudFormation is infrastructure as code. The service helps you model and set up AWS resources so that you can spend less time managing them. It is a template-based tool, with formatted text files in JSON or YAML. You can create templates to define what AWS infrastructure you want to build and any relationships that exist among the parts of your AWS infrastructure.

Use AWS CloudFormation templates to provision and configure your stack resources.

Automatically Adjust Capacity

Use AWS Auto Scaling to monitor the AWS resources that are part of your application. The service automatically adjusts capacity to maintain steady, predictable performance. You can build scaling plans to manage your resources, including Amazon EC2 instances and Spot Fleets, Amazon Elastic Container Registry (Amazon ECR) tasks, Amazon DynamoDB tables and indexes, and Amazon Aurora Replicas.

AWS Auto Scaling makes scaling simple, with recommendations that allow you to optimize performance, costs, or balance between them. If you are already using EC2 Auto Scaling to scale your Amazon EC2 instances dynamically, you can now combine it with AWS Auto Scaling to scale additional resources for other AWS services. With AWS Auto Scaling, your applications have the right resources at the right time.

Auto Scaling Groups

An Auto Scaling group contains a collection of Amazon EC2 instances that share similar characteristics. This collection is treated as a logical grouping to manage the scaling of instances. For example, if a single application operates across multiple instances, you might want to increase the number of instances in that group to improve the performance of the application or decrease the number of instances to reduce costs when demand is low.

You can use the Auto Scaling group to scale the number of instances automatically based on criteria that you specify or maintain a fixed number of instances even if an instance becomes unhealthy. This automatic scaling and maintaining the number of instances in an Auto Scaling group make up the core functionality of the EC2 Auto Scaling service.

An Auto Scaling group launches enough Amazon EC2 instances to meet its desired capacity. The Auto Scaling group maintains this number of instances by performing periodic health checks on the instances in the group. If an instance becomes unhealthy, the group terminates the unhealthy instance and launches another instance to replace it.

You can use scaling policies to increase or decrease the number of instances in your group dynamically to meet changing conditions. When the scaling policy is in effect, the Auto Scaling group adjusts the desired capacity of the group and launches or terminates the instances as needed. You can also manually scale or scale on a schedule.

AWS Elastic Beanstalk

AWS Elastic Beanstalk is an AWS service that you can use to deploy applications, services, and architecture. It provides provisioned scalability, load balancing, and high availability. It uses common languages, including Java, .NET, PHP, Node.js, Python, Ruby, Go, and Docker, on common-type web servers, such as Apache, NGINX, Passenger, and IIS.

Elastic Beanstalk charges only for the resources you use to run your application.

Elastic Beanstalk is a solution that enables the automated deployments and management of applications on the AWS Cloud. Elastic Beanstalk can launch AWS resources automatically with Amazon Route 53, AWS Auto Scaling, Elastic Load Balancing, Amazon EC2, and Amazon Relational Database Service (Amazon RDS) instances, and it allows you to customize additional AWS resources.

Deploy applications without worrying about managing the underlying technologies, including the following:

Components

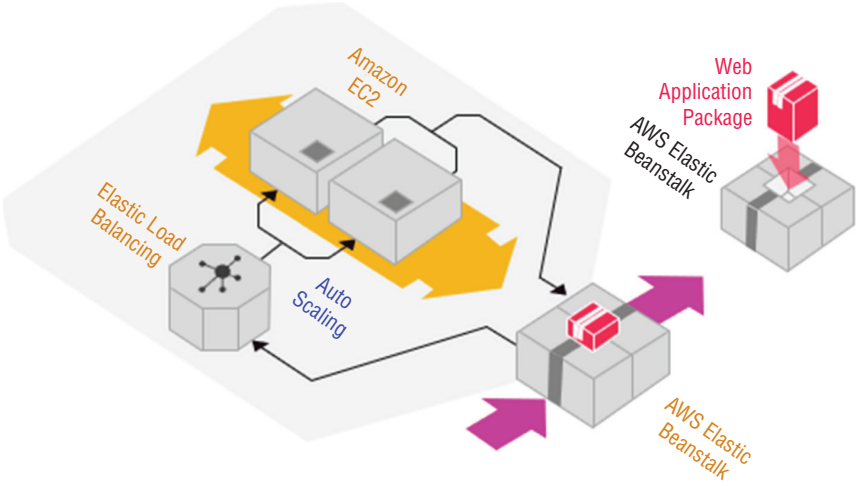
- Environments
- Application versions
- Environment configurations

Permission Model

- Service role
- Instance profile

Figure 6.6 displays the Elastic Beanstalk underlying technologies.

FIGURE 6.6 AWS Elastic Beanstalk underlying technologies



Elastic Beanstalk supports customization and N-tier architectures. It mitigates common manual configurations required in a traditional infrastructure deployment model. With Elastic Beanstalk, you can also create repeatable environments and reduce redundancy, thus rapidly updating environments and facilitating service-managed application stacks. You can deploy multiple environments in minutes and use various automated deployment strategies.

AWS Elastic Beanstalk allows you to focus on building your application.

Implementation Responsibilities

AWS and our customers share responsibility for achieving a high level of software component security and compliance. This shared model reduces your operational burden. The service you select determines the level of your responsibility. For example, Elastic Beanstalk helps you perform your side of the shared responsibility model by providing a managed updates feature. This feature automatically applies patch and minor updates for an Elastic Beanstalk supported platform version.

Developer Teams

Using AWS Elastic Beanstalk, you build full-stack environments for web and worker tiers. The service provides a preconfigured infrastructure.

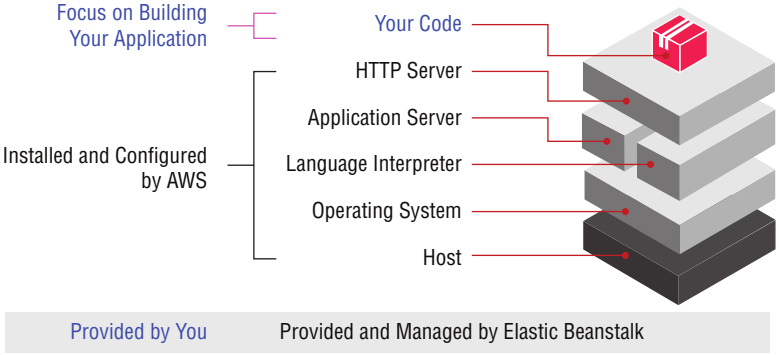
- Single-instance (development, low cost)
- Load balanced, AWS Auto Scaling (production)

Elastic Beanstalk Responsibilities

Elastic Beanstalk provisions the necessary infrastructure resources, such as the load balancer, Auto Scaling group, security groups, and database (optional). It also provides a unique domain name for your application (for example, `yourapp.elasticbeanstalk.com`).

Figure 6.7 displays Elastic Beanstalk responsibilities.

FIGURE 6.7 AWS Elastic Beanstalk responsibilities



Working with Your Source Repository

Developer teams generally begin their SDLC processes by managing their source code in a source repository. Uploading and managing the multiple changes on application source code is a repeated process. With Elastic Beanstalk, you can create an application, upload a version of the application as a source bundle, and provide pertinent information about the application.

The first step is to integrate Elastic Beanstalk with your source code to create your source bundle. As your source repository, you can install Git for your applications or use an existing repository and map your current branch from a local repository in Git to retrieve the source code.

Alternatively, you can use AWS CodeCommit as a source control system to retrieve source code. By using Elastic Beanstalk with the AWS CodeCommit repository, you extract from a current branch on CodeCommit.

To deploy a new application or application version, Elastic Beanstalk works with source bundles or packaged code. Prepare the code package with all of the necessary code dependencies and components.

Elastic Beanstalk can either retrieve the source bundle from a source repository or download the bundle from an Amazon Simple Storage Service (Amazon S3) bucket. You can use the IAM role to grant Elastic Beanstalk access to all services. The service accesses the source bundle from the location you designate, extracts the components from the bundle, deploys new application versions by launching the code, creates and

configures the infrastructure, and allocates the platform on Amazon EC2 instances to run the code.

The application runs on the resources and instances that the service generates. Your configuration for these resources and your application will become your environment settings, supporting the entire configuration of your deployment. Each deployment has an auto-incremented deployment identity (ID), so you are able to manage your multiple running deployments. Think of these as multiple running code releases in the AWS Cloud.

You can also work with different hosting services, such as GitHub or Bitbucket, with your code source.

Concepts

AWS Elastic Beanstalk enables you to manage all the resources that run your application as environments. This section describes some key Elastic Beanstalk concepts.

Application

Elastic Beanstalk focuses on managing your applications as environments and all of the resources to run them. Each application that launches in the service is a logical collection of environment variables and components, application versions, and environment configurations.

Application Versions

Application versions are iterations of the application's deployable code. Application versions in Elastic Beanstalk point to an Amazon S3 object with the code source package. An application can have many versions, with each version being unique. You can deploy and access any application version at any time. For example, you may want to deploy different versions for different types of tests.

Environment

Each Elastic Beanstalk environment is a separate version of the application, and that version's AWS Cloud components deploy onto AWS resources to support that version. Each environment runs one application version at a time, but you can run multiple environments, with the same application on each, along with its own customizations and resources.

Environment Tier

To launch an environment, you must first choose an environment tier. Elastic Beanstalk provisions the required resources to support both the infrastructure and types of requests

the application will support. The environment can launch and access other AWS resources. For example, it may pull tasks from Amazon Simple Queue Service (Amazon SQS) queues or store temporary configuration files in Amazon S3 buckets (according to your customizations). Each environment will then have an environment configuration—a collection of settings and parameters based on your customizations that define associated resources and how the environment will work.

Environment Configuration

You can change your environment to create, modify, delete, or deploy resources and change the settings for each. Your environment configuration saves to a configuration template exclusive to each environment and is accessible by either the Elastic Beanstalk application programming interface (API) calls or the service's command line interface (EB CLI).

In Elastic Beanstalk, you can run either a web server environment or a worker environment. Figure 6.8 displays an example of a web server environment running in Elastic Beanstalk with Amazon Route 53 as the domain name service (DNS) and ELB to route traffic to the web server instances.

FIGURE 6.8 Application running on AWS Elastic Beanstalk

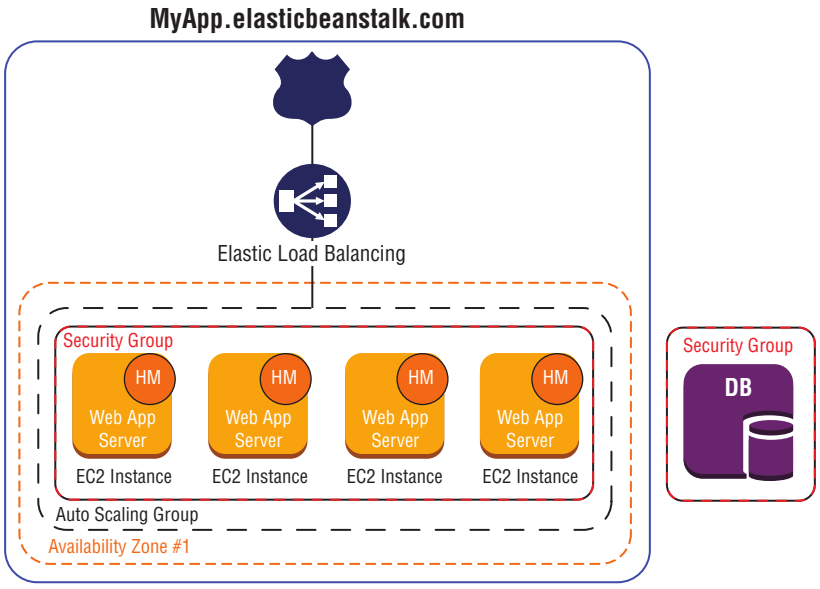
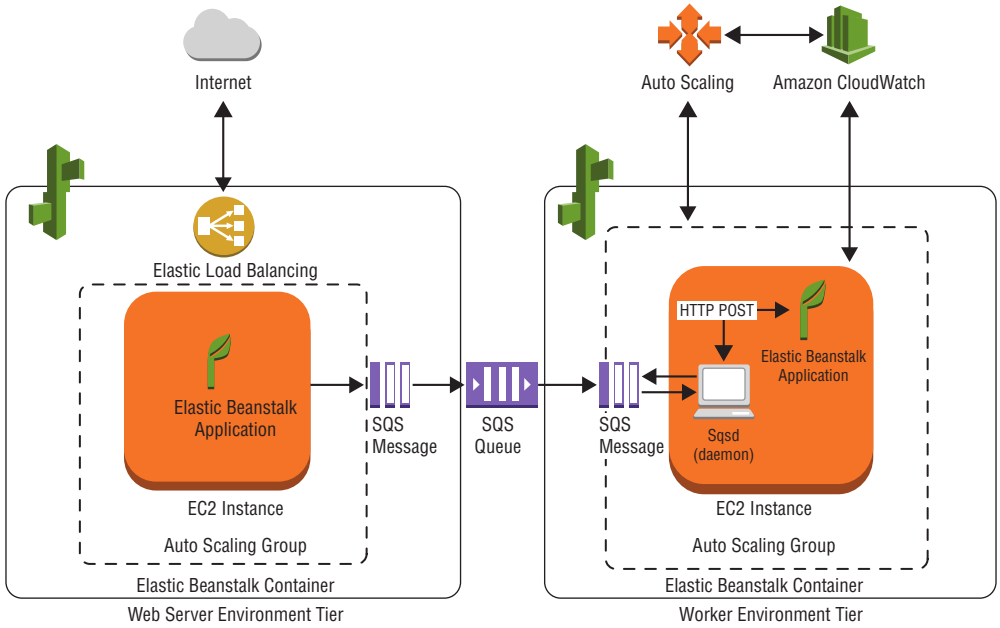


Figure 6.9 shows a worker environment architecture, where AWS resources create configurations, such as Auto Scaling groups, Amazon EC2 instances, and an IAM role, to manage resources for your worker applications.

FIGURE 6.9 Worker tier on AWS Elastic Beanstalk



For the worker environment tier, Elastic Beanstalk creates and provisions additional resources and files to support the tier. This includes services like Amazon SQS queues operating between worker applications, AWS Auto Scaling groups, security groups, and EC2 instances.

The worker environment infrastructure uses all of your customization and provision resources to determine the types of requests it receives.

Docker Containers

You can also use Docker containers with Elastic Beanstalk to run your applications from a container. Install Docker, choose the software you require, and select the Docker images you want to launch. Define your runtime environment, platform, programming language, and application dependencies and tools. Docker containers are self-contained and include configurations and software that you specify for your application to run. Each Docker container restarts automatically if another container crashes. When you choose to deploy your applications with Docker containers, your infrastructure is provisioned with capacity provisioning, load balancing, scaling, and health monitoring, much like a noncontainer environment. You can continue to manage your application and the AWS resources you use.

