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

in the additional labor and time involved in performing database backups, upgrades, and monitoring, you may find that using RDS to handle these tasks for you would be more cost effective.

Operational Excellence

Operational excellence is fundamentally about automating the processes required to achieve and maintain the other four goals of reliability, performance efficiency, cost optimization, and security. In reality, few organizations automate everything, so operational excellence is more of a journey than a goal. But the idea behind operational excellence is to incrementally improve and automate more activities for the purpose of strengthening the other pillars. Here are some simple examples of how automation can help achieve operational excellence:

Reliability Use elastic load balancing health checks to monitor the health of an application running on several EC2 instances. When an instance fails, route users away from the failed instance and create a new one.

Performance efficiency Use EC2 Auto Scaling dynamic scaling policies to scale in and out automatically, ensuring you always have as many instances as you need and no more.

Security Use CodeBuild to automatically test new application code for security vulnerabilities. When deploying an application, use CloudFormation to automatically deploy fresh, secure infrastructure, rather than following a manual checklist.

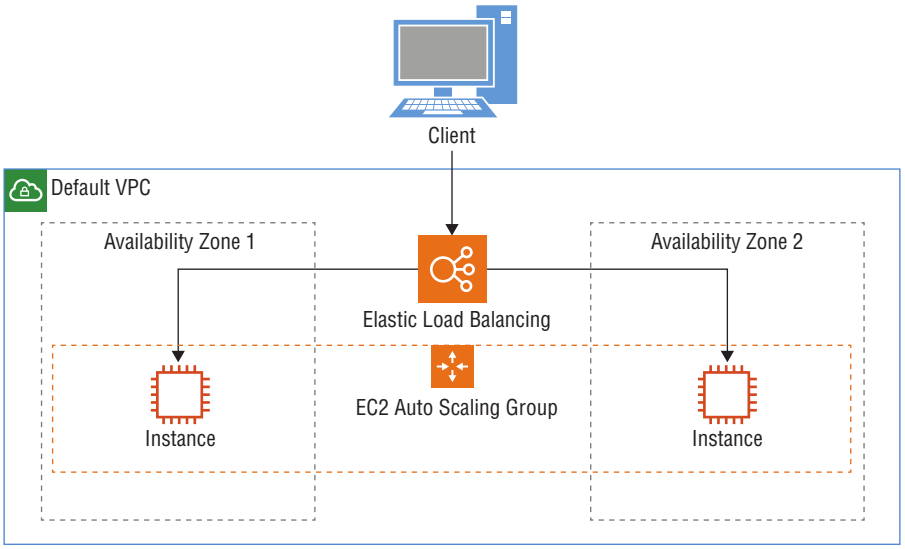
Cost optimization Automatically shut down or decommission unused resources. For example, implement S3 object life cycle configurations to delete unneeded objects. Or if you have EC2 instances that are used for testing only during business hours, you can automatically shut them down at the end of the workday and restart them the following morning.

Keep in mind that these examples are by no means exhaustive. Many opportunities for automation aren't obvious until there's a failure. For instance, if someone deletes an application load balancer that's operating as the front end of a critical web application, recovering from such an event would entail performing some quick manual work-arounds and re-creating the load balancer. Understanding how failures affect your application can help you avoid such failures in the future and automate recovery when they occur.

A Highly Available Web Application Using Auto Scaling and Elastic Load Balancing

The first scenario we'll consider is a highly available web application that you'll implement using the topology shown in Figure 12.1.

FIGURE 12.1 A highly available web application using Auto Scaling and elastic load balancing



Don't worry if this looks intimidating. We'll start by taking a high-level look at how these components fit together, and then we'll dig into the configuration specifics. EC2 Auto Scaling will initially provision two EC2 instances running in different Availability Zones inside the default VPC. Every default VPC has a default subnet in each Availability Zone, so there's no need to create subnets explicitly. An application load balancer will proxy the connections from clients on the internet and distribute those connections to the instances that are running the popular open-source Apache web server.

We'll implement this scenario in four basic steps:

1. Create an inbound security group rule in the VPC's default security group.
2. Create an application load balancer.
3. Create a launch template.
4. Create an Auto Scaling group.

Creating an Inbound Security Group Rule

Every VPC comes with a default security group. Recall that security groups block traffic that's not explicitly allowed by a rule, so we'll start by adding an inbound rule to the VPC's default security group to allow only inbound HTTP access to the application load balancer and the instances. Complete Exercise 12.1 to create the inbound rule.

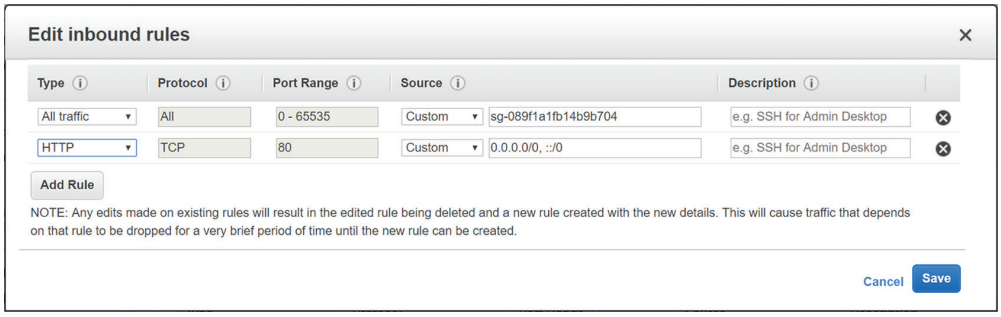
EXERCISE 12.1

Create an Inbound Security Group Rule

You'll start by modifying the default security group to allow inbound HTTP access. The application load balancer that you'll create later, as well as the instances that Auto Scaling will create, will use this security group to permit users to connect to the web application.

1. Browse to the EC2 service console. In the navigation pane, choose the Security Groups link.
2. Select the Create Security Group button.
3. Select the default security group for the default VPC.
4. Select the Inbound tab.
5. Select the Edit button.
6. Select the Add Rule button.
7. Under the Type drop-down list, select HTTP. The Management Console automatically populates the Protocol field with TCP and the Port Range field with 80, which together correspond to the HTTP protocol used for web traffic. It also populates the Source field with the IP subnet 0.0.0.0/0 and IPv6 subnet ::0/0, allowing traffic from all sources. You can optionally change the Source field to reflect the subnet of your choice or a specific IP address. For example, to allow only the IP address 198.51.100.100, you'd enter **198.51.100.100/32**. Refer to Figure 12.2 to see what your new inbound rule should look like.
8. Select the Save button.

FIGURE 12.2 Modifying the default security group



Creating an Application Load Balancer

Next, you need to create an application load balancer that will distribute connections to the instances in the Auto Scaling group that you'll create in a later exercise. The load balancer will route traffic to healthy instances using a round-robin algorithm that distributes traffic evenly to the instances without regard for how busy those instances are.

You'll configure the load balancer to perform a health check to monitor the health of the application on each instance. If the application fails on an instance, the load balancer will route connections away from the failed instance. The Auto Scaling group will also use this health check to determine whether an instance is healthy or needs to be replaced. Follow the steps in Exercise 12.2 to create the application load balancer.

EXERCISE 12.2

Create an Application Load Balancer

Now you'll create an application load balancer to receive incoming connections from users. The application load balancer will distribute those connections to the instances in the Auto Scaling group that you'll create later.