Docker requires platform configurations that enable you to launch single or multicontainer deployments. A single container deployment launches a single Docker image, and your application uses a single container configuration for a single Amazon EC2 instance.

A multicontainer deployment uses the Amazon ECS to launch a cluster of containers with Docker images. A multicontainer configuration is applied to each instance. You can also run preconfigured Docker platform configurations with generic customization for popular software stacks that you want to use for your application.

AWS Elastic Beanstalk Command Line Interface

Elastic Beanstalk has its own command line interface separate from the AWS CLI tool. To create deployments from the command line, you download and install the AWS Elastic Beanstalk CLI (EB CLI).

Table 6.1 lists common EB CLI commands.

TABLE 6.1 Common AWS Elastic Beanstalk Commands

Command	Definition
`eb init application-name`	Sets default values for Elastic Beanstalk applications with the EB CLI configuration wizard
`eb create`	Creates a new environment and deploys an application version to it
`eb deploy`	Deploys the application source bundle from the initialized project directory to the running application
`eb clone`	Clones an environment to a new environment so that both have identical environment settings
`eb codesource`	Configures the EB CLI to deploy from an AWS CodeCommit repository, or disables AWS CodeCommit integration and uploads the source bundle from your local machine

Customizing Environment Configurations

You can use Elastic Beanstalk to customize the platforms used to support your application and your infrastructure. To do so, create a configuration file in the `ebextensions` directory (or `.ebextensions`) to include with your web application's source code. The configuration file allows for simple and advanced customizations of your environment and contains settings for your AWS resources. To deploy customized resources to support your application source bundle, use YAML to configure the file.

The configuration file has several sections. The `option_settings` section defines your configuration option values for your AWS resources. The `resources` section adds further customization in your application environment beyond the service functionality, which includes AWS CloudFormation–supported resources that Elastic Beanstalk can access and

run. The remaining sections allow for fine-grained configurations to integrate packages, sources, files, and container commands.

Launch environments from integrated development environment (IDE) tools to avoid poorly formatted configurations and source bundles that could cause unrecoverable failures.

You apply configuration files in the `ebextensions` directory to Elastic Beanstalk stacks. The stacks are the AWS resources that you allocate for your infrastructure and application. If you have any resource, such as Amazon VPC, Amazon EC2, or Amazon S3, that was updated or configured, these files deploy with your changes. You can zip your `ebextension` files, upload, and apply them to multiple application environments. You can view your environment variables in `option_settings` for future evaluation or changes. These are accessible from the AWS Management Console, command line, and API calls.

You can view Elastic Beanstalk stacks in AWS CloudFormation, but always use the Elastic Beanstalk service and `ebextensions` to make modifications. This way, edits and modifications to the application stacks are simplified without introducing unrecoverable failures.

Elastic Beanstalk generates logs that you can view to troubleshoot your environments and resources. The logs display Amazon EC2 operational logs and logs that are specific to servers running for your applications.

Integrating with Other AWS Services

Elastic Beanstalk automatically integrates or manages other AWS services with application code to provision efficient working environments. However, you might find it necessary to add additional services, such as Amazon S3 for content storage or Amazon DynamoDB for data records, to work with an environment. To grant access between any integrated service and Elastic Beanstalk, you must configure permissions in IAM.

Amazon S3

You can use Amazon S3 to store static content you want to integrate with your application and point directly to objects you store in Amazon S3 from your application or from other resources. In addition to setting permissions in IAM policies, take advantage of presigned URLs for controlled Amazon S3 `GET` and `PUT` operations.

Amazon CloudFront

You can integrate your Elastic Beanstalk environment with Amazon CloudFront, which provides content delivery and distribution through the use of edge locations throughout the world. This can decrease the time in which your content is delivered to you, as the content

is cached and routed through the closest edge location serving you. After you deploy your application on Elastic Beanstalk, use the Amazon CloudFront content delivery network (CDN) to cache static content from your application. To identify the source of your content in Amazon CloudFront, you can use URL path patterns to cache your content and then retrieve it from the cache. This approach serves your content more rapidly and offloads requests directly sourced from your application.

AWS Config

With *AWS Config*, you can visualize configuration history and how configurations evolve over time. Tracking changes helps you to fulfill compliance obligations and meet auditing requirements. You can integrate AWS Config directly with your application and its versions or your Elastic Beanstalk environment. *You can customize AWS Config to record changes per resource, per region, or globally.* In the AWS Config console, you can select Elastic Beanstalk resource types to record specific applications and environment resources. You can view the recorded information in the AWS Config dashboard under Resource Inventory.

Amazon RDS

Various options are available for creating databases for your environment, such as Amazon Relational Database Service (Amazon RDS) for SQL databases and Amazon DynamoDB for NoSQL databases. Elastic Beanstalk can create a database and store and retrieve data for any of your environments. Each service has its own features to handle scaling, capacity, performance, and availability.

To store, read, or write to your data records, you can set up an Amazon RDS database instance or an Amazon DynamoDB table by using the same configuration files for your other service option settings. You must create connections to the database, which require you to set up password management in Elastic Beanstalk. Your configurations are saved in the `ebextensions` directory. You can also create direct connections, within your application code or application configuration files, to both internal and external databases. When using Amazon RDS, avoid accidentally deleting and re-creating databases without a properly installed backup. To reduce the risk of losing data, take a manual snapshot of the master Amazon RDS database immediately before deleting.

If you create periodic tasks with a worker environment, Elastic Beanstalk automatically creates an Amazon DynamoDB table to perform leader election and stores task information.

Amazon ElastiCache

For caching capabilities, you can integrate Amazon ElastiCache service clusters with the Elastic Beanstalk environment. If you use a nonlegacy container, you can set your configuration files to use the supported container and then offload requests to the cache cluster.

Doing so enables you to increase the performance of your application and databases running in your Elastic Beanstalk environment.

AWS Identity and Access Management Roles

Elastic Beanstalk integrates with AWS Identity and Access Management (IAM) roles to enable access to the services you require to run your architecture.

When you launch the service to create an environment, a default service role and instance profile are created for you through the service API. Managed policies for resources permissions are also attached, including policies for Elastic Beanstalk instance health monitoring within your infrastructure and platform updates that can be made on behalf of the service. These policies, called `AWSElasticBeanstalkEnhancedHealth` and `AWSElasticBeanstalkService`, attach to the default service role and enable the default service role to specify a trusted entity and trust policy.

When you use commands from the EB CLI, the role allows automatic management of the AWS Cloud that services you run. The service creates an environment, if you don't identify it specifically; creates a service-linked role; and uses it when you spin up a new environment. To create the environment successfully, the `CreateServiceLinkedRole` policy must be available in your IAM account.

You use IAM roles to automate the management of allocated services for your application through Elastic Beanstalk. With IAM, you can also launch code with inline policies. It is important to understand how the service creates and uses the roles to keep your application and data secure.

For IAM to manage the policies for the account better, create policies at the account level.

Deployment Strategies

A *deployment* is the process of copying content and executing scripts on instances in your deployment group. To accomplish this, AWS CodeDeploy performs the tasks outlined in the `AppSpec` configuration file. For both Amazon EC2 on-premises instances and AWS Lambda functions, the deployment succeeds or fails based on whether individual `AppSpec` tasks complete successfully.

After you have created a deployment, you can update it as your application or service changes. You can update a deployment by adding or removing resources from a deployment, thus updating the properties of existing resources in a deployment.

A serverless application is typically a combination of AWS Lambda and other AWS services.

To create seamless deployments, choose an effective deployment strategy. Each strategy has specific advantages relative to different use cases. Appropriate strategies help create deployments where you experience minimal or no downtime, and you can apply the strategy for different purposes within your environments. Each change needs a strategy that best fits your application deployments.

All-at-Once and In-Place Deployments

An *all-at-once deployment* applies updates to all your instances at once. When you execute this strategy, you experience downtime, as all instances receive the change at the same time.

This is an appropriate strategy for simple, immediate update requirements when it's not critical to have your application always available, and you're comfortable with the site being offline for a short duration. To enable all-at-once updates, set a deployment policy either in the AWS Management Console or in the command line (`DeploymentPolicy`).

When you perform an *in-place deployment*, AWS CodeDeploy stops currently running applications on the target instance, deploys the latest revision, restarts applications, and validates successful deployment. In-place deployments can support the automatic configuration of a load balancer. In this case, the instance is deregistered from the load balancer before deployment and registered again after the deployment processes successfully.

In-place updates are also available for your platform updates, such as a coding-language platform update for a web server. Select the new platform and then run the update from the AWS Management Console or command line directly as a platform update.

AWS Lambda does not support in-place deployments.

Rolling Deployments

A *rolling deployment* applies changes to all of your instances by rolling the updates from one instance to another. Elastic Beanstalk can deploy configuration changes in batches. This approach reduces possible downtime during implementation of the change and allows available instances to run while you deploy.

As updates are applied in a batch, the batch will be out of service for a short period while the changes propagate and then relaunch with the new configuration. When the change is complete, the service moves on to the next batch of instances to apply the changes. With this strategy, you can implement both periodic changes and pauses between updates. For example, you might specify a time to wait between health-based updates so that instances must pass health checks before moving on to the next batch. If the rolling update fails, the service begins another rolling update for a rollback to the previous configuration.

Rolling updates include changes for Auto Scaling group configurations, Amazon EC2 instance configurations, and Amazon VPC settings. It is an effective method for updating an application version on fleets of instances through the Elastic Beanstalk service. To enable

rolling updates, set a deployment policy either in the AWS Management Console or in the command line (`DeploymentPolicy`) and choose this strategy along with specific options. You can select *Rolling* or *Rolling with additional batch*. By using *Rolling with additional batch*, you can launch a new batch of instances before you begin to take instances out of service for your rolling updates. This option provides an available batch for rollback from a failed update. After the deployment is successfully executed, Elastic Beanstalk terminates the instances from the additional batch. This is helpful for a critical application that must continue running with less downtime than the standard rolling update.

Blue/Green Deployment

When high availability is critical for applications, you may want to choose a *blue/green deployment*, where your newer environment will be separate from your existing environment. The running production environment is considered the *blue environment*, and the newer environment with your update is considered the *green environment*. When your changes are ready and have gone through all tests in your green environment, you can swap the CNAMEs of the environments to redirect traffic to the newer running environment. This strategy provides an instantaneous update with typically zero downtime.

When you deploy to AWS Lambda functions, blue/green deployments publish new versions of each function. Traffic shifting then routes requests to the new functioning versions according to the deployment configuration you define.

If your infrastructure contains Amazon RDS database instances, the data does not automatically transfer to the new environment. Without performing backups, you will experience data loss when you use the blue/green strategy. If you have Amazon RDS instances in your infrastructure, implement a different deployment strategy or a series of steps to create snapshot backups outside of Elastic Beanstalk before you execute this type of deployment.

Immutable Deployment

An *immutable deployment* is best when an environment requires a total replacement of instances, rather than updates to an existing part of an infrastructure. This approach implements a safety feature for updates and rollbacks. Elastic Beanstalk creates a temporary Auto Scaling group behind your environment's load balancer to contain the new instances with the updates you apply. If the update fails, the rollback process terminates the Auto Scaling group. Immutable instances implement a number of health checks. If all instances pass these checks, Elastic Beanstalk transfers the new configurations to the original Auto Scaling group, providing an additional check before you apply your changes to other instances. Enhanced health reports evaluate instance health in the update. After the updates are made, Elastic Beanstalk deletes the temporary Auto Scaling group of the older instances.

During this type of deployment, your capacity doubles for a short duration between the updates and terminations of instances. Before you use this strategy, verify that your instances have a low on-demand limit and enough capacity to support immutable updates.

See Table 6.2 for feature comparisons between all deployment strategies. The check mark indicates options that the deployment strategy supports.

TABLE 6.2 Deployment Strategies

Method	Impact of Failed Deployment	Deploy Time	Zero Downtime	No DNS Change	Rollback Process	Code Deployed To
All-at-once	Downtime	🕒		✓	Redeploy	Existing instances
In-place	Downtime	🕒		✓	Redeploy	Existing instances
Rolling	Single batch out of service; any successful batches before failure running new application version	🕒🕒	✓	✓	Redeploy	Existing instances
Rolling with additional batch	Minimal if first batch fails; otherwise, similar to Rolling	🕒🕒🕒	✓	✓	Redeploy	New and existing instances
Blue/Green	Minimal	🕒🕒🕒🕒	✓		Swap URL	New instances
Immutable	Minimal	🕒🕒🕒🕒	✓	✓	Redeploy	New instances

Container Deployments

Elastic Beanstalk enables you to launch your applications with Docker containers. With a Docker container, you can create a runtime environment with all of the dependencies, packages, and tools that your application may require to run. Your container can have all of the configurations necessary for your application. By using Docker with Elastic Beanstalk, you have the infrastructure for capacity provisioning, scalability, load balancing, and health monitoring for the instances that run on containers. The containers integrate with your

Amazon VPC for network requirements and with IAM to enable resource management. You can launch different software engines with containers to provide various options and third-party tools to run containers.

You can choose from single container configurations and multicontainer configurations. A single container runs one container per instance. A multicontainer runs multiple applications or engines on one instance, with all of the software and settings you require. Preconfigured options are available with Docker, and you can integrate them with instances that run in your architecture through Elastic Beanstalk.

Monitoring and Troubleshooting

After you launch your code, check on its performance and availability. You can monitor statistics and view information about the health of your application, its environment, and specific services from the AWS Management Console. Elastic Beanstalk also creates alerts that trigger at established thresholds to monitor your environment's health. In the AWS Management Console, the AWS Elastic Beanstalk Monitoring page shows aggregated statistics and graphs for your applications and resources. Each environment is color-coded to indicate the environment's status. You can see at a glance whether your environment is available online at any point in time. Metrics gathered by the resources in your environment are published to Amazon CloudWatch in five-minute intervals. You can adjust the time range for the statistics and graphs and customize your views of the metrics.

Figure 6.10 shows an example of the statistics that you can view for your environment.

FIGURE 6.10 Health dashboard on AWS Elastic Beanstalk

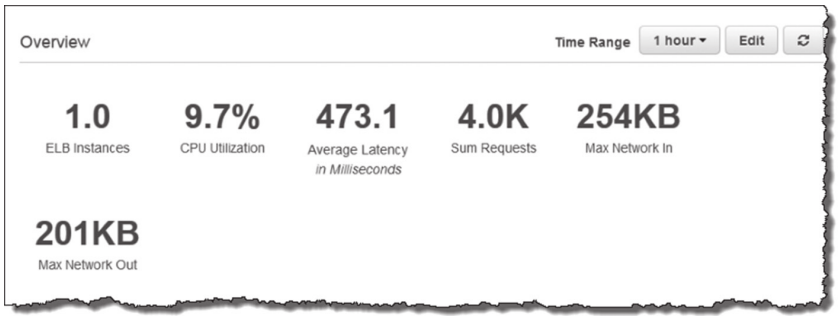


Figure 6.11 shows an example of the graphs that you can view.

FIGURE 6.11 Metrics for monitoring on AWS Elastic Beanstalk

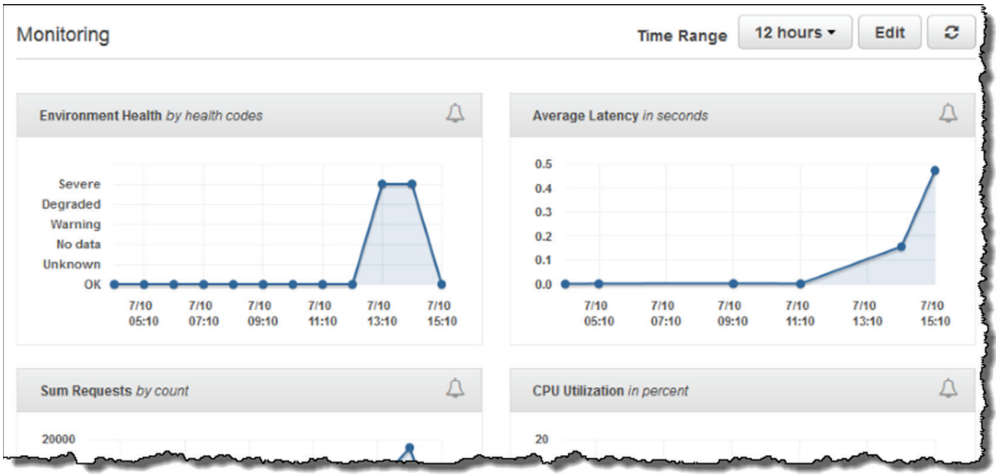


Table 6.3 defines the AWS Elastic Beanstalk Monitoring page colors.

TABLE 6.3 AWS Elastic Beanstalk Health Page Color Definitions

Color	Description
Gray	Your environment is being updated.
Green	Your environment has passed the most recent health check. At least one instance in your environment is available and taking requests.
Yellow	Your environment has failed one or more health checks. Requests to your environment are failing.
Red	Your environment has failed three or more health checks, or an environment resource has become unavailable. Requests are consistently failing.

By default, Elastic Beanstalk displays Amazon EC2, Auto Scaling, and Elastic Load Balancing metrics for your application environments. These metrics are available to you on your AWS Elastic Beanstalk Monitoring page as soon as you deploy your application environment. You can access the health status from the AWS Management Console or the EB CLI.

Basic Health Monitoring

To access the health status from the AWS Management Console, select the Elastic Beanstalk service and then select the tab for your specific application environment. An

environment overview shows your architecture's instance status details, resource details, and filter capabilities. Health statuses are indicated in four distinct colors.

To access the health status from the EB CLI, enter the `eb health` command. The output shows the environment and the health of associated instances. Enhanced health reporting also provides the following seven health statuses, which are single-word descriptors that provide a better indication of the state of your environment:

```
ok   warning   degraded   severe   info   pending   unknown
```

You can also use the `eb status` command in the EB CLI or the `DescribeEnvironments` API call to retrieve the health status for an environment. You can check the health of the overall environment or the individual services of Amazon EC2 or an Elastic Load Balancing load balancer. Health checks on your Elastic Load Balancing port execute both for the default port 80 and a custom Elastic Load Balancing port/path.

For `GET` requests with the load balancer, `200 OK` is the default success code and indicates a healthy status. The service can also return `400` level responses. You can also configure a health check URL for custom static page responses.

Be sure to adjust the caching time to live for any health check static pages or URLs in Amazon CloudFront or for any caching mechanism you may use.

Elastic Beanstalk also reports missing configurations or other issues that could affect the health of the application environment.

Enhanced Health Monitoring

There are two types of reporting: the default health information about your resources and the enhanced health reporting that provides you more information for monitoring health.

You can use the enhanced health reporting feature to gather additional resource data and display graphs and statistics of environment health in greater detail. This is important when you deploy multiple versions of your application and when you need to analyze factors that could be degrading your application's availability or performance. You can view these details in the AWS Elastic Beanstalk Monitoring page from the AWS Management Console. These reports require the creation of two IAM roles: a *service role* to allow access between the services and Elastic Beanstalk and an *instance profile* to write logs into an Amazon S3 bucket.

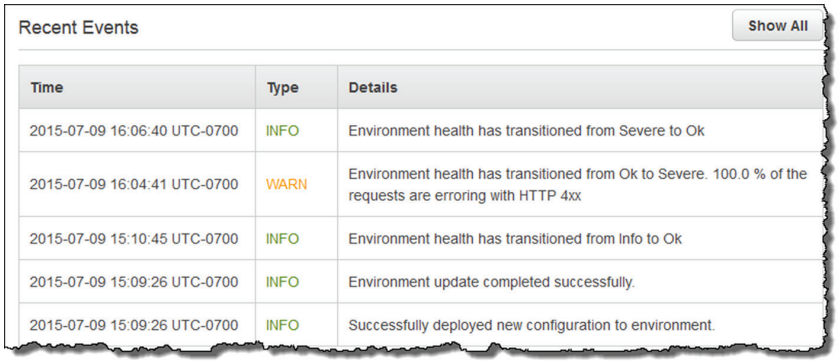
Running the enhanced health report requires a version 2 or newer platform configuration that supports all platforms except Windows Server with IIS. The enhanced health reports provide data directly to Elastic Beanstalk and do not run through Amazon CloudWatch.

By default, health monitoring on Elastic Beanstalk does not publish metrics to Amazon CloudWatch, so you are not charged for the metrics. There are also custom metrics that you can run and view, for which you are not charged a fee. You can enable custom metrics by using the `PutMetricData` operation in worker environments. For example, you might have an Amazon SQS daemon that publishes custom metrics for environment health under the same environment namespace. You can also enable custom metrics from Amazon CloudWatch, but AWS charges for these additional metrics you publish to your monthly Amazon CloudWatch. To save costs, use the available metrics on the Elastic Beanstalk service, or enable the custom metrics that you need, paying only for what you use.

Elastic Beanstalk runs a health agent to provide detailed health resource data for enhanced health monitoring. The health agent runs in the Amazon Machine Image (AMI) for each instance operating system on a platform configuration for your application. The agent analyzes system metrics and logs to communicate the health status to Elastic Beanstalk. You receive alerts, data, and actionable insights that you can use to monitor your applications and understand, prevent, and respond to performance issues.

You can monitor recent health events that you have enabled on Elastic Beanstalk in real time. There are several health event types that can change as an environment transitions from the create state to the run state. Figure 6.12 displays the health events available in the AWS Elastic Beanstalk Monitoring page and examples of the details that allow you to respond to issues identified.

FIGURE 6.12 Events on AWS Elastic Beanstalk



Elastic Beanstalk integrates with AWS CloudTrail to capture Elastic Beanstalk API calls as log files that you can store in an Amazon S3 bucket. To view additional actions occurring with your running resources, you can also capture AWS API calls in your code using AWS CloudTrail.

Summary

In this chapter, you learned about the features of Elastic Beanstalk, how to automate deployments for your multi-tier architectures, and different deployment strategies. You also discovered options for configuring your environments and managing your resources with services such as IAM, Amazon VPC, Amazon EC2, and Amazon S3.

Exam Essentials

Know how to deploy AWS Elastic Beanstalk. Know how to deploy an application AWS Elastic Beanstalk and what platforms it supports. To complete the exam successfully, you should also understand how the architectures and services interact with the web, application, and database tiers. Focus on foundational services and how you create and work with Elastic Beanstalk.

Know about `ebextensions`**.** Understand `ebextensions` and the part they play in the service configuration. Be able to recognize the stacks you create and how to change them.

Know about Elastic Beanstalk resources. Understand how to manage resources with Elastic Beanstalk, including IAM. Understand the definitions and differentiate between the functions of the default IAM service role and the instance profile, which are automatically created. Understand permissions for your AWS resources in your environment.

Know Elastic Beanstalk deployment strategies. Understand what deployment strategies you can use, their differences, and which ones would be best for different use cases and other resources. Know which strategy offers less downtime and which is best suited for complex changes.

Know about Elastic Beanstalk components. Understand all of the components of Elastic Beanstalk, including applications, environments, versions, configurations, and the AWS resources it launches and with which it integrates. Know how to retain or dispose of resources as needed.

Know about Elastic Beanstalk different environment tiers. Know the differences between the single-instance tier and the web-server environment tier and when to choose one over the other. Understand the services and features used for both.

On the test itself, do not get sidetracked with small details about Elastic Beanstalk. Focus your understanding on how it works as a whole and interacts with other services.

TABLE 7.1 AWS CodePipeline Service Limits

Limit	Value
Pipelines per region	US East (N. Virginia) (`us-east-1`): 40 US West (Oregon) (`us-west-2`): 60 EU (Ireland) (`eu-west-1`): 60 Other supported regions: 20
Stages per pipeline	Minimum: 2 Maximum: 10
Actions per stage	Minimum: 1 Maximum: 20
Parallel actions per stage	Maximum: 10
Sequential actions per stage	Maximum: 10
Maximum artifact size	Amazon S3 source: 2 GB AWS CodeCommit source: 1 GB GitHub source: 1 GB

When you deploy to AWS CloudFormation, the maximum artifact size is 256 MB.

AWS CodePipeline Tasks

The remainder of this section will focus on the tasks you need to build and execute a simple pipeline and how to outline the requirements to build cross-account pipelines. This concept is particularly important for organizations that have multiple AWS accounts, especially when you separate environments across accounts (such as Account A for development, Account B for Quality Assurance [QA], and Account C for production), as AWS CodePipeline will need access to resources in each account to automate deployments successfully.

Before you start the next steps, make sure that you have an IAM user with an access key and secret access key and that the user has sufficient AWS CodePipeline permissions.

Create an AWS CodePipeline

It is best to name your pipeline something meaningful, such as Dev_S3_Bucket. After you select a source provider (Amazon S3, AWS CodeCommit, or GitHub), you must enter a full object path. This corresponds to the `.zip` archive that will be tracked for changes.

When you select Amazon S3, AWS CodePipeline creates an Amazon CloudWatch Events rule, IAM role, and AWS CloudTrail trail. These are the default methods that notify AWS CodePipeline of changes to the source archive. You can also use AWS CodePipeline to check regularly for changes. This, however, will provide a slower update experience.

You select AWS CodeBuild, Jenkins, or Solano CI for a build provider.

The Jenkins build provider requires you to install the AWS CodePipeline plugin for Jenkins on the server.

The Solano CI build provider requires authentication to GitHub with a valid user. After authenticating to GitHub, you must authenticate to Solano CI.

If you do not select a build provider, you must select a deployment provider (if you select a build provider, the deployment step is optional). This option is useful if you desire the pipeline execution to be a finished build artifact, such as the case with custom media transcoding with AWS CodeBuild. The available providers for the deployment stage are Amazon ECS, AWS CloudFormation, AWS CodeDeploy, AWS Elastic Beanstalk, and AWS OpsWorks Stacks.

AWS Elastic Beanstalk allows customers to automate deployment of application archives to one or more Amazon EC2 instances. It also handles health checks, load balancing, log gathering, and other important tasks automatically. Since it requires a bundled application archive to upload to instances for deployment, it is a natural fit for AWS CodePipeline, which provides artifacts as archives. To deploy to AWS Elastic Beanstalk from AWS CodePipeline, simply provide the application and environment name.

For deployment to AWS Elastic Beanstalk, the maximum application archive size is 512 MB. The deployment artifact must not exceed this size, or the deployment will fail.

You select a service role for AWS CodePipeline to access AWS resources within your account. You can select an existing IAM role or create a new role.

You can only select IAM roles with a trust policy that allows AWS CodePipeline to assume them.

Start a Pipeline

After you create a pipeline, the first stage updates the source repository or archive, and then the pipeline will automatically begin execution. To rerun the pipeline for the most recent

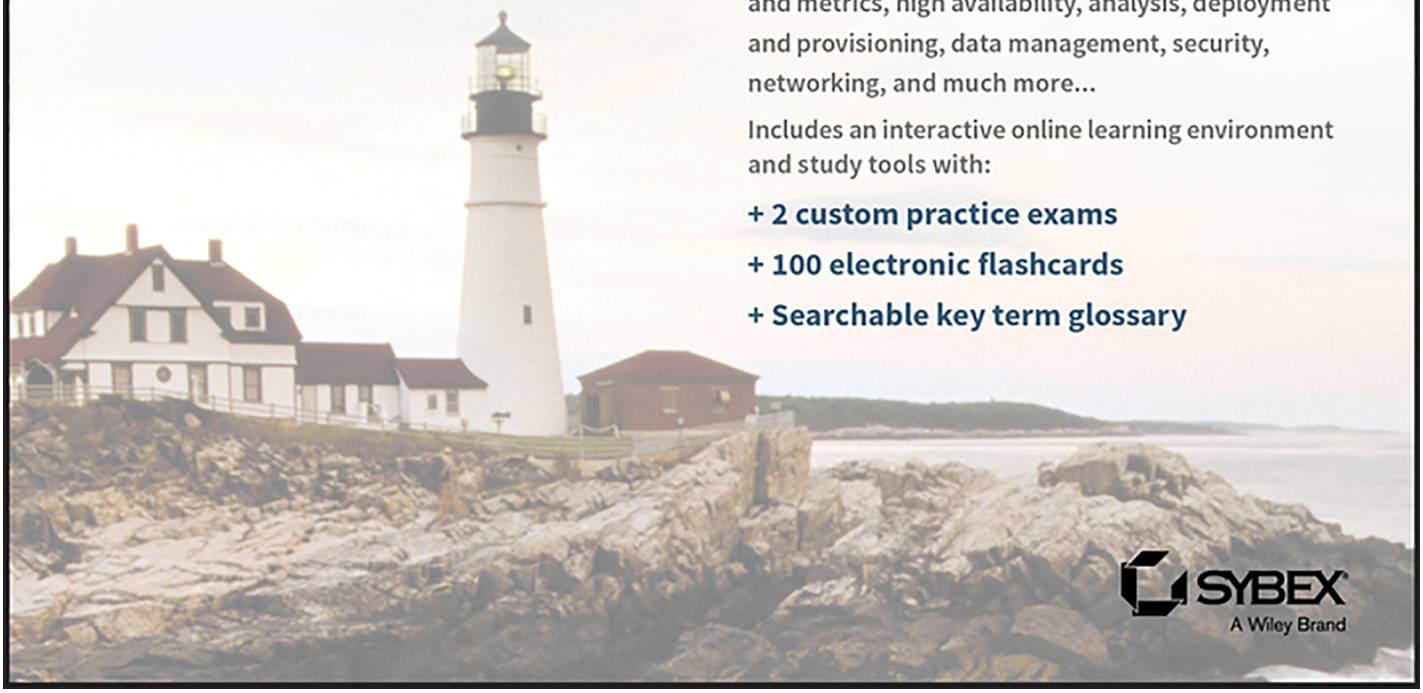
amazon web services

Stephen Cole, Gareth Digby, Chris Fitch,
Steve Friedberg, Shaun Qualheim, Jerry Rhoads,
Michael Roth, Blaine Sundrud

AWS Certified SysOps Administrator

OFFICIAL STUDY GUIDE

ASSOCIATE EXAM



Covers exam objectives, including monitoring and metrics, high availability, analysis, deployment and provisioning, data management, security, networking, and much more...