1. While in the EC2 service console, in the navigation pane on the left, choose the Load Balancers link.
2. Select the Create Load Balancer button.
3. Under Application Load Balancer, select the Create button.
4. In the Name field, type **sample-web-app-load-balancer**.
5. For the Scheme, select the radio button next to Internet-facing. This will assign the load balancer a publicly resolvable domain name and allow the load balancer to receive connections from the internet.
6. In the IP Address Type drop-down list, select IPv4.
7. Under Listeners, make sure the Load Balancer Protocol field is HTTP and the Load Balancer Port field is 80. Refer to Figure 12.3 for an example of what your basic load balancer configuration should look like.
8. On the Configure Load Balancer page, under the Availability Zones section, in the VPC drop-down, make sure your default VPC is selected.
9. Select at least two Availability Zones. Refer to Figure 12.4 for an example.
10. Select the button titled Next: Configure Security Settings.
11. You may see a message warning you that your load balancer isn't using a secure listener. Select the button titled Next: Configure Security Groups.
12. Select the default security group for the VPC.

FIGURE 12.3 Application load balancer basic configuration

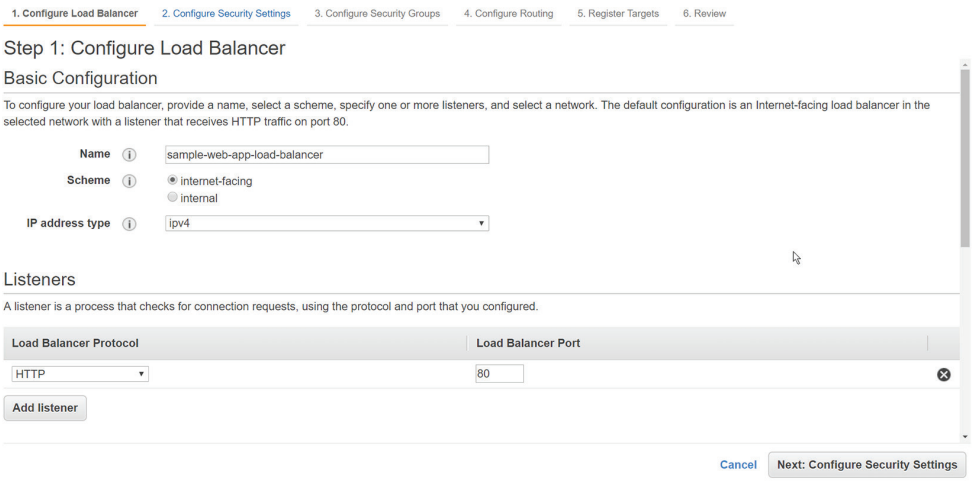
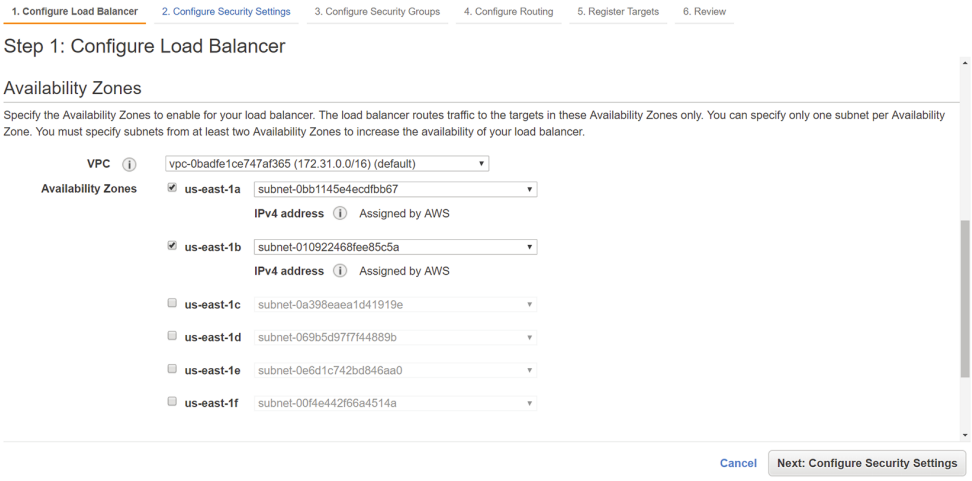


FIGURE 12.4 Application load balancer Availability Zones configuration



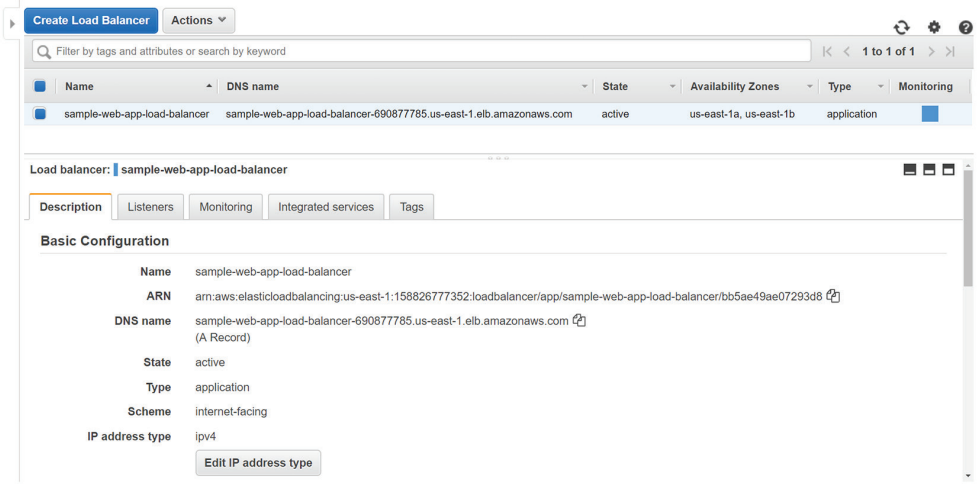
13. Select the button titled Next: Configure Routing.
14. Under Target Group, in the Target Group drop-down list, select New Target Group.
15. In the Name field, type **sample-web-app-target-group**.
16. Next to Target Type, select the Instance radio button.

EXERCISE 12.2 *(continued)*

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

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

FIGURE 12.5 Application load balancer details



Creating a Launch Template

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

EXERCISE 12.3

Create a Launch Template

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

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

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

Creating an Auto Scaling Group

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

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

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

This may sound like a lot to do, but the wizard makes it easy. Complete Exercise 12.4 to create and configure the Auto Scaling group.

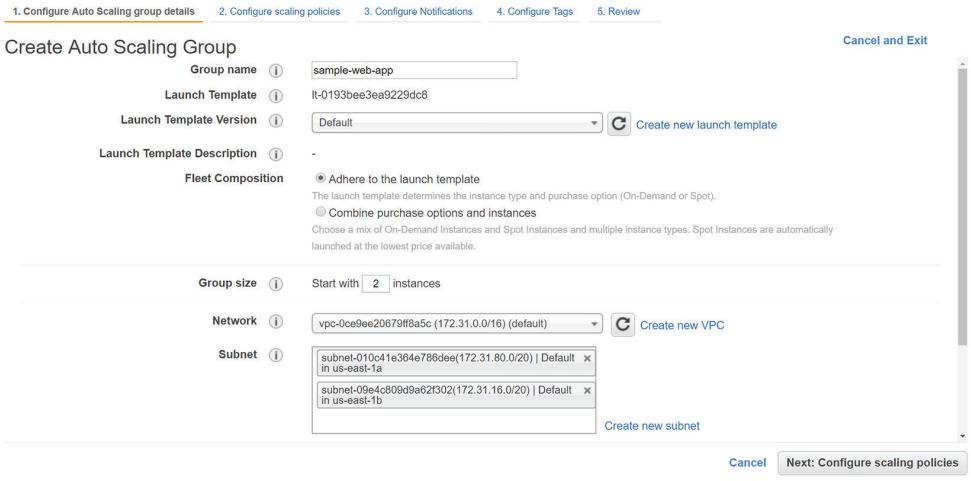
EXERCISE 12.4

Create an Auto Scaling Group

In this exercise, you'll create an Auto Scaling group that will provision your web server instances using the launch template.