Includes an interactive online learning environment and study tools with:

+ 2 custom practice exams

+ 100 electronic flashcards

+ Searchable key term glossary

SYBEX
A Wiley Brand

b. For Description, type a brief description of the option group. The description is used for display purposes.

c. For Engine, choose the DB engine that you want.

d. For Major Engine Version, choose the major version of the DB engine that you want.

5. To continue, choose Yes, Create. To cancel the operation instead, choose Cancel.

EXERCISE 7.2

Create an Amazon DynamoDB Table from the AWS CLI.

This exercise assumes that the AWS CLI has been installed. For readability, long commands in this section are broken into separate lines. The backslash character lets you copy and paste (or type) multiple lines into a Linux terminal. For Windows PowerShell, the backtick (`) does the same thing.

This exercise will create an Amazon DynamoDB table called `MusicCollection` that has the attributes Artist and Title.

Type the following command into the terminal window with the AWS CLI installed:

For Linux, use the following:

```
aws dynamodb create-table \
  --table-name MusicCollection \
  --attribute-definitions \
    AttributeName=Artist,AttributeType=S AttributeName=Title,AttributeType=S \
  --key-schema AttributeName=Artist,KeyType=HASH \
    AttributeName=Title,KeyType=RANGE \
  --provisioned-throughput ReadCapacityUnits=1,WriteCapacityUnits=1
```

For Microsoft Windows, use the following:

```
aws dynamodb create-table `
  --table-name MusicCollection `
  --attribute-definitions `
    AttributeName=Artist,AttributeType=S AttributeName=Title,AttributeType=S `
  --key-schema AttributeName=Artist,KeyType=HASH `
    AttributeName=Title,KeyType=RANGE `
  --provisioned-throughput ReadCapacityUnits=1,WriteCapacityUnits=1
```

EXERCISE 7.3

Add Items to the Amazon DynamoDB Table MusicCollection Using the AWS CLI.

Now that the table has been created, add data to it using the AWS CLI.

For Linux, use the following:

```
aws dynamodb put-item \
  --table-name MusicCollection \
  --item '{
    "Artist": {"S": "Mystical Father"},
    "Title": {"S": "Song for Elijah"} ,
    "AlbumTitle": {"S": "Lovely Laura Likes Llamas"} }' \
  --return-consumed-capacity TOTAL

aws dynamodb put-item \
  --table-name MusicCollection \
  --item '{
    "Artist": {"S": "Steps in Springtime"},
    "Title": {"S": "Allemande Left but will Return"} ,
    "AlbumTitle": {"S": "Square Dances for Round Rooms"} }' \
  --return-consumed-capacity TOTAL
```

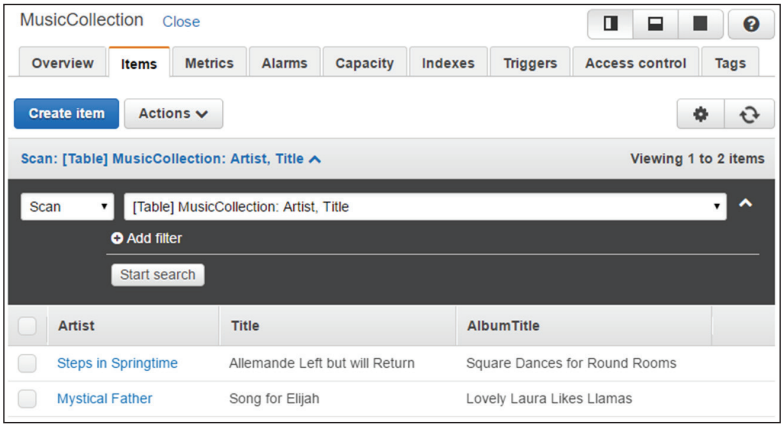
For Microsoft Windows, use the following:

```
aws dynamodb put-item `
  --table-name MusicCollection `
  --item '{
    \"Artist\": {\"S\": \"Mystical Father\"},
    \"Title\": {\"S\": \"Song for Elijah\"} ,
    \"AlbumTitle\": {\"S\": \"Lovely Laura Likes Llamas\"} }' `
  --return-consumed-capacity TOTAL

aws dynamodb put-item `
  --table-name MusicCollection `
  --item '{
    \"Artist\": {\"S\": \"Steps in Springtime\"},
    \"Title\": {\"S\": \"Allemande Left but will Return\"} ,
    \"AlbumTitle\": {\"S\": \"Square Dances for Round Rooms\"} }' `
  --return-consumed-capacity TOTAL
```

Microsoft Windows eats double quotes (") for breakfast. To hide them so that Amazon DynamoDB (or other services) can use them, use the backslash (\) as an escape character.

In the Amazon DynamoDB console, the table with items will look like the following graphic.



EXERCISE 7.4

Create a MySQL Amazon RDS DB Instance.

The basic building block of Amazon RDS is the DB instance. This is the environment in which you will run your MySQL databases.

In this exercise, create a DB instance running the MySQL database engine called **`west2-mysql-instance1`**, with a `db.m1.small` DB instance class, 5 GB of storage, and automated backups enabled with a retention period of one day.

Create a MySQL DB Instance

1. Sign in to the AWS Management Console, and open the Amazon RDS console.
2. In the top-right corner of the Amazon RDS console, choose the region in which to create the DB instance.
3. In the navigation pane, choose Instances.
4. Choose Launch DB Instance. The Launch DB Instance wizard opens on the Select Engine page.
5. On the Select Engine page, choose the MySQL icon, and then choose Select for the MySQL DB engine.

EXERCISE 7.4 *(continued)*

6. On the Specify DB Details page, specify the DB instance information as shown here:

DB Details

For this Parameter	Do This
License Model	Choose the default, general-public license to use the general license agreement for MySQL. MySQL has only one license model.
DB Engine Version	Choose the default version of MySQL. Note that Amazon RDS supports multiple versions of MySQL in some regions.
DB Instance Class	Choose `db.m1.small` for a configuration that equates to 1.7 GB memory, 1 ECU (1 virtual core with 1 ECU), 64-bit platform, and moderate I/O capacity.
Multi-AZ Deployment	Choose Yes to have a standby replica of the DB instance created in another Availability Zone for failover support. AWS recommends Multi-AZ for production workloads to maintain high availability.
Allocated Storage	Type **5** to allocate 5 GB of storage for the database. In some cases, allocating a higher amount of storage for your DB instance than the size of your database can improve I/O performance.
Storage Type	Choose the Magnetic storage type.
DB Instance Identifier	Type a name for the DB instance that is unique to the account in the region chosen. Feel free to add some intelligence to the name, such as including the region and DB engine chosen; for example, **`west2-mysql-instance1`**.
Master Username	Type a name using alphanumeric characters to use as the master user name to log on to the DB instance. This will be the user name used to log on to the database on the DB instance for the first time.
Master Password and Confirm Password	Type a password that contains from 8 to 41 printable ASCII characters (excluding /,", and @) for the master user password. This will be the password used for the user name to log on to the database. Type the password again in the Confirm Password box.

7. When the settings have been configured, choose Next.
8. On the Configure Advanced Settings page, provide additional information that Amazon RDS needs to launch the MySQL DB instance. The following table shows settings for the Amazon DB instance in this exercise:

Configure Advanced Settings

For This Parameter	Do This
VPC	Choose the name of the Amazon VPC that will host your MySQL DB instance.
Availability Zone	On the previous page, choosing Yes for the Multi-AZ Deployment parameter removes this option. Otherwise, select an AZ in which to deploy the database.
DB Security Groups	Choose the security group to use with this DB instance.
Database Name	Type a name for the default database that is between 1 to 64 alphanumeric characters long. Without a name, Amazon RDS will not automatically create a database on the newly provisioned DB instance.
Database Port	Leave the default value of 3306.
DB Parameter Group	Leave the default value.
Option Group	Choose the default value. This option group is used with the MySQL version chosen on the previous page.
Copy Tags To Snapshots	Choose this option to have any DB instance tags copied to a DB snapshot when creating a snapshot.
Enable Encryption	Choose Yes to enable encryption at rest for this DB instance.
Backup Retention Period	Set this value to 1.
Backup Window	Use the default of No Preference.
Enable Enhanced Monitoring	Use the default of No.
Auto Minor Version Upgrade	Choose Yes to enable the DB instance to receive minor DB engine version upgrades automatically when they become available.
Maintenance Window	Choose No Preference.

EXERCISE 7.4 *(continued)*

9. Specify your DB instance information and then choose Launch DB Instance.

On the Amazon RDS console, the new DB instance appears in the list of DB instances. The DB instance will have a status of `creating` until the DB instance is created and ready for use. When the state changes to `available`, connect to the database created on the DB instance.

Depending on the DB instance class and store allocated, it could take several minutes for the new DB instance to become available.

Review Questions

1. How are charges calculated for data transferred between Amazon Relational Database Service (Amazon RDS) and Amazon Elastic Compute Cloud (Amazon EC2) instances in the same Availability Zone?
   - **A.** GB out from Amazon RDS
   - **B.** GB in from Amazon RDS
   - **C.** IOPS from Amazon EC2
   - **D.** There is no charge.
2. On an Amazon Elastic Compute Cloud (Amazon EC2) instance using an Amazon Elastic Block Store (Amazon EBS) volume for persistence, there is a large inconsistency in the response times of the database queries. While investigating the issue, you find that there is a large amount of wait time on the database's disk volume. How can you improve the performance of the database's storage while maintaining the current persistence of the data? (Choose two.)
   - **A.** Move to a Solid-State Drive (SSD)-backed instance.
   - **B.** Use an Amazon EC2 instance with an Instance Store volume.
   - **C.** Use an Amazon EC2 instance type that is Amazon EBS-Optimized.
   - **D.** Use a Provisioned IOPs Amazon EBS volume.
3. Amazon Relational Database Service (Amazon RDS) automated backups and DB snapshots are currently supported for which storage engine?
   - **A.** JDBC
   - **B.** ODBC
   - **C.** InnoDB
   - **D.** MyISAM
4. In Amazon CloudWatch, which metric monitors the amount of free space on a DB instance?
   - **A.** `FreeStorageTotal`
   - **B.** `FreeStorageVolume`
   - **C.** `FreeStorageSpace`
   - **D.** `FreeRDSStorage`
5. In an Amazon DynamoDB table, what happens if the application performs more reads or writes than the provisioned capacity?
   - **A.** Requests above the provisioned capacity will be performed, but it will return HTTP 400 error codes.
   - **B.** Requests above the provisioned capacity will be throttled, and it will return HTTP 400 error codes.
   - **C.** Requests above the provisioned capacity will be performed, but it will return HTTP 500 error codes.
   - **D.** Requests above the provisioned capacity will be throttled, and it will return HTTP 500 error codes.

6. Which metric should be monitored carefully in order to know when a Read Replica should be recreated if it becomes out of sync due to replication errors?
   - **A.** `ReadLag`
   - **B.** `ReadReplica`
   - **C.** `ReplicaLag`
   - **D.** `ReplicaThreshold`

7. Can connections between my application and a DB instance be encrypted using Secure Sockets Layer (SSL)?
   - **A.** Yes
   - **B.** No
   - **C.** Yes, but only inside an Amazon Virtual Private Cloud (Amazon VPC)
   - **D.** Yes, but only inside certain regions

8. After inadvertently deleting a database table, which Amazon Relational Database Service (Amazon RDS) feature will allow you to restore a database reliably to within five minutes of the deletion?
   - **A.** Amazon RDS Automated Backup
   - **B.** Multi-AZ Amazon RDS
   - **C.** Amazon RDS Read Replicas
   - **D.** Amazon RDS Snapshots

9. What is the range, in days, to which the backup retention period can be set?
   - **A.** From 1 day to 7 days
   - **B.** From 1 day to 28 days
   - **C.** From 1 day to 35 days
   - **D.** From 1 day to 90 days

10. Regarding Amazon DynamoDB, which of the following statements is correct?
    - **A.** Each item should have at least two value sets: a primary key and an attribute.
    - **B.** An item can have multiple attributes.
    - **C.** Primary keys must store a single value.
    - **D.** Individual attributes can have one or more sub-attributes.

11. The maximum Amazon DynamoDB item size limit is which of the following?
    - **A.** 256 KB
    - **B.** 400 KB
    - **C.** 512 KB
    - **D.** 1,024 KB

12. Will a standby Amazon Relational Database Service (Amazon RDS) instance be in the same Availability Zone as the primary?
    - **A.** Yes
    - **B.** No
    - **C.** Only for Oracle and Microsoft SQL Server-based Amazon RDS instances
    - **D.** Only if configured at launch

13. For both Single and Multi-AZ deployments, defining a subnet for all Availability Zones in a region allows Amazon RDS to create a new standby in another Availability Zone should the need arise. Is creating a Read Replica of another Read Replica supported inside Amazon Relational Database Service (Amazon RDS)?
    - **A.** No
    - **B.** Only in certain regions
    - **C.** Only with MySQL-based Amazon RDS instances
    - **D.** Only for Oracle Amazon RDS DB instances

14. When running a DB instance as a Multi-AZ deployment, can the standby DB instance be used for read or write operations?
    - **A.** Yes
    - **B.** No
    - **C.** Only with Microsoft SQL Server-based Amazon Relational Database Service (Amazon RDS) DB instances
    - **D.** Only for Oracle-based Amazon RDS DB instances

15. If there are multiple Read Replicas for a primary DB instance and one of them is promoted, what happens to the rest of the Read Replicas?
    - **A.** The remaining Read Replicas will still replicate from the original primary DB instance.
    - **B.** The other Read Replicas will be deleted.
    - **C.** The remaining Read Replicas suspend operations.
    - **D.** Read Replicas cannot be promoted.

16. When using a Multi-AZ deployment, in the event of a planned or unplanned outage of the primary DB instance, Amazon RDS automatically switches to the standby replica. Which DNS record is updated by the automatic failover mechanism and points to the standby DB instance?
    - **A.** The A Record
    - **B.** CNAME
    - **C.** MX
    - **D.** SOA

17. When automatic failover occurs, Amazon Relational Database Service (Amazon RDS) will create a DB instance event. Using the AWS CLI, which of the following will return information about the events related to the DB instance?
    - **A.** `ReturnEventFailure`
    - **B.** `DescribeFailureEvent`
    - **C.** `DescribeEvents`
    - **D.** `ReturnEvents`

18. Is it possible to force a failover for a MySQL Multi-AZ DB instance deployment?
    - **A.** Yes
    - **B.** No
    - **C.** Only in certain regions
    - **D.** Only in Amazon Virtual Private Cloud (Amazon VPC)

19. How does the Amazon Relational Database Service (Amazon RDS) Multi-AZ model work?
    - **A.** A second, standby database is deployed and maintained in a different Availability Zone from the primary database, using synchronous replication.
    - **B.** A second, standby database is deployed and maintained in a different region from the primary database, using synchronous replication.
    - **C.** A second, standby database is deployed and maintained in a different Availability Zone from the primary database, using asynchronous replication.
    - **D.** A second, standby database is deployed and maintained in a different region from the primary database, using asynchronous replication.

20. What does Amazon ElastiCache provide?
    - **A.** An Amazon Elastic Compute Cloud (Amazon EC2) instance with a large amount of memory and CPU
    - **B.** A managed in-memory cache service
    - **C.** An Amazon EC2 instance with Redis and Memcached preinstalled
    - **D.** A highly available and fast indexing service for searching

21. When developing a highly available web application using stateless web servers, which services are suitable for storing session-state data? (Choose three.)
    - **A.** Amazon Simple Queue Service (Amazon SQS)
    - **B.** Amazon Relational Database Service (Amazon RDS)
    - **C.** Amazon CloudWatch
    - **D.** Amazon ElastiCache
    - **E.** Amazon DynamoDB
    - **F.** Amazon CloudFront

22. Which statement best describes Amazon ElastiCache?
    - **A.** It reduces the latency of a DB instance by splitting the workload across multiple Availability Zones.
    - **B.** It provides a simple web interface to create and store multiple datasets, query your data easily, and return the results.
    - **C.** It is a managed service from AWS that makes it easy to set up, operate, and scale a relational database in the cloud.
    - **D.** It offloads the read traffic from a database in order to reduce latency caused by read-heavy workload.

23. Which two AWS Cloud services provide out-of-the-box, user-configurable, automatic backup-as-a-service and backup rotation options? (Choose two.)
    - **A.** AWS CloudFormation
    - **B.** Amazon Simple Storage Service (Amazon S3)
    - **C.** Amazon Relational Database Service (Amazon RDS)
    - **D.** Amazon Elastic Block Store (Amazon EBS)
    - **E.** Amazon DynamoDB
    - **F.** Amazon Redshift

Chapter 8

Application Deployment and Management

THE AWS CERTIFIED SYSOPS ADMINISTRATOR - ASSOCIATE EXAM TOPICS COVERED IN THIS CHAPTER MAY INCLUDE, BUT ARE NOT LIMITED TO, THE FOLLOWING:

Domain 2.0 High Availability

✓ **2.1 Implement scalability and elasticity based on scenarios**

Content may include the following:

- Which AWS compute service to use for deploying scalable and elastic environments
- Including scalability of specific AWS compute service in the deployment and management of applications
- Methods for implementing upgrades while maintaining high availability

✓ **2.2 Ensure level of fault tolerance based on business needs**

Content may include the following:

- What AWS Cloud deployment services can be used for deploying fault-tolerant applications

Domain 4.0 Deployment and Provisioning

✓ **4.1 Demonstrate the ability to build the environment to conform to architectural design**

Content may include the following:

- Choosing the appropriate AWS Cloud service to meet requirements for deploying applications

✓ **4.2 Demonstrate the ability to provision cloud resources and manage implementation automation**

Content may include the following:

- Automating the deployment and provisioning of AWS Cloud services

Introduction to Application Deployment and Management

As a candidate for the AWS Certified SysOps Administrator – Associate certification, you will need to be familiar with application deployment strategies and services used for the deployment and management of applications. AWS offers many capabilities for provisioning your infrastructure and deploying your applications. The deployment model varies from customer to customer depending on the capabilities required to support operations. Understanding these capabilities and techniques will help you pick the best strategy and toolset for deploying the infrastructure that can handle your workload.

An experienced systems operator is aware of the "one size doesn't fit all" philosophy. For enterprise computing or to create the next big social media or gaming company, AWS provides multiple customization options to serve a broad range of use cases. The AWS platform is designed to address scalability, performance, security, and ease of deployment, as well as to provide tools to help migrate applications and an ecosystem of developers and architects that are deeply involved in the growth of its products and services.

This chapter details different deployment strategies and services. It reviews common features available on these deployment services, articulates strategies for updating application stacks, and presents examples of common usage patterns for various workloads.

Deployment Strategies

AWS offers several key features that are unique to each deployment service that will be discussed later in this chapter. There are some characteristics that are common to these services, however. This section discusses various common features and capabilities that a systems operator will need to understand to choose deployment strategies. Each feature can influence service adoption in its own way.

Provisioning Infrastructure

You can work with building-block services individually, such as provisioning an Amazon Elastic Compute Cloud (Amazon EC2) instance, Amazon Elastic Block Store (Amazon EBS)

volume, Amazon Simple Storage Service (Amazon S3) bucket, Amazon Virtual Private Cloud (Amazon VPC) environment, and so on. Alternately, you can use automation provided by deployment services to provision infrastructure components. The main advantage of using automated capabilities is the rich feature set that they offer for deploying and configuring your application and all the resources it requires. For example, you can use an AWS CloudFormation template to treat your infrastructure as code. The template describes all of the resources that will be provisioned and how they should be configured.

Deploying Applications

The AWS deployment services can also make it easier to deploy your application on the underlying infrastructure. You can create an application, specify the source repository to your desired deployment service, and let the deployment service handle the complexity of provisioning the AWS resources needed to run your application. Despite providing similar functionality in terms of deployment, each service has its own unique method for deploying and managing your application.

Configuration Management

Why does a systems operator need to consider configuration management? You may need to deploy resources quickly for a variety of reasons—upgrading your deployments, replacing failed resources, automatically scaling your infrastructure, etc. It is important for a systems operator to consider how the application deployment should be configured to respond to scaling events automatically.

In addition to deploying your application, the deployment services can customize and manage the application configuration. The underlying task could be replacing custom configuration files in your custom web application or updating packages that are required by your application. You can customize the software on your Amazon EC2 instance as well as the infrastructure resources in your stack configuration.

Systems operators need to track configurations and any changes made to the environments. When you need to implement configuration changes, the strategy you use will allow you to target the appropriate resources. Configuration management also enables you to have an automated and repeatable process for deployments.

Tagging

Another advantage of using deployment services is the automation of tag usage. A *tag* consists of a user-defined key and value. For example, you can define tags such as application, project, cost centers, department, purpose, and stack so that you can easily identify a resource. When you use tags during your deployment steps, the tools automatically propagate the tags to underlying resources such as Amazon EC2 instances, Auto Scaling groups, or Amazon Relational Database Service (Amazon RDS) instances.

Appropriate use of tagging can provide a better way to manage your budgets with cost allocation reports. Cost allocation reports aggregate costs based on tags. This way, you can determine how much you are spending for each application or a particular project.

Custom Variables

When you develop an application, you want to customize configuration values, such as database connection strings, security credentials, and other information, which you don't want to hardcode into your application. Defining variables can help loosely couple your application configuration and give you the flexibility to scale different tiers of your application independently. Embedding variables outside of your application code also helps improve portability of your application. Additionally, you can differentiate environments into development, test, and production based on customized variables. The deployment services facilitate customizing variables so that once they are set, the variables become available to your application environments. For example, an AWS CloudFormation template could contain a parameter that's used for your web-tier Amazon EC2 instance to connect to an Amazon RDS instance. This parameter is inserted into the user data script so that the application installed on the Amazon EC2 instance can connect to the database.

Baking Amazon Machine Images (AMIs)

An *Amazon Machine Image (AMI)* provides the information required to launch an instance. It contains the configuration information for instances, including the block device mapping for volumes and what snapshot will be used to create the volume. A *snapshot* is an image of the volume. The root volume would consist of the base operating system and anything else within the volume (such as additional applications) that you've installed.

In order to launch an Amazon EC2 instance, you need to choose which AMI you will use for your application. A common practice is to install an application on an instance at the first boot. This process is called *bootstrapping an instance*.

The bootstrapping process can be slower if you have a complex application or multiple applications to install. Managing a fleet of applications with several build tools and dependencies can be a challenging task during rollouts. Furthermore, your deployment service should be designed to perform faster rollouts to take advantage of Auto Scaling.

Baking an image is the process of creating your own AMI. Instead of using a bootstrap script to deploy your application, which could take an extended amount of time, this custom AMI could contain a portion of your application artifacts within it. However, during deployment of your instance, you can also use user data to customize application installations further. AMIs are regionally scoped—to use an image in another region, you will need to copy the image to all regions where it will be used.

The key factor to keep in mind is how long it takes for the instance to launch. If scripted installation of each instance takes an extended amount of time, this could impact your ability to scale quickly. Alternatively, copying the block of a volume where your application is already installed could be faster. The disadvantage is that if your application is changed, you'll need either to bake a new image, which can also be automated, or use a configuration management tool to apply the changes.

For example, let's say that you are managing an environment consisting of web, application, and database tiers. You can have logical grouping of your base AMIs that can take 80 percent of application binaries loaded on these AMI sets. You can choose to install the remaining applications during the bootstrapping process and alter the installation based on configuration sets grouped by instance tags, Auto Scaling groups, or other instance artifacts. You can set a tag on your resources to track for which tier of your environment they are used. When deploying an update, the process can query for the instance tag, validate whether it's the most current version of the application, and then proceed with the installation. When it's time to update the AMI, you can simply swap your existing AMI with the most recent version in the underlying deployment service and update the tag.

You can script the process of baking an AMI. In addition, there are multiple third-party tools for baking AMIs. Some well-known ones are Packer by HashiCorp and Aminator by Netflix. You can also choose third-party tools for your configuration management, such as Chef, Puppet, Salt, and Ansible.

Logging

Logging is an important element of your application deployment cycle. Logging can provide important debugging information or provide key characteristics of your application behavior. The deployment services make it simpler to access these logs through a combination of the AWS Management Console, AWS Command Line Interface (AWS CLI), and Application Programming Interface (API) methods so that you don't have to log in to Amazon EC2 instances to view them.

In addition to built-in features, the deployment services provide seamless integration with Amazon CloudWatch Logs to expand your ability to monitor the system, application, and custom log files. You can use Amazon CloudWatch Logs to monitor logs from Amazon EC2 instances in real time, monitor AWS CloudTrail events, or archive log data in Amazon S3 for future analysis.

Instance Profiles

Applications that run on an Amazon EC2 instance must include AWS credentials in their API requests. You could have your developers store AWS credentials directly within the Amazon EC2 instance and allow applications in that instance to use those credentials. But developers would then have to manage the credentials and ensure that they securely pass the credentials to each instance and update each Amazon EC2 instance when it's time to rotate the credentials. That's a lot of additional work. There is also the potential that the credentials could be compromised, copied from the Amazon EC2 instance, and used elsewhere.

Instead, you can and should use an AWS Identity and Access Management (IAM) role to manage temporary credentials for applications that run on an Amazon EC2 instance. When you use a role, you don't have to distribute long-term credentials to an Amazon EC2 instance. Instead, the role supplies temporary permissions that applications can use when they make calls to other AWS resources. When you launch an Amazon EC2 instance, you specify an IAM role to associate with the instance. Applications that run on the instance can then use the role-supplied temporary credentials to sign API requests.

Instance profiles are a great way of embedding necessary IAM roles that are required to carry out an operation to access an AWS resource. An instance profile is a container for an IAM role that you can use to pass role information to an Amazon EC2 instance when the instance starts. An instance profile can contain only one IAM role, although a role can be included in multiple instance profiles. These IAM roles can be used to make API requests securely from your instances to AWS Cloud services without requiring you to manage security credentials. The deployment services integrate seamlessly with instance profiles to simplify credentials management and relieve you from hardcoding API keys in your application configuration.

For example, if your application needs to access an Amazon S3 bucket with read-only permission, you can create an instance profile and assign read-only Amazon S3 access in the associated IAM role. The deployment service will take the complexity of passing these roles to Amazon EC2 instances so that your application can securely access AWS resources with the privileges that you define.

Scalability Capabilities

Scaling your application fleet automatically to handle periods of increased demand not only provides a better experience for your end users, but it also keeps the cost low. As demand decreases, resources can automatically be scaled in. Therefore, you're only paying for the resources needed based on the load.

For example, you can configure Auto Scaling to add or remove Amazon EC2 instances dynamically based on metrics triggers that you set within Amazon CloudWatch (such as CPU, memory, disk I/O, and network I/O). This type of Auto Scaling configuration is integrated seamlessly into AWS Elastic Beanstalk and AWS CloudFormation. Similarly, AWS OpsWorks and Amazon EC2 Container Services (Amazon ECS) have capabilities to manage scaling automatically based on time or load. Amazon ECS has Service Auto Scaling that uses a combination of the Amazon ECS, Amazon CloudWatch, and Application Auto Scaling APIs to scale application containers automatically.

Monitoring Resources

Monitoring gives you visibility into the resources you launch in the cloud. Whether you want to monitor the resource utilization of your overall stack or get an overview of your application health, the deployment services are integrated with monitoring capabilities to provide this info within your dashboards. You can navigate to the Amazon CloudWatch console to get a system-wide view into all of your resources and operational health. Alarms can be created for metrics that you want to monitor. When the threshold is surpassed, the alarm is triggered and can send an alert message or take an action to mitigate an issue. For example, you can set an alarm that sends an email alert when an Amazon EC2 instance fails on status checks or trigger a scaling event when the CPU utilization meets a certain threshold.

Each deployment service provides the progress of your deployment. You can track the resources that are being created or removed via the AWS Management Console, AWS CLI, or APIs.

Continuous Deployment

This section introduces various deployment methods, operations principles, and strategies a systems operator can use to automate integration, testing, and deployment.

Depending on your choice of deployment service, the strategy for updating your application code could vary a fair amount. AWS deployment services bring agility and improve the speed of your application deployment cycle, but using a proper tool and the right strategy is key for building a robust environment.

The following section looks at how the deployment service can help while performing application updates. Like any deployment lifecycle, the methods you use have trade-offs and considerations, so the method you implement will need to meet the specific requirements of a given deployment.