1. Browse to the EC2 service console.
2. In the navigation pane, choose the Launch Templates link.
3. Select the launch template.
4. Select the Action button, and choose Create Auto Scaling Group.
5. In the Group Name field, enter **sample-web-app**.
6. In the Group Size field, enter **2**.
7. In the Network drop-down list, select the default VPC.
8. In the Subnet drop-down list, select at least two subnets. Refer to Figure 12.6 to get an idea of what the basic configuration should look like.

FIGURE 12.6 Auto Scaling group basic configuration



9. Expand the Advanced Details section.
10. Select the check box next to Receive Traffic From One Or More Load Balancers.

11. In the Target Groups field, select sample-web-app-target-group.
12. Next to Health Check Type, select the ELB radio button.
13. Select the button titled Next: Configure Scaling Policies.
14. Under Create Auto Scaling Group, select the Use Scaling Policies To Adjust The Capacity Of This Group radio button. This will create a dynamic scaling policy that will automatically scale in or out based on the average aggregate CPU utilization of the instances in the Auto Scaling group.
15. Adjust the minimum and maximum group size to scale between one and three instances. This will ensure that the group always has at least one instance but never more than three instances.
16. Under Scale Group Size, select the Metric Type Average CPU Utilization.
17. For the Target Value, enter **50**. This will cause Auto Scaling to attempt to keep the average CPU utilization of each instance at 50 percent. If the average CPU utilization falls below 50 percent, Auto Scaling will scale in, leaving only one instance in the group. If the average CPU utilization rises above 50 percent, Auto Scaling will add more instances to the group for a total of up to three.
18. Select the button titled Next: Configure Notifications.
19. Select the button titled Next: Configure Tags.
20. Select the Review button.
21. Review the settings, and select the Create Auto Scaling Group button.
22. Select the View Your Auto Scaling Groups link, and wait a few minutes for Auto Scaling to provision the instances.
23. Open a web browser, and browse to the DNS name of the application load balancer that you noted in Exercise 12.2. You should see the Apache Linux AMI test page, as shown in Figure 12.7.

FIGURE 12.7 The Apache Linux AMI test page

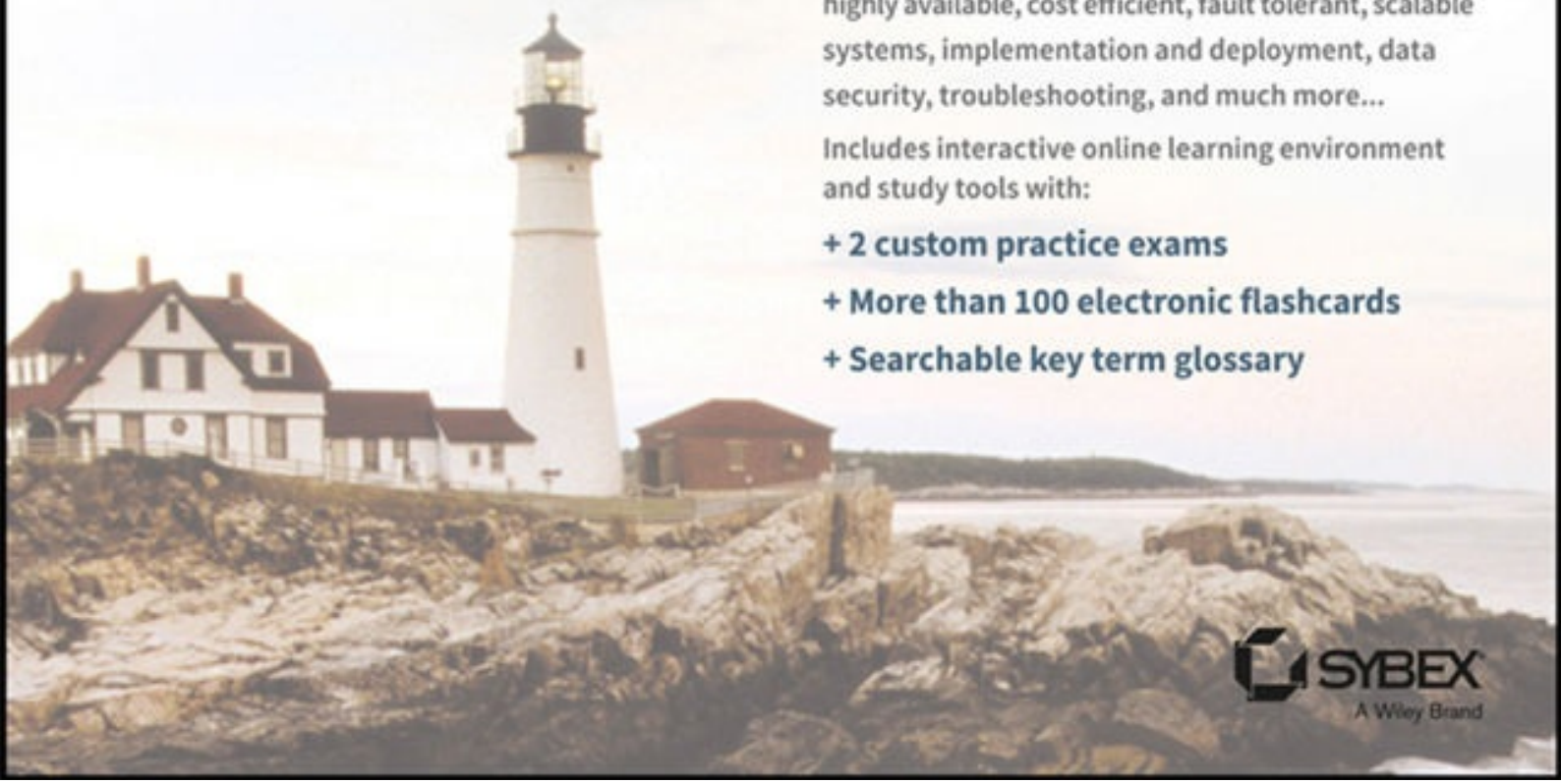
amazon web services

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

Chapter 5
Elastic Load Balancing, Amazon CloudWatch, and Auto Scaling

THE AWS CERTIFIED SOLUTIONS ARCHITECT EXAM TOPICS COVERED IN THIS CHAPTER MAY INCLUDE, BUT ARE NOT LIMITED TO, THE FOLLOWING:

Domain 1.0: Designing highly available, cost-effective, fault-tolerant, scalable systems

✓ **1.1 Identify and recognize cloud architecture considerations, such as fundamental components and effective designs.**

- Elasticity and scalability

Domain 2.0: Implementation/Deployment

✓ **2.1 Identify the appropriate techniques and methods using Amazon Elastic Compute Cloud (Amazon EC2), Amazon Simple Storage Service (Amazon S3), AWS Elastic Beanstalk, AWS CloudFormation, AWS OpsWorks, Amazon Virtual Private Cloud (Amazon VPC), and AWS Identity and Access Management (IAM) to code and implement a cloud solution.**

Content may include the following:

- Launch instances across the AWS global infrastructure

Domain 3.0: Data Security

✓ **3.1 Recognize and implement secure practices for optimum cloud deployment and maintenance.**

- CloudWatch Logs

Domain 4.0: Troubleshooting

Content may include the following:

- General troubleshooting information and questions

Introduction

In this chapter, you will learn how Elastic Load Balancing, Amazon CloudWatch, and Auto Scaling work both independently and together to help you efficiently and cost-effectively deploy highly available and optimized workloads on AWS.

Elastic Load Balancing is a highly available service that distributes traffic across Amazon Elastic Compute Cloud (Amazon EC2) instances and includes options that provide flexibility and control of incoming requests to Amazon EC2 instances.

Amazon CloudWatch is a service that monitors AWS Cloud resources and applications running on AWS. It collects and tracks metrics, collects and monitors log files, and sets alarms. Amazon CloudWatch has a basic level of monitoring for no cost and a more detailed level of monitoring for an additional cost.