Deployment Methods

There are two primary methods that you can use with deployment services to update your application stack: in-place upgrade and replacement upgrade. An *in-place upgrade* involves performing application updates on existing Amazon EC2 instances. A *replacement upgrade*, however, involves provisioning new Amazon EC2 instances, redirecting traffic to the new resources, and terminating older instances.

An in-place upgrade is typically useful in a rapid deployment with a consistent rollout schedule. It is designed for stateless applications. You can still use the in-place upgrade method for stateful applications by implementing a rolling deployment schedule and by following the guidelines mentioned in the section below on blue/green deployments.

In contrast, replacement upgrades offer a simpler way to deploy by provisioning new resources. By deploying a new stack and redirecting traffic from the old to the new one, you don't have the complexity of upgrading existing resource and potential failures. This is also useful if your application has unknown dependencies. The underlying Amazon EC2 instance usage is considered temporary or ephemeral in nature for the period of deployment until the current release is active. During the new release, a new set of Amazon EC2 instances is rolled out by terminating older instances. This type of upgrade technique is more common in an immutable infrastructure.

There are several deployment services that are especially useful for an in-place upgrade: AWS CodeDeploy, AWS OpsWorks, and AWS Elastic Beanstalk. *AWS CodeDeploy* is a deployment service that automates application deployments to Amazon EC2 instances or on-premises instances in your own facility. AWS CodeDeploy makes it easier for you to release new features rapidly, helps you avoid downtime during application deployment, and handles the complexity of updating your applications without many of the risks associated with error-prone manual deployments. You can also use *AWS OpsWorks* to manage your application deployment and updates. When you deploy an application, AWS OpsWorks Stacks triggers a Deploy event, which runs each layer's Deploy recipes. AWS OpsWorks Stacks also installs stack configuration and deployment attributes that contain all of the information needed to deploy the application, such as the application's repository and database connection data. *AWS Elastic Beanstalk* provides several options for how deployments are processed, including deployment policies (All at Once, Rolling, Rolling with Additional

Batch, and Immutable) and options that let you configure batch size and health check behavior during deployments.

For replacement upgrades, you provision a new environment with the deployment services, such as AWS Elastic Beanstalk, AWS CloudFormation, and AWS OpsWorks. A full set of new instances running the new version of the application in a separate Auto Scaling group will be created alongside the instances running the old version. Immutable deployments can prevent issues caused by partially completed rolling deployments. Typically, you will use a different Elastic Load Balancing load balancer for both the new stack and the old stack. By using Amazon Route 53 with weighted routing, you can roll traffic to the load balancer of the new stack.

In-Place Upgrade

AWS CodeDeploy is a deployment service that automates application deployments to Amazon EC2 instances or on-premises instances in your own facility. You can deploy a nearly unlimited variety of application content, such as code, web and configuration files, executables, packages, scripts, multimedia files, and so on. AWS CodeDeploy can deploy application content stored in Amazon S3 buckets, GitHub repositories, or Bitbucket repositories. Once you prepare deployment content and the underlying Amazon EC2 instances, you can deploy an application and its revisions on a consistent basis. You can push the updates to a set of instances called a *deployment group* that is made up of tagged Amazon EC2 instances and/or Auto Scaling groups. In addition, AWS CodeDeploy works with various configuration management tools, continuous integration and deployment systems, and source control systems. You can find a complete list of product integration options in the AWS CodeDeploy documentation.

AWS CodeDeploy is used for deployments by AWS CodeStar. *AWS CodeStar* enables you to develop, build, and deploy applications quickly on AWS. AWS CodeStar provides a unified user interface, enabling you to manage your software development activities easily in one place. With AWS CodeStar, you can set up your entire continuous delivery toolchain in minutes, allowing you to start releasing code faster. AWS CodeStar stores your application code securely on *AWS CodeCommit*, a fully managed source control service that eliminates the need to manage your own infrastructure to host Git repositories. AWS CodeStar compiles and packages your source code with *AWS CodeBuild*, a fully managed build service that makes it possible for you to build, test, and integrate code more frequently. AWS CodeStar accelerates software release with the help of *AWS CodePipeline*, a Continuous Integration and Continuous Delivery (CI/CD) service. AWS CodeStar integrates with AWS CodeDeploy and AWS CloudFormation so that you can easily update your application code and deploy to Amazon EC2 and AWS Lambda.

Another service to use for managing the entire lifecycle of an application is AWS OpsWorks. You can use built-in layers or deploy custom layers and recipes to launch your application stack. In addition, numerous customization options are available for configuration and pushing application updates. When you deploy an application, AWS OpsWorks Stacks triggers a Deploy event, which runs each layer's Deploy recipes. AWS OpsWorks Stacks also installs stack configuration and deployment attributes that contain all of the information needed to deploy the application, such as the application's repository and database connection data.

Replacement Upgrade

The replacement upgrade method replaces in-place resources with newly provisioned resources. There are advantages and disadvantages between the in-place upgrade method and replacement upgrade method. You can perform a replacement upgrade in a number of ways. You can use an Auto Scaling policy to define how you want to add (scale out) or remove (scale in) instances. By coupling this with your update strategy, you can control the rollout of an application update as part of the scaling event.

For example, you can create a new Auto Scaling Launch Configuration that specifies a new AMI containing the new version of your application. Then you can configure the Auto Scaling group to use the new launch configuration. The Auto Scaling termination policy by default will first terminate the instance with the oldest launch configuration and that is closest to the next billing hour. This in effect provides the most cost-effective method to phase out all instances that use the previous configuration. If you are using Elastic Load Balancing, you can attach an additional Auto Scaling configuration behind the load balancer and use a similar approach to phase in newer instances while removing older instances.

Similarly, you can configure rolling deployments in conjunction with deployment services such as AWS Elastic Beanstalk and AWS CloudFormation. You can use update policies to describe how instances in an Auto Scaling group are replaced or modified as part of your update strategy. With these deployment services, you can configure the number of instances to get updated concurrently or in batches, apply the updates to certain instances while isolating in-service instances, and specify the time to wait between batched updates. In addition, you can cancel or roll back an update if you discover a bug in your application code. These features can help increase the availability of your application during updates.

Blue/Green Deployments

Blue/green is a method where you have two identical stacks of your application running in their own environments. You use various strategies to migrate the traffic from your current application stack (blue) to a new version of the application (green). This method is used for a replacement upgrade. During a blue/green deployment, the latest application revision is installed on replacement instances and traffic is rerouted to these instances either immediately or as soon as you are done testing the new environment.

This is a popular technique for deploying applications with zero downtime. Deployment services like AWS Elastic Beanstalk, AWS CloudFormation, or AWS OpsWorks are particularly useful for blue/green deployments because they provide a simple way to duplicate your existing application stack.

Blue/green deployments offer a number of advantages over in-place deployments. An application can be installed and tested on the new instances ahead of time and deployed to production simply by switching traffic to the new servers. Switching back to the most recent version of an application is faster and more reliable because traffic can be routed back to the original instances as long as they have not been terminated. With an in-place deployment, versions must be rolled back by redeploying the previous version of the application. Because the instances provisioned for a blue/green deployment are new, they reflect

the most up-to-date server configurations, which helps you avoid the types of problems that sometimes occur on long-running instances.

For a stateless web application, the update process is pretty straightforward. Simply upload the new version of your application and let your deployment service deploy a new version of your stack (green). To cut over to the new version, you simply replace the Elastic Load Balancing URLs in your Domain Name Server (DNS) records. AWS Elastic Beanstalk has a Swap Environment URLs feature to facilitate a simpler cutover process. If you use Amazon Route 53 to manage your DNS records, you need to swap Elastic Load Balancing endpoints for AWS CloudFormation or AWS OpsWorks deployment services.

For applications with session states, the cutover process can be complex. When you perform an update, you don't want your end users to experience downtime or lose data. You should consider storing the sessions outside of your deployment service because creating a new stack will re-create the session database with a certain deployment service. In particular, consider storing the sessions separately from your deployment service if you are using an Amazon RDS database.

If you use Amazon Route 53 to host your DNS records, you can consider using the Weighted Round Robin (WRR) feature for migrating from blue to green deployments. The feature helps to drive the traffic gradually rather than instantly. If your application has a bug, this method helps ensure that the blast radius is minimal, as it only affects a small number of users. This method also simplifies rollbacks if they become necessary by redirecting traffic back to the blue stack. In addition, you only use the required number of instances while you scale up in the green deployment and scale down in the blue deployment. For example, you can set WRR to allow 10 percent of the traffic to go to green deployment while keeping 90 percent of traffic on blue. You gradually increase the percentage of green instances until you achieve a full cutover. Keeping the DNS cache to a shorter Time To Live (TTL) on the client side also ensures that the client will connect to the green deployment with a rapid release cycle, thus minimizing bad DNS caching behavior. For more information on Amazon Route 53, see Chapter 5, "Networking."

Hybrid Deployments

You can also use the deployment services in a hybrid fashion for managing your application fleet. For example, you can combine the simplicity of managing AWS infrastructure provided by AWS Elastic Beanstalk and the automation of custom network segmentation provided by AWS CloudFormation. Leveraging a hybrid deployment model also simplifies your architecture because it decouples your deployment method so that you can choose different strategies for updating your application stack.

Deployment Services

AWS deployment services provide easier integration with other AWS Cloud services. Whether you need to load-balance across multiple Availability Zones by using Elastic Load Balancing or by using Amazon RDS as a back end, the deployment services like AWS Elastic

Beanstalk, AWS CloudFormation, and AWS OpsWorks make it simpler to use these services as part of your deployment.

If you need to use other AWS Cloud services, you can leverage tool-specific integration methods to interact with the resource. For example, if you are using AWS Elastic Beanstalk for deployment and want to use Amazon DynamoDB for your back end, you can customize your environment resources by including a configuration file within your application source bundle. With AWS OpsWorks, you can create custom recipes to configure the application so that it can access other AWS Cloud services. Similarly, several template snippets with a number of example scenarios are available for you to use within your AWS CloudFormation templates.

AWS offers multiple strategies for provisioning infrastructure. You could use the building blocks (for example Amazon EC2, Amazon EBS, Amazon S3, and Amazon RDS) and leverage the integration provided by third-party tools to deploy your application. But for even greater flexibility, you can consider the automation provided by the AWS deployment services.

AWS Elastic Beanstalk

AWS Elastic Beanstalk is the fastest and simplest way to get an application up and running on AWS. It is ideal for developers who want to deploy code and not worry about managing the underlying infrastructure. AWS Elastic Beanstalk reduces management complexity without restricting choice or control. You simply upload your application, and AWS Elastic Beanstalk automatically handles the details of capacity provisioning, load balancing, scaling, and application health monitoring.

AWS Elastic Beanstalk provides platforms for programming languages (such as Java, PHP, Python, Ruby, or Go), web containers (for example Tomcat, Passenger, and Puma), and Docker containers, with multiple configurations of each. AWS Elastic Beanstalk provisions the resources needed to run your application, including one or more Amazon EC2 instances. The software stack running on the Amazon EC2 instances depends on the configuration. In a configuration name, the version number refers to the version of the platform configuration. You can also perform most deployment tasks, such as changing the size of your fleet of Amazon EC2 instances or monitoring your application, directly from the AWS Elastic Beanstalk Console.

Applications

The first step in using AWS Elastic Beanstalk is to create an application that represents your web application in AWS. In AWS Elastic Beanstalk, an application serves as a container for the environments that run your web application and versions of your web application's source code, saved configurations, logs, and other artifacts that you create while using AWS Elastic Beanstalk.

Environment Tiers

When you launch an AWS Elastic Beanstalk environment, you choose an environment tier, platform, and environment type. The environment tier that you choose determines whether

that provisions the Amazon VPC in a standard way for the organization. By parameterizing the template, they can reuse it for different deployments.

AWS Command Line Interface (AWS CLI)

The AWS CLI can be used to automate systems operations. You can use it to manage an operational environment that is currently in production, automating the provisioning and updating of application deployments.

Generating Skeletons

Most AWS CLI commands support `--generate-cli-skeleton` and `--cli-input-json` parameters that you can use to store parameters in JSON and read them from a file instead of typing them at the command line. You can generate AWS CLI skeleton outputs in JSON that outline all of the parameters that can be specified for the operation. This is particularly useful for generating templates that are used in scripting the deployment of applications. For example, you can use it to generate a template for Amazon ECS task definitions or an AWS CloudFormation resource's `Properties` section.

To generate an Amazon ECS task definition template, run the following AWS CLI command.

```
aws ecs register-task-definition --generate-cli-skeleton
```

You can use this template to create your task definition, which can then be pasted into the console JSON input area or saved to a file and used with the AWS CLI `--cli-input-json` option.

This process is also useful in generating the properties attributes for an AWS CloudFormation resource. You could use the output generated in the previous command.

```
{
  "Type" : "AWS::ECS::TaskDefinition",
  "Properties" : {
       <paste-generate-cli-skeleton>
  }
}
```

This can be done for other resources in an AWS CloudFormation template, such as an Amazon EC2 instance. Run the following command and paste the output into the `Properties` section of the resource.

```
aws ec2 run-instances --generate-cli-skeleton
```

```
{
  "Type" : "AWS::EC2::Instance",
  "Properties" : {
       <paste-generate-cli-skeleton>
  }
}
```

Summary

In this chapter, we discussed the following deployment strategies:

- Provisioning Infrastructure
  - Automating the provisioning of building-block services with AWS Cloud deployment services
- Deploying Applications
  - Methods of deploying applications onto your infrastructure
- Configuration Management
  - Using tagging to track resources and manage infrastructure
  - Using custom variables to make deployments flexible and reusable
  - Strategies for creating AMIs
  - Configuring infrastructure for security and troubleshooting
- Scalability Capabilities
  - Including scaling capabilities for the infrastructure of an application
  - Scalability considerations during the upgrade of an application
- Continuous Deployment
  - In-place vs. replacement upgrades
  - Methods for deploying application and infrastructure upgrades

In this chapter, we discussed the following deployment services:

- AWS Elastic Beanstalk
  - Creating and managing AWS Elastic Beanstalk environments
  - Deploying and managing an application in an AWS Elastic Beanstalk environment
- Amazon ECS
  - Building an Amazon ECS cluster
  - Deploying instances used for an Amazon ECS cluster
  - Managing containers with Amazon ECS tasks and services
  - Using a repository for container images
- AWS OpsWorks Stacks
  - Creating an AWS OpsWorks stack
  - Using layers for managing Amazon EC2 instances
  - Deploying applications to a layer of your stack
- AWS CloudFormation
  - Creating, updating, and deleting an AWS CloudFormation Stack
  - Methods and considerations for updating an AWS CloudFormation Stack

JON BONSO AND KENNETH SAMONTE

AWS CERTIFIED DEVOPS ENGINEER PROFESSIONAL

Tutorials Dojo
Study Guide and Cheat Sheets

Types of Blue Green Deployment via ELB, Route 53, Elastic Beanstalk

AWS Elastic Beanstalk Blue/Green

Elastic Beanstalk, by default, performs an in-place update when you deploy a newer version of your application. This can cause a short downtime since your application will be stopped while Elastic Beanstalk performs the application update.

Blue/Green deployments allow you to deploy without application downtime.

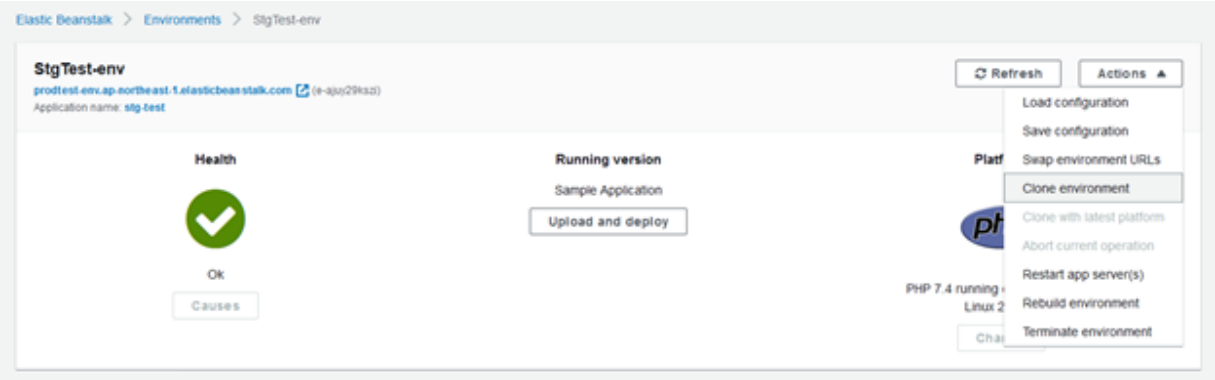
DevOps Exam Notes:

Remember these key points on when to use blue/green deployments:

- No downtime during deployment because you are deploying the newer version on a separate environment
- CNAMEs of the environment URLs are swapped to redirect traffic to the newer version.
- Route 53 will swap the CNAMEs of the application endpoints.
- Fast deployment time and quick rollback since both old and new versions are running at the same time, you just have to swap back the URLs if you need to rollback.
- Useful if your newer version is incompatible with the current platform version of your application. (ex. Jumping from major versions of NodeJS, Python, Ruby, PHP, etc.)
- Your RDS Database instance should be on a separate stack because the data will not transfer to your second environment. You should decouple your database from the web server stack.

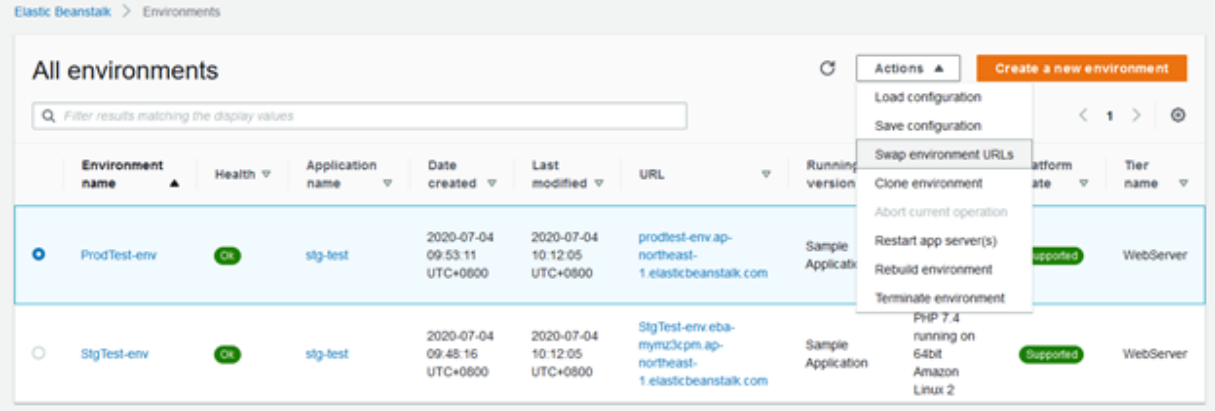
To implement a Blue/Green deployment for your Elastic Beanstalk application, you can perform the following steps:

1. Create another environment on which you will deploy the newer version of your application. You can clone your current environment for easier creation.

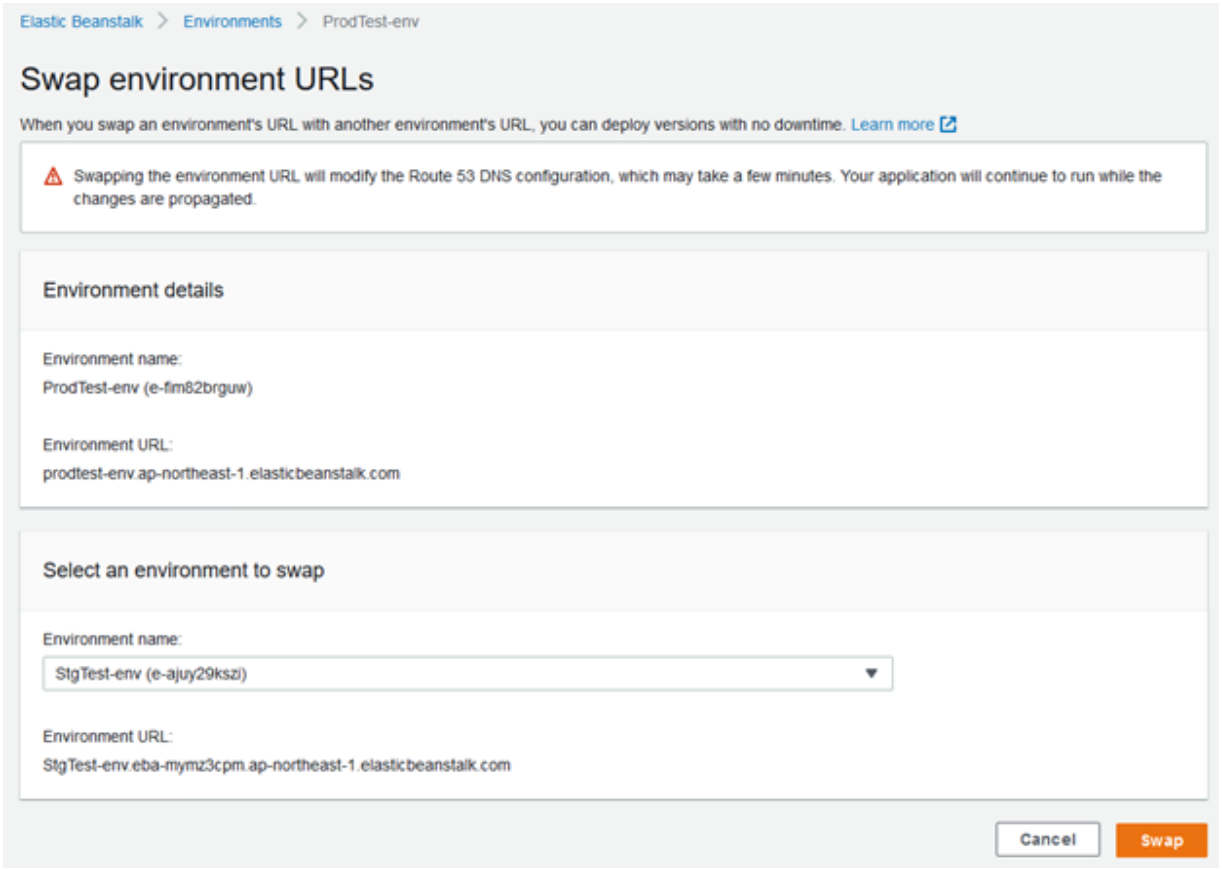


2. Once the new environment is ready, deploy a new version of your application. Perform your tests on the URL endpoint of your new environment.

3. After testing, select your Production environment, click Actions > Swap environment URLs.



4. On the Swap Environment URLs page, select the newer environment and click Swap to apply the changes.


Elastic Beanstalk > Environments > ProdTest-env
Swap environment URLs
When you swap an environment's URL with another environment's URL, you can deploy versions with no downtime. Learn more
Swapping the environment URL will modify the Route 53 DNS configuration, which may take a few minutes. Your application will continue to run while the changes are propagated.
Environment details
Environment name:
ProdTest-env (e-fim82brguw)
Environment URL:
prodtest-env.ap-northeast-1.elasticbeanstalk.com
Select an environment to swap
Environment name:
StgTest-env (e-ajuy29kszi)
Environment URL:
StgTest-env.eba-mymz3cpm.ap-northeast-1.elasticbeanstalk.com
Cancel
Swap

AWS Lambda Blue/Green

You can also implement Blue/Green deployments on your Lambda functions. The concept is the same as in Elastic beanstalk blue/green deployment i.e. you will need to create two versions of your Lambda function and use function Aliases to swap the traffic flow.

Lambda versions – lets you publish a new version of a function that you can test without affecting the current application accessed by users. You can create multiple versions as needed for your testing environments. The ARN of Lambda version is the same as the ARN of the Lambda function with added version suffix.

arn:aws:lambda:aws-region:acct-id:function:helloworld:$LATEST

Lambda aliases – Aliases are merely pointers to specific Lambda versions. You can't select a Lambda alias and edit the function. You need to select the LATEST version if you want to edit the function. Aliases are helpful for blue/green deployments because it allows you to use a fixed ARN and point it to a particular Lambda version that you want to deploy.

DevOps Exam Notes:

Remember the difference between Lambda $LATEST, Lambda Versions and Lambda Aliases:

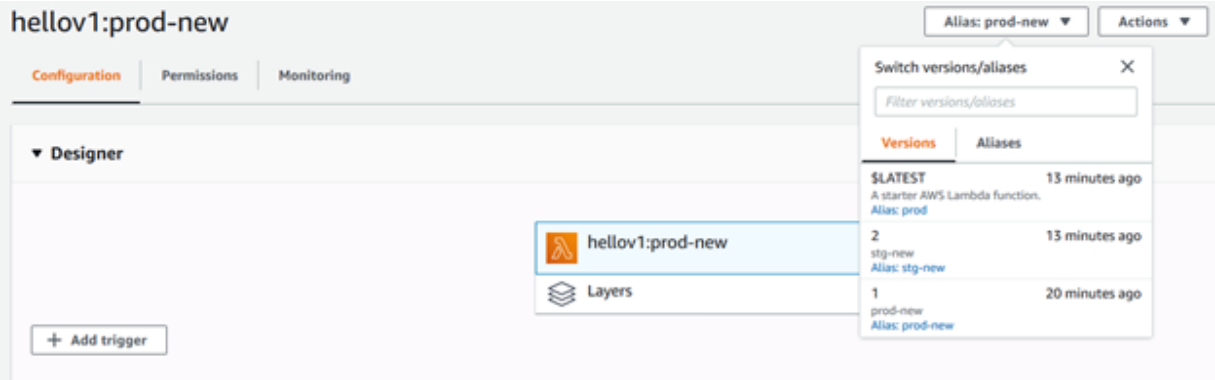
$LATEST - this is the latest version of your Lambda function. You can freely edit this version.

Lambda Version - fixed version of your function. You can't edit this directly.

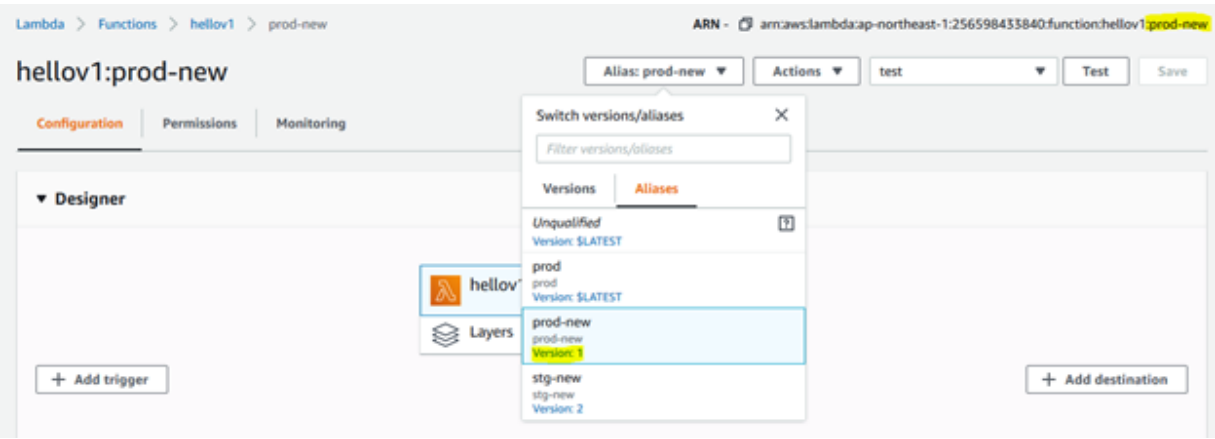
Lambda Alias - a pointer to a specific Lambda version. You can perform blue/green deployment with Aliases by pointing to a newer version.

The following steps will show how blue/green deployment can be done on Lambda functions.

1. The current version of your Lambda function is deployed on Version 1. Create another version and make your changes, this will be Version 2.



2. Create an Alias that will point to the current production version. Use this alias as your fixed production ARN.



3. Create another Alias that you will use for your newer version. Perform your testing and validation on this newer version. Once testing is complete, edit the production alias to point to the newer version. Traffic will now instantly be shifted from the previous version to the newer version.

Sources:
https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.CNAMESwap.html
https://docs.aws.amazon.com/lambda/latest/dg/configuration-aliases.html
https://docs.aws.amazon.com/lambda/latest/dg/configuration-versions.html

Elastic Beanstalk - Deployment Policies and Settings

AWS Elastic Beanstalk provides several options for how deployments are processed, including deployment policies (All at once, Rolling, Rolling with additional batch, Immutable, and Traffic splitting) and options that let you configure batch size and health check behavior during deployments. By default, your environment uses all-at-once deployments.

All at once – The quickest deployment method. Suitable if you can accept a short loss of service, and if quick deployments are important to you. With this method, Elastic Beanstalk deploys the new application version to each instance.

Rolling deployments - Elastic Beanstalk splits the environment's Amazon EC2 instances into batches and deploys the new version of the application to one batch at a time. During a rolling deployment, some instances serve requests with the old version of the application, while instances in completed batches serve other requests with the new version.

Rolling deployment with additional batch - launches new batches during the deployment. To maintain full capacity during deployments, you can configure your environment to launch a new batch of instances before taking any instances out of service. When the deployment completes, Elastic Beanstalk terminates the additional batch of instances.