Auto Scaling is a service that allows you to maintain the availability of your applications by scaling Amazon EC2 capacity up or down in accordance with conditions you set.

This chapter covers all three services separately, but it also highlights how they can work together to build more robust and highly available architectures on AWS.

Elastic Load Balancing

An advantage of having access to a large number of servers in the cloud, such as Amazon EC2 instances on AWS, is the ability to provide a more consistent experience for the end user. One way to ensure consistency is to balance the request load across more than one server. A load balancer is a mechanism that automatically distributes traffic across multiple Amazon EC2 instances. You can either manage your own virtual load balancers on Amazon EC2 instances or leverage an AWS Cloud service called Elastic Load Balancing, which provides a managed load balancer for you.

The Elastic Load Balancing service allows you to distribute traffic across a group of Amazon EC2 instances in one or more *Availability Zones*, enabling you to achieve high availability in your applications. Elastic Load Balancing supports routing and load balancing of Hypertext Transfer Protocol (HTTP), Hypertext Transfer Protocol Secure (HTTPS), *Transmission Control Protocol (TCP)*, and *Secure Sockets Layer (SSL)* traffic to Amazon EC2 instances. Elastic Load Balancing provides a stable, single *Canonical Name record (CNAME)* entry point for *Domain Name System (DNS)* configuration and supports both Internet-facing and internal application-facing load balancers. Elastic Load Balancing supports health checks for Amazon EC2 instances to ensure traffic is not routed to unhealthy or failing instances. Also, Elastic Load Balancing can automatically scale based on collected metrics.

There are several advantages of using Elastic Load Balancing. Because Elastic Load Balancing is a managed service, it scales in and out automatically to meet the demands of increased application traffic and is highly available within a region itself as a service. Elastic Load Balancing helps you achieve high availability for your applications by distributing traffic across healthy instances in multiple Availability Zones. Additionally, Elastic Load Balancing seamlessly integrates with the Auto Scaling service to automatically scale the Amazon EC2 instances behind the load balancer. Finally, Elastic Load Balancing is secure, working with Amazon Virtual Private Cloud (Amazon VPC) to route traffic internally between application tiers, allowing you to expose only Internet-facing public IP addresses. Elastic Load Balancing also supports integrated certificate management and SSL termination.

 Elastic Load Balancing is a highly available service itself and can be used to help build highly available architectures.

Types of Load Balancers

Elastic Load Balancing provides several types of load balancers for handling different kinds of connections including Internet-facing, internal, and load balancers that support encrypted connections.

Internet-Facing Load Balancers

An *Internet-facing load balancer* is, as the name implies, a load balancer that takes requests from clients over the Internet and distributes them to Amazon EC2 instances that are registered with the load balancer.

When you configure a load balancer, it receives a public DNS name that clients can use to send requests to your application. The DNS servers resolve the DNS name to your load balancer's public IP address, which can be visible to client applications.

 An AWS recommended best practice is always to reference a load balancer by its DNS name, instead of by the IP address of the load balancer, in order to provide a single, stable entry point.

Because Elastic Load Balancing scales in and out to meet traffic demand, it is not recommended to bind an application to an IP address that may no longer be part of a load balancer's pool of resources.

Elastic Load Balancing in Amazon VPC supports IPv4 addresses only. Elastic Load Balancing in EC2-Classic supports both IPv4 and IPv6 addresses.

Internal Load Balancers

In a multi-tier application, it is often useful to load balance between the tiers of the application. For example, an Internet-facing load balancer might receive and balance external traffic to the presentation or web tier whose Amazon EC2 instances then send its requests to a load balancer sitting in front of the application tier. You can use *internal load balancers* to route traffic to your Amazon EC2 instances in VPCs with private subnets.

HTTPS Load Balancers

You can create a load balancer that uses the SSL/Transport Layer Security (TLS) protocol for encrypted connections (also known as *SSL offload*). This feature enables traffic encryption between your load balancer and the clients that initiate HTTPS sessions, and for connections between your load balancer and your back-end instances. Elastic Load Balancing provides security policies that have predefined SSL negotiation configurations to use to negotiate connections between clients and the load balancer. In order to use SSL, you must install an SSL certificate on the load balancer that it uses to terminate the connection and then decrypt requests from clients before sending requests to the back-end Amazon EC2 instances. You can optionally choose to enable authentication on your back-end instances.

Elastic Load Balancing does not support *Server Name Indication* (SNI) on your load balancer. This means that if you want to host multiple websites on a fleet of Amazon EC2 instances behind Elastic Load Balancing with a single SSL certificate, you will need to add a *Subject Alternative Name (SAN)* for each website to the certificate to avoid site users seeing a warning message when the site is accessed.

Listeners

Every load balancer must have one or more *listeners* configured. A listener is a process that checks for connection requests—for example, a CNAME configured to the A record name of the load balancer. Every listener is configured with a protocol and a port (client to load balancer) for a front-end connection and a protocol and a port for the back-end (load balancer to Amazon EC2 instance) connection. Elastic Load Balancing supports the following

protocols:

- HTTP
- HTTPS
- TCP
- SSL

Elastic Load Balancing supports protocols operating at two different *Open System Interconnection (OSI)* layers. In the OSI model, Layer 4 is the transport layer that describes the TCP connection between the client and your back-end instance through the load balancer. Layer 4 is the lowest level that is configurable for your load balancer. Layer 7 is the application layer that describes the use of HTTP and HTTPS connections from clients to the load balancer and from the load balancer to your back-end instance.

The SSL protocol is primarily used to encrypt confidential data over insecure networks such as the Internet. The SSL protocol establishes a secure connection between a client and the back-end server and ensures that all the data passed between your client and your server is private.

Configuring Elastic Load Balancing

Elastic Load Balancing allows you to configure many aspects of the load balancer, including *idle connection timeout*, *cross-zone load balancing*, *connection draining*, *proxy protocol*, *sticky sessions,* and *health checks*. Configuration settings can be modified using either the AWS Management Console or a Command Line Interface (CLI). Some of the options are described next.

Idle Connection Timeout

For each request that a client makes through a load balancer, the load balancer maintains two connections. One connection is with the client and the other connection is to the back-end instance. For each connection, the load balancer manages an idle timeout that is triggered when no data is sent over the connection for a specified time period. After the idle timeout period has elapsed, if no data has been sent or received, the load balancer closes the connection.

By default, Elastic Load Balancing sets the idle timeout to 60 seconds for both connections. If an HTTP request doesn't complete within the idle timeout period, the load balancer closes the connection, even if data is still being transferred. You can change the idle timeout setting for the connections to ensure that lengthy operations, such as file uploads, have time to complete.

If you use HTTP and HTTPS listeners, we recommend that you enable the *keep-alive* option for your Amazon EC2 instances. You can enable keep-alive in your web server settings or in the kernel settings for your Amazon EC2 instances. Keep-alive, when enabled, allows the load balancer to reuse connections to your back-end instance, which reduces CPU utilization.

 To ensure that the load balancer is responsible for closing the connections to your back-end instance, make sure that the value you set for the keep-alive time is greater than the idle timeout setting on your load balancer.

Cross-Zone Load Balancing

To ensure that request traffic is routed evenly across all back-end instances for your load balancer, regardless of the Availability Zone in which they are located, you should enable cross-zone load balancing on your load balancer. Cross-zone load balancing reduces the need to maintain equivalent numbers of back-end instances in each Availability Zone and improves your application's ability to handle the loss of one or more back-end instances. However, it is still recommended that you maintain approximately equivalent numbers of instances in each Availability Zone for higher fault tolerance.

For environments where clients cache DNS lookups, incoming requests might favor one of the Availability Zones. Using cross-zone load balancing, this imbalance in the request load is spread across all available back-end instances in the region, reducing the impact of misconfigured clients.

Connection Draining

You should enable *connection draining* to ensure that the load balancer stops sending requests to instances that are deregistering or unhealthy, while keeping the existing connections open. This enables the load balancer to complete in-flight requests made to these instances.

When you enable connection draining, you can specify a maximum time for the load balancer to keep connections alive before reporting the instance as deregistered. The maximum timeout value can be set between 1 and 3,600 seconds (the default is 300 seconds). When the maximum time limit is reached, the load balancer forcibly closes connections to the deregistering instance.

Proxy Protocol

When you use TCP or SSL for both front-end and back-end connections, your load balancer forwards requests to the back-end instances without modifying the request headers. If you enable *Proxy Protocol*, a human-readable header is added to the request header with connection information such as the source IP address, destination IP address, and port numbers. The header is then sent to the back-end instance as part of the request.

Before using Proxy Protocol, verify that your load balancer is not behind a proxy server with Proxy Protocol enabled. If Proxy Protocol is enabled on both the proxy server and the load balancer, the load balancer adds another header to the request, which already has a header from the proxy server. Depending on how your back-end instance is configured, this duplication might result in errors.

Sticky Sessions

By default, a load balancer routes each request independently to the registered instance with

the smallest load. However, you can use the *sticky session* feature (also known as *session affinity*), which enables the load balancer to bind a user's session to a specific instance. This ensures that all requests from the user during the session are sent to the same instance.

The key to managing sticky sessions is to determine how long your load balancer should consistently route the user's request to the same instance. If your application has its own session cookie, you can configure Elastic Load Balancing so that the session cookie follows the duration specified by the application's session cookie. If your application does not have its own session cookie, you can configure Elastic Load Balancing to create a session cookie by specifying your own stickiness duration. Elastic Load Balancing creates a cookie named AWSELB that is used to map the session to the instance.

Health Checks

Elastic Load Balancing supports health checks to test the status of the Amazon EC2 instances behind an Elastic Load Balancing load balancer. The status of the instances that are healthy at the time of the health check is `InService`. The status of any instances that are unhealthy at the time of the health check is `OutOfService`. The load balancer performs health checks on all registered instances to determine whether the instance is in a healthy state or an unhealthy state. A health check is a ping, a connection attempt, or a page that is checked periodically. You can set the time interval between health checks and also the amount of time to wait to respond in case the health check page includes a computational aspect. Finally, you can set a threshold for the number of consecutive health check failures before an instance is marked as unhealthy.

Updates Behind an Elastic Load Balancing Load Balancer

Long-running applications will eventually need to be maintained and updated with a newer version of the application. When using Amazon EC2 instances running behind an Elastic Load Balancing load balancer, you may deregister these long-running Amazon EC2 instances associated with a load balancer manually and then register newly launched Amazon EC2 instances that you have started with the new updates installed.

Amazon CloudWatch

Amazon CloudWatch is a service that you can use to monitor your AWS resources and your applications in real time. With Amazon CloudWatch, you can collect and track metrics, create alarms that send notifications, and make changes to the resources being monitored based on rules you define.

For example, you might choose to monitor CPU utilization to decide when to add or remove Amazon EC2 instances in an application tier. Or, if a particular application-specific metric that is not visible to AWS is the best indicator for assessing your scaling needs, you can perform a `PUT` request to push that metric into Amazon CloudWatch. You can then use this custom metric to manage capacity.

You can specify parameters for a metric over a time period and configure alarms and automated actions when a threshold is reached. Amazon CloudWatch supports multiple types of actions such as sending a notification to an Amazon Simple Notification Service (Amazon SNS) topic or executing an Auto Scaling policy.

Amazon CloudWatch offers either basic or detailed monitoring for supported AWS products. *Basic monitoring* sends data points to Amazon CloudWatch every five minutes for a limited number of preselected metrics at no charge. *Detailed monitoring* sends data points to Amazon CloudWatch every minute and allows data aggregation for an additional charge. If you want to use detailed monitoring, you must enable it—basic is the default.

Amazon CloudWatch supports monitoring and specific metrics for most AWS Cloud services, including: Auto Scaling, Amazon CloudFront, Amazon CloudSearch, Amazon DynamoDB, Amazon EC2, Amazon EC2 Container Service (Amazon ECS), Amazon ElastiCache, Amazon Elastic Block Store (Amazon EBS), Elastic Load Balancing, Amazon Elastic MapReduce (Amazon EMR), Amazon Elasticsearch Service, Amazon Kinesis Streams, Amazon Kinesis Firehose, AWS Lambda, Amazon Machine Learning, AWS OpsWorks, Amazon Redshift, Amazon Relational Database Service (Amazon RDS), Amazon Route 53, Amazon SNS, Amazon Simple Queue Service (Amazon SQS), Amazon S3, AWS Simple Workflow Service (Amazon SWF), AWS Storage Gateway, AWS WAF, and Amazon WorkSpaces.

Read Alert

You may have an application that leverages Amazon DynamoDB, and you want to know when read requests reach a certain threshold and alert yourself with an email. You can do this by using `ProvisionedReadCapacityUnits` for the Amazon DynamoDB table for which you want to set an alarm. You simply set a threshold value during a number of consecutive periods and then specify email as the notification type. Now, when the threshold is sustained over the number of periods, your specified email will alert you to the read activity.

Amazon CloudWatch metrics can be retrieved by performing a `GET` request. When you use detailed monitoring, you can also aggregate metrics across a length of time you specify. Amazon CloudWatch does not aggregate data across regions but can aggregate across

Availability Zones within a region.

AWS provides a rich set of metrics included with each service, but you can also define custom metrics to monitor resources and events AWS does not have visibility into—for example, Amazon EC2 instance memory consumption and disk metrics that are visible to the operating system of the Amazon EC2 instance but not visible to AWS or application-specific thresholds running on instances that are not known to AWS. Amazon CloudWatch supports an Application Programming Interface (API) that allows programs and scripts to `PUT` metrics into Amazon CloudWatch as name-value pairs that can then be used to create events and trigger alarms in the same manner as the default Amazon CloudWatch metrics.

Amazon CloudWatch Logs can be used to monitor, store, and access log files from Amazon EC2 instances, AWS CloudTrail, and other sources. You can then retrieve the log data and monitor in real time for events—for example, you can track the number of errors in your application logs and send a notification if an error rate exceeds a threshold. Amazon CloudWatch Logs can also be used to store your logs in Amazon S3 or Amazon Glacier. Logs can be retained indefinitely or according to an aging policy that will delete older logs as no longer needed.

A *CloudWatch Logs agent* is available that provides an automated way to send log data to CloudWatch Logs for Amazon EC2 instances running Amazon Linux or Ubuntu. You can use the Amazon CloudWatch Logs agent installer on an existing Amazon EC2 instance to install and configure the CloudWatch Logs agent. After installation is complete, the agent confirms that it has started and it stays running until you disable it.

Amazon CloudWatch has some limits that you should keep in mind when using the service. Each AWS account is limited to 5,000 alarms per AWS account, and metrics data is retained for two weeks by default (at the time of this writing). If you want to keep the data longer, you will need to move the logs to a persistent store like Amazon S3 or Amazon Glacier. You should familiarize yourself with the limits for Amazon CloudWatch in the Amazon CloudWatch Developer Guide.

Auto Scaling

A distinct advantage of deploying applications to the cloud is the ability to launch and then release servers in response to variable workloads. Provisioning servers on demand and then releasing them when they are no longer needed can provide significant cost savings for workloads that are not steady state. Examples include a website for a specific sporting event, an end-of-month data-input system, a retail shopping site supporting flash sales, a music artist website during the release of new songs, a company website announcing successful earnings, or a nightly processing run to calculate daily activity.

Auto Scaling is a service that allows you to scale your Amazon EC2 capacity automatically by scaling out and scaling in according to criteria that you define. With Auto Scaling, you can ensure that the number of running Amazon EC2 instances increases during demand spikes or peak demand periods to maintain application performance and decreases automatically during demand lulls or troughs to minimize costs.

Embrace the Spike

Many web applications have unplanned load increases based on events outside of your control. For example, your company may get mentioned on a popular blog or television program driving many more people to visit your site than expected. Setting up Auto Scaling in advance will allow you to embrace and survive this kind of fast increase in the number of requests. Auto Scaling will scale up your site to meet the increased demand and then scale down when the event subsides.

Auto Scaling Plans

Auto Scaling has several schemes or plans that you can use to control how you want Auto Scaling to perform.

Maintain Current Instance Levels

You can configure your Auto Scaling group to maintain a minimum or specified number of running instances at all times. To maintain the current instance levels, Auto Scaling performs a periodic health check on running instances within an *Auto Scaling group*. When Auto Scaling finds an unhealthy instance, it terminates that instance and launches a new one.

Steady state workloads that need a consistent number of Amazon EC2 instances at all times can use Auto Scaling to monitor and keep that specific number of Amazon EC2 instances running.

Manual Scaling

Manual scaling is the most basic way to scale your resources. You only need to specify the change in the maximum, minimum, or desired capacity of your Auto Scaling group. Auto

Scaling manages the process of creating or terminating instances to maintain the updated capacity.

 Manual scaling out can be very useful to increase resources for an infrequent event, such as the release of a new game version that will be available for download and require a user registration. For extremely large-scale events, even the Elastic Load Balancing load balancers can be pre-warmed by working with your local solutions architect or AWS Support.

Scheduled Scaling

Sometimes you know exactly when you will need to increase or decrease the number of instances in your group, simply because that need arises on a predictable schedule. Examples include periodic events such as end-of-month, end-of-quarter, or end-of-year processing, and also other predictable, recurring events. Scheduled scaling means that scaling actions are performed automatically as a function of time and date.

 Recurring events such as end-of-month, quarter, or year processing, or scheduled and recurring automated load and performance testing, can be anticipated and Auto Scaling can be ramped up appropriately at the time of the scheduled event.

Dynamic Scaling

Dynamic scaling lets you define parameters that control the Auto Scaling process in a scaling policy. For example, you might create a policy that adds more Amazon EC2 instances to the web tier when the network bandwidth, measured by Amazon CloudWatch, reaches a certain threshold.

Auto Scaling Components

Auto Scaling has several components that need to be configured to work properly: a *launch configuration*, an *Auto Scaling group*, and an optional *scaling policy*.

Launch Configuration

A *launch configuration* is the template that Auto Scaling uses to create new instances, and it is composed of the configuration name, *Amazon Machine Image (AMI)*, Amazon EC2 instance type, security group, and instance key pair. Each Auto Scaling group can have only one launch configuration at a time.

The CLI command that follows will create a launch configuration with the following attributes:

Name: `myLC`

AMI: `ami-0535d66c`

Instance type: `m3.medium`

Security groups: `sg-f57cde9d`

Instance key pair: `myKeyPair`

```
> aws autoscaling create-launch-configuration --launch-configuration-name myLC --
image-id ami-0535d66c --instance-type m3.medium --security-groups sg-f57cde9d --
key-name myKeyPair
```

Security groups for instances launched in EC2-Classic may be referenced by security group name such as “SSH” or “Web” if that is what they are named, or you can reference the security group IDs, such as `sg-f57cde9d`. If you launched the instances in Amazon VPC, which is recommended, you must use the security group IDs to reference the security groups you want associated with the instances in an Auto Scaling launch configuration.

The default limit for launch configurations is 100 per region. If you exceed this limit, the call to create-launch-configuration will fail. You may view and update this limit by running `describe-account-limits` at the command line, as shown here.

```
> aws autoscaling describe-account-limits
```

Auto Scaling may cause you to reach limits of other services, such as the default number of Amazon EC2 instances you can currently launch within a region, which is 20. When building more complex architectures with AWS, it is important to keep in mind the service limits for all AWS Cloud services you are using.

When you run a command using the CLI and it fails, check your syntax first. If that checks out, verify the limits for the command you are attempting, and check to see that you have not exceeded a limit. Some limits can be raised and usually defaulted to a reasonable value to limit a race condition, an errant script running in a loop, or other similar automation that might cause unintended high usage and billing of AWS resources. AWS service limits can be viewed in the AWS General Reference Guide under AWS Service Limits. You can raise your limits by creating a support case at the AWS Support Center online and then choosing Service Limit Increase under Regarding. Then fill in the appropriate service and limit to increase value in the online form.

Auto Scaling Group

An Auto Scaling group is a collection of Amazon EC2 instances managed by the Auto Scaling service. Each Auto Scaling group contains configuration options that control when Auto Scaling should launch new instances and terminate existing instances. An Auto Scaling group must contain a name and a minimum and maximum number of instances that can be in the group. You can optionally specify desired capacity, which is the number of instances that the group must have at all times. If you don’t specify a desired capacity, the default desired capacity is the minimum number of instances that you specify.

The CLI command that follows will create an Auto Scaling group that references the previous launch configuration and includes the following specifications:

Name: `myASG`

Launch configuration: `myLC`

Availability Zones: `us-east-1a` and `us-east-1c`

Minimum size: `1`

Desired capacity: `3`

Maximum capacity: `10`

Load balancers: `myELB`

```
> aws autoscaling create-auto-scaling-group --auto-scaling-group-name myASG --
launch-configuration-name myLC --availability-zones us-east-1a, us-east-1c --min-
size 1 --max-size 10 --desired-capacity 3 --load-balancer-names myELB
```

Figure 5.1 depicts deployed AWS resources after a load balancer named `myELB` is created and the launch configuration `myLC` and Auto Scaling Group `myASG` are set up.

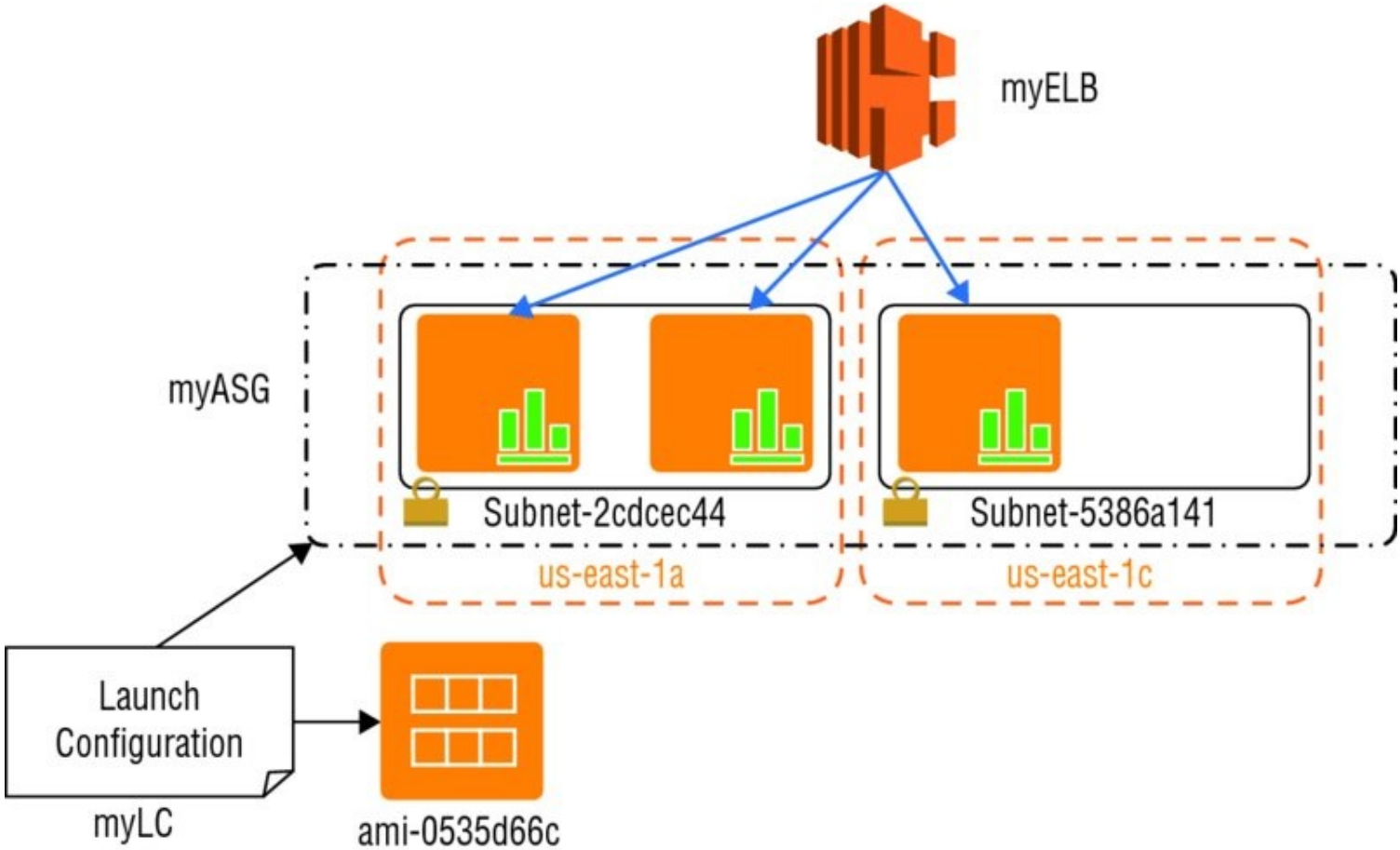


FIGURE 5.1 Auto Scaling group behind an Elastic Load Balancing load balancer

An Auto Scaling group can use either On-Demand or Spot Instances as the Amazon EC2 instances it manages. On-Demand is the default, but Spot Instances can be used by referencing a maximum bid price in the launch configuration (`–spot-price "0.15"`) associated with the Auto Scaling group. You may change the bid price by creating a new launch configuration with the new bid price and then associating it with your Auto Scaling group. If instances are available at or below your bid price, they will be launched in your Auto Scaling group. Spot Instances in an Auto Scaling group follow the same guidelines as Spot

Instances outside an Auto Scaling group and require applications that are flexible and can tolerate Amazon EC2 instances that are terminated with short notice, for example, when the Spot price rises above the bid price you set in the launch configuration. A launch configuration can reference On-Demand Instances or Spot Instances, but not both.

Spot On!

Auto Scaling supports using cost-effective Spot Instances. This can be very useful when you are hosting sites where you want to provide additional compute capacity but are price constrained. An example is a “freemium” site model where you may offer some basic functionality to users for free and additional functionality for premium users who pay for use. Spot Instances can be used for providing the basic functionality when available by referencing a maximum bid price in the launch configuration `(–spot-price "0.15")` associated with the Auto Scaling group.

Scaling Policy

You can associate Amazon CloudWatch alarms and *scaling policies* with an Auto Scaling group to adjust Auto Scaling dynamically. When a threshold is crossed, Amazon CloudWatch sends alarms to trigger changes (scaling in or out) to the number of Amazon EC2 instances currently receiving traffic behind a load balancer. After the Amazon CloudWatch alarm sends a message to the Auto Scaling group, Auto Scaling executes the associated policy to scale your group. The policy is a set of instructions that tells Auto Scaling whether to scale out, launching new Amazon EC2 instances referenced in the associated launch configuration, or to scale in and terminate instances.

There are several ways to configure a scaling policy: You can increase or decrease by a specific number of instances, such as adding two instances; you can target a specific number of instances, such as a maximum of five total Amazon EC2 instances; or you can adjust based on a percentage. You can also scale by steps and increase or decrease the current capacity of the group based on a set of scaling adjustments that vary based on the size of the alarm threshold trigger.

You can associate more than one scaling policy with an Auto Scaling group. For example, you can create a policy using the trigger for CPU utilization, called *CPULoad*, and the CloudWatch metric *CPUUtilization* to specify scaling out if CPU utilization is greater than 75 percent for two minutes. You could attach another policy to the same Auto Scaling group to scale in if CPU utilization is less than 40 percent for 20 minutes.

The following CLI commands will create the scaling policy just described.

```
> aws autoscaling put-scaling-policy --auto-scaling-group-name myASG --policy-name
CPULoadScaleOut --scaling-adjustment 1 --adjustment-type ChangeInCapacity --
cooldown 30  > aws autoscaling put-scaling-policy --auto-scaling-group-name myASG -
-policy-name CPULoadScaleIn --scaling-adjustment -1 --adjustment-type
ChangeInCapacity --cooldown 600
```

The following CLI commands will associate Amazon CloudWatch alarms for scaling out and scaling in with the scaling policy, as shown in Figure 5.2. In this example, the Amazon CloudWatch alarms reference the scaling policy by Amazon Resource Name (ARN).

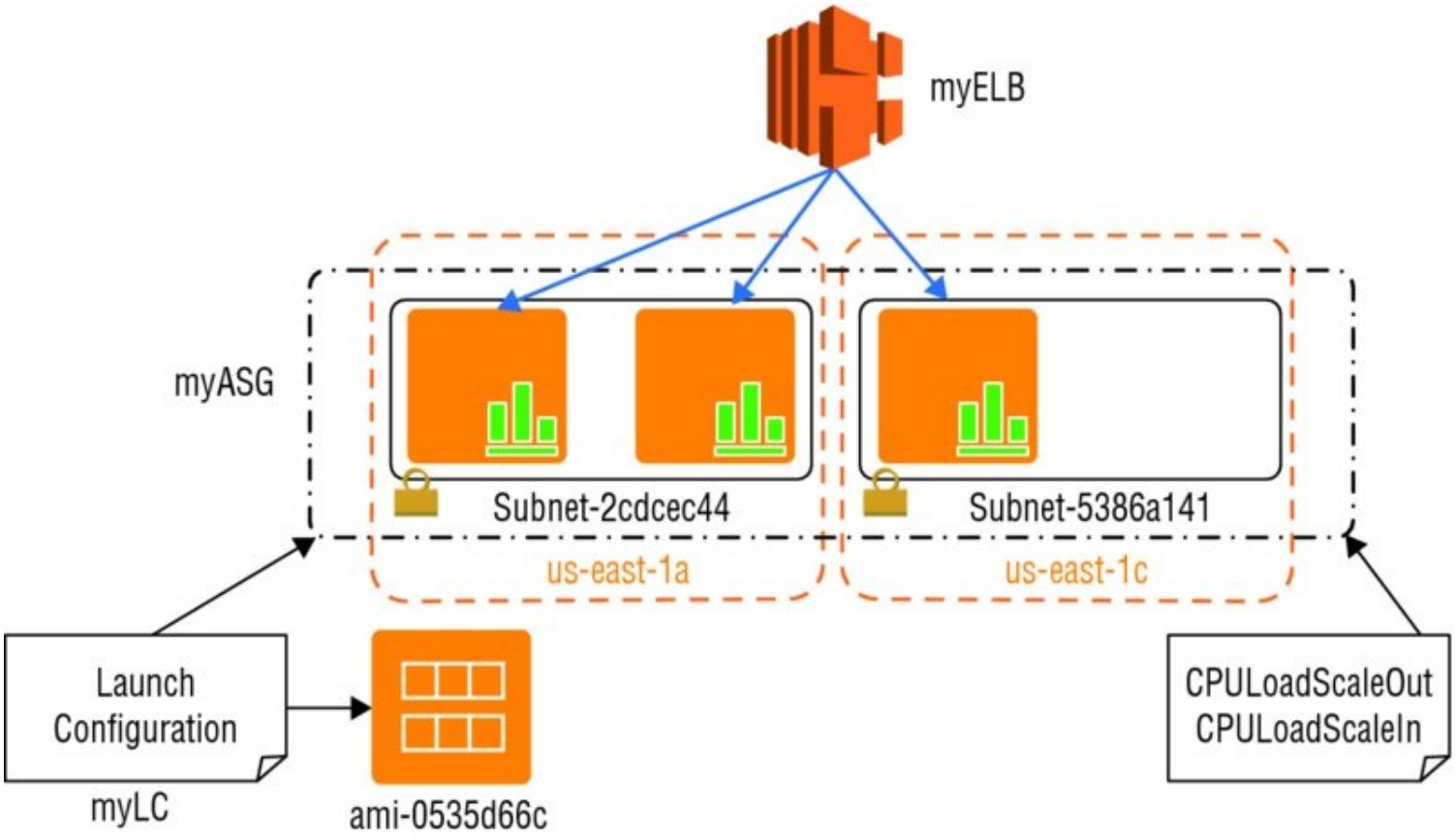


FIGURE 5.2 Auto Scaling group with policy

```
> aws cloudwatch put-metric-alarm --alarm name capacityAdd --metric-name
CPUUtilization --namespace AWS/EC2 --statistic Average –-period 300 --threshold 75
--comparison-operator GreaterThanOrEqualToThreshold --dimensions
"Name=AutoScalingGroupName, Value=myASG" --evaluation-periods 1 --alarm-actions
arn:aws:autoscaling:us-east-1:123456789012:scalingPolicy:12345678-90ab-cdef-
1234567890ab:autoScalingGroupName/myASG:policyName/CPULoadScaleOut --unit Percent
> aws cloudwatch put-metric-alarm --alarm name capacityReduce --metric-name
CPUUtilization --namespace AWS/EC2 --statistic Average --period 1200 --threshold 40
--comparison-operator GreaterThanOrEqualToThreshold --dimensions
"Name=AutoScalingGroupName, Value=myASG" --evaluation-periods 1 --alarm-actions
arn:aws:autoscaling:us-east-1:123456789011:scalingPolicy:11345678-90ab-cdef-
1234567890ab:autoScalingGroupName/myASG:policyName/CPULoadScaleIn --unit Percent
```

If the scaling policy defined in the previous paragraph is associated with the Auto Scaling group named `myASG`, and the CPU utilization is over 75 percent for more than five minutes, as shown in Figure 5.3, a new Amazon EC2 instance will be launched and attached to the load balancer named `myELB`.

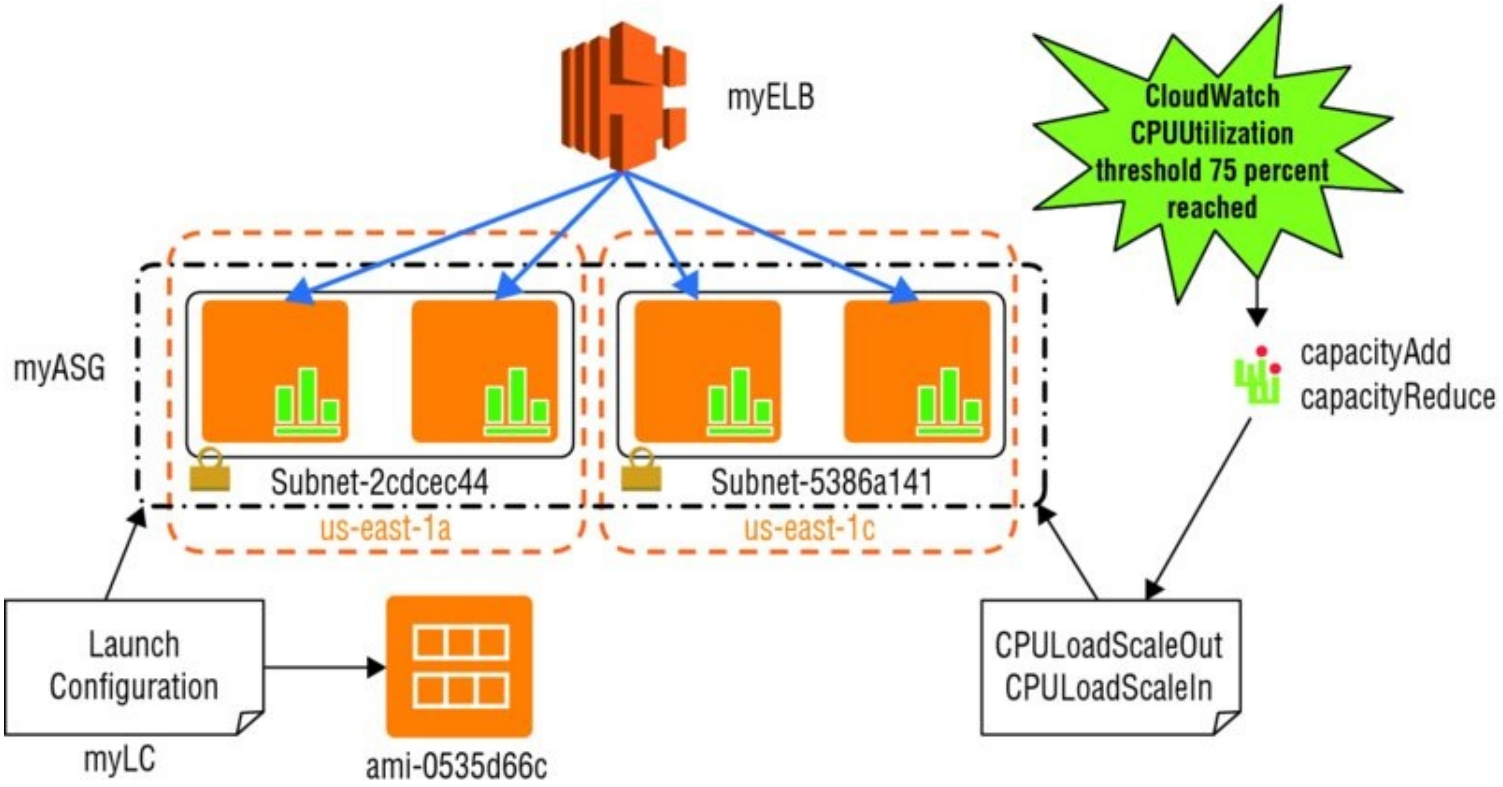


FIGURE 5.3 Amazon CloudWatch alarm triggering scaling out

A recommended best practice is to scale out quickly and scale in slowly so you can respond to bursts or spikes but avoid inadvertently terminating Amazon EC2 instances too quickly, only having to launch more Amazon EC2 instances if the burst is sustained. Auto Scaling also supports a *cooldown period*, which is a configurable setting that determines when to suspend scaling activities for a short time for an Auto Scaling group.

If you start an Amazon EC2 instance, you will be billed for one full hour of running time. Partial instance hours consumed are billed as full hours. This means that if you have a permissive scaling policy that launches, terminates, and relaunches many instances an hour, you are billing a full hour for each and every instance you launch, even if you terminate some of those instances in less than hour. A recommended best practice for cost effectiveness is to scale out quickly when needed but scale in more slowly to avoid having to relaunch new and separate Amazon EC2 instances for a spike in workload demand that fluctuates up and down within minutes but generally continues to need more resources within an hour.


Scale out quickly; scale in slowly.

It is important to consider bootstrapping for Amazon EC2 instances launched using Auto Scaling. It takes time to configure each newly launched Amazon