Immutable deployments - perform an immutable update to launch a full set of new instances running the new version of the application in a separate Auto Scaling group, alongside the instances running the old version. Immutable deployments can prevent issues caused by partially completed rolling deployments. If the new instances don't pass health checks, Elastic Beanstalk terminates them, leaving the original instances untouched.

Traffic-splitting deployments - let you perform canary testing as part of your application deployment. In a traffic-splitting deployment, Elastic Beanstalk launches a full set of new instances just like during an immutable deployment. It then forwards a specified percentage of incoming client traffic to the new application version for a specified evaluation period. If the new instances stay healthy, Elastic Beanstalk forwards all traffic to them and terminates the old ones. If the new instances don't pass health checks, or if you choose to abort the deployment, Elastic Beanstalk moves traffic back to the old instances and terminates the new ones.

Here's a summary of the deployment methods, how long each deployment takes, and how rollback is handled.

Method	Impact of failed deployment	Deploy time	Zero downtime	No DNS change	Rollback process	Code deployed to
All at once	Downtime	🕒	No	Yes	Manual redeploy	Existing instances
Rolling	Single batch out of service; any successful batches before failure running new application version	🕒 🕒 †	Yes	Yes	Manual redeploy	Existing instances
Rolling with an additional batch	Minimal if first batch fails; otherwise, similar to **Rolling**	🕒 🕒 🕒 †	Yes	Yes	Manual redeploy	New and existing instances
Immutable	Minimal	🕒 🕒 🕒 🕒	Yes	Yes	Terminate new instances	New instances
Traffic splitting	Percentage of client traffic routed to new version temporarily impacted	🕒 🕒 🕒 🕒 ††	Yes	Yes	Reroute traffic and terminate new instances	New instances
Blue/green	Minimal	🕒 🕒 🕒 🕒	Yes	No	Swap URL	New instances

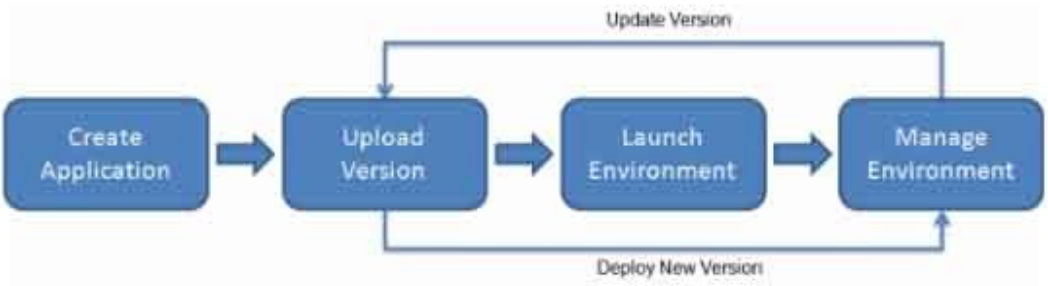
† *Varies depending on batch size.*

†† *Varies depending on* ***evaluation time*** *option setting.*

Sources:
https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.rolling-version-deploy.html
https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.deploy-existing-version.html

AWS Elastic Beanstalk

- Allows you to quickly deploy and manage applications in the AWS Cloud without worrying about the infrastructure that runs those applications.
- Elastic Beanstalk automatically handles the details of capacity provisioning, load balancing, scaling, and application health monitoring for your applications.
- Elastic Beanstalk supports Docker containers.
- Elastic Beanstalk Workflow



- Your application's domain name is in the format: *subdomain.region*.elasticbeanstalk.com

Elastic Beanstalk Concepts

- **Application** - a logical collection of Elastic Beanstalk components, including environments, versions, and environment configurations. It is conceptually similar to a folder.
- **Application Version** - refers to a specific, labeled iteration of deployable code for a web application. An application version points to an Amazon S3 object that contains the deployable code. Applications can have many versions and each application version is unique.
- **Environment** - a version that is deployed on to AWS resources. Each environment runs only a single application version at a time, however you can run the same version or different versions in many environments at the same time.
- **Environment Tier** - determines whether Elastic Beanstalk provisions resources to support an application that handles HTTP requests or an application that pulls tasks from a queue. An application that serves HTTP requests runs in a **web server environment**. An environment that pulls tasks from an Amazon SQS queue runs in a **worker environment**.
- **Environment Configuration** - identifies a collection of parameters and settings that define how an environment and its associated resources behave.
- **Configuration Template** - a starting point for creating unique environment configurations.
- There is a limit to the number of application versions you can have. You can avoid hitting the limit by applying an *application version lifecycle policy* to your applications to tell Elastic Beanstalk to delete

application versions that are old, or to delete application versions when the total number of versions for an application exceeds a specified number.

AWS CodePipeline

- A fully managed **continuous delivery service** that helps you automate your release pipelines for application and infrastructure updates.
- You can easily integrate AWS CodePipeline with third-party services such as GitHub or with your own custom plugin.
- **Concepts**
    - A **pipeline** defines your release process workflow, and describes how a new code change progresses through your release process.
    - A pipeline comprises a series of **stages** (e.g., build, test, and deploy), which act as logical divisions in your workflow. Each stage is made up of a sequence of actions, which are tasks such as building code or deploying to test environments.
        - Pipelines must have **at least two stages**. The first stage of a pipeline is required to be a source stage, and the pipeline is required to additionally have at least one other stage that is a build or deployment stage.
    - Define your pipeline structure through a **declarative JSON** document that specifies your release workflow and its stages and actions. These documents enable you to update existing pipelines as well as provide starting templates for creating new pipelines.
    - A **revision** is a change made to the source location defined for your pipeline. It can include source code, build output, configuration, or data. A pipeline can have multiple revisions flowing through it at the same time.
    - A **stage** is a group of one or more actions. A pipeline can have two or more stages.
    - An **action** is a task performed on a revision. Pipeline actions occur in a specified order, in serial or in parallel, as determined in the configuration of the stage.
        - You can add actions to your pipeline that are in an AWS Region different from your pipeline.
        - There are six types of actions
            - Source
            - Build
            - Test
            - Deploy
            - Approval
            - Invoke
    - When an action runs, it acts upon a file or set of files called **artifacts**. These artifacts can be worked upon by later actions in the pipeline. You have an artifact store which is an S3 bucket in the same AWS Region as the pipeline to store items for all pipelines in that Region associated with your account.
    - The stages in a pipeline are connected by **transitions.** Transitions can be disabled or enabled between stages. If all transitions are enabled, the pipeline runs continuously.

    - An **approval action** prevents a pipeline from transitioning to the next action until permission is granted. This is useful when you are performing code reviews before code is deployed to the next stage.
- **Features**
    - AWS CodePipeline provides you with a graphical user interface to create, configure, and manage your pipeline and its various stages and actions.
    - A pipeline starts automatically (default) when a change is made in the source location, or when you manually start the pipeline. You can also set up a rule in CloudWatch to automatically start a pipeline when events you specify occur.
    - You can model your build, test, and deployment actions to run **in parallel** in order to increase your workflow speeds.
    - AWS CodePipeline can pull source code for your pipeline directly from AWS CodeCommit, GitHub, Amazon ECR, or Amazon S3.
    - It can run builds and unit tests in AWS CodeBuild.
    - It can deploy your changes using AWS CodeDeploy, AWS Elastic Beanstalk, Amazon ECS, AWS Fargate, Amazon S3, AWS Service Catalog, AWS CloudFormation, and/or AWS OpsWorks Stacks.
    - You can use the CodePipeline Jenkins plugin to easily register your existing build servers as a custom action.
    - When you use the console to create or edit a pipeline that has a GitHub source, CodePipeline creates a **webhook**. A webhook is an HTTP notification that detects events in another tool, such as a GitHub repository, and connects those external events to a pipeline. CodePipeline deletes your webhook when you delete your pipeline.
- As a best practice, when you use a Jenkins build provider for your pipeline's build or test action, install Jenkins on an Amazon EC2 instance and configure a separate EC2 instance profile. Make sure the instance profile grants Jenkins only the AWS permissions required to perform tasks for your project, such as retrieving files from Amazon S3.

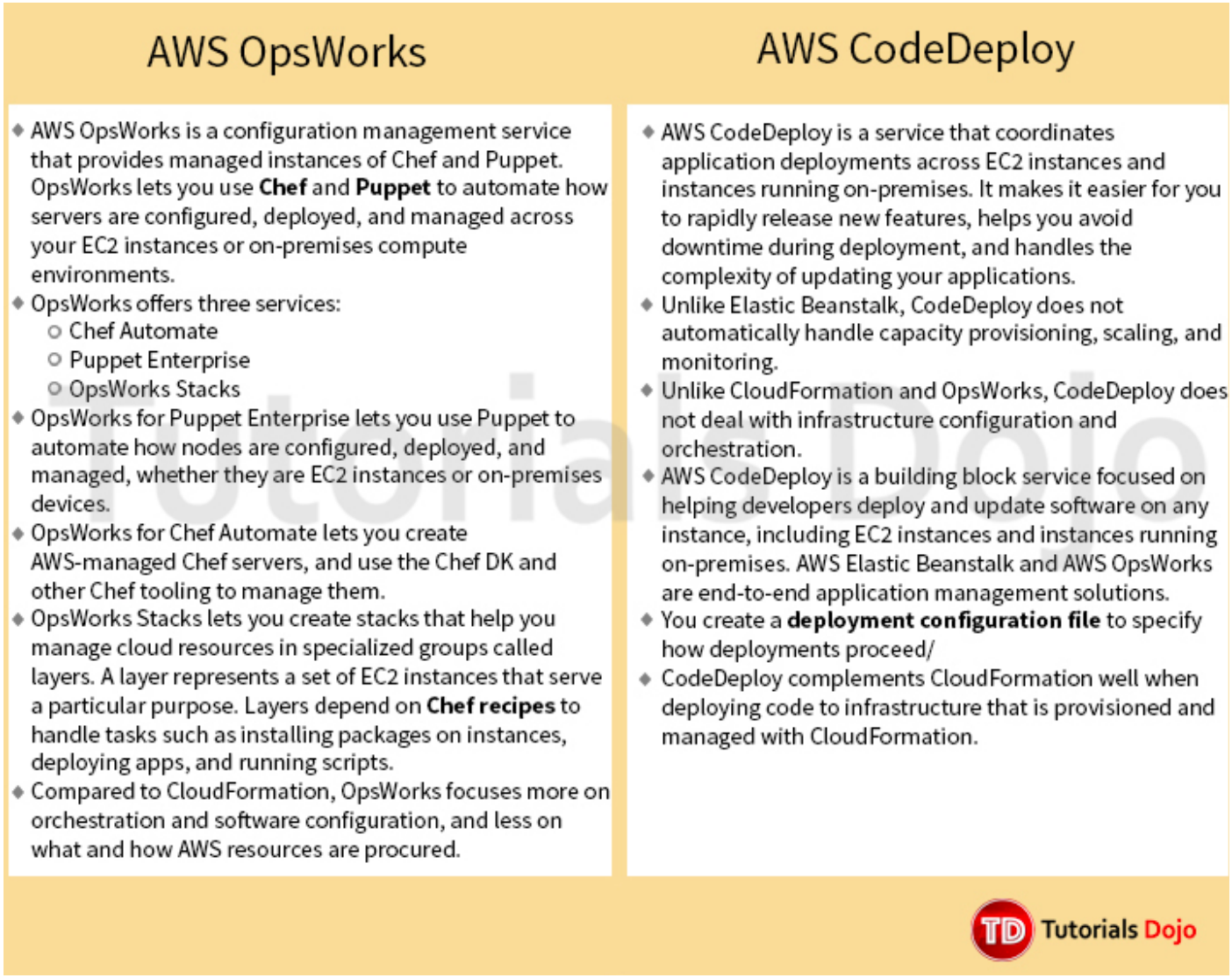
Elastic Beanstalk vs CloudFormation vs OpsWorks vs CodeDeploy

AWS Elastic Beanstalk

- AWS Elastic Beanstalk makes it even easier for developers to **quickly deploy and manage applications** in the AWS Cloud. Developers simply upload their application, and Elastic Beanstalk **automatically handles the deployment details** of capacity provisioning, load balancing, auto-scaling, and application health monitoring.
- This **platform-as-a-service solution** is typically for those who want to deploy and manage their applications within minutes in the AWS Cloud without worrying about the underlying infrastructure.
- AWS Elastic Beanstalk supports the following languages and development stacks:
  - Apache Tomcat for Java applications
  - Apache HTTP Server for PHP applications
  - Apache HTTP Server for Python applications
  - Nginx or Apache HTTP Server for Node.js applications
  - Passenger or Puma for Ruby applications
  - Microsoft IIS for .NET applications
  - Java SE
  - Docker
  - Go
- Elastic Beanstalk also supports deployment versioning. It maintains a copy of older deployments so that it is easy for the developer to rollback any changes made on the application.

AWS CloudFormation

- AWS CloudFormation is a service that gives developers and businesses an easy way to create a **collection of related AWS resources** and provision them in an orderly and predictable fashion. This is typically known as **"infrastructure as code".**
- The main difference between CloudFormation and Elastic Beanstalk is that CloudFormation deals more with the AWS infrastructure rather than applications. AWS CloudFormation introduces two concepts:
  - The **template**, a JSON or YAML-format, text-based file that describes all the AWS resources and configurations you need to deploy to run your application.
  - The **stack**, which is the set of AWS resources that are created and managed as a single unit when AWS CloudFormation instantiates a template.
- CloudFormation also supports a rollback feature through template version controls. When you try to update your stack but the deployment failed midway,
- CloudFormation will automatically revert the changes back to their previous working states.
- CloudFormation supports Elastic Beanstalk application environments. This allows you, for example, to create and manage an AWS Elastic Beanstalk–hosted application along with an RDS database to store the application data.
- AWS CloudFormation can be used to bootstrap both Chef (Server and Client) and Puppet (Master and Client) softwares on your EC2 instances.
- CloudFormation also supports OpsWorks. You can now model OpsWorks components (stacks, layers, instances, and applications) inside CloudFormation templates, and provision them as CloudFormation stacks. This enables you to document, version control, and share your OpsWorks configuration.
- AWS CodeDeploy is a recommended adjunct to CloudFormation for managing the application deployments and updates.



Additional Notes:

- Elastic Beanstalk, CloudFormation, or OpsWorks are particularly useful for **blue-green deployment method** as they provide a simple way to clone your running application stack.
- CloudFormation and OpsWorks are best suited for the **prebaking AMIs**.
- CodeDeploy and OpsWorks are best suited for performing **in-place application upgrades**. For **disposable upgrades**, you can set up a cloned environment with Elastic Beanstalk, CloudFormation, and OpsWorks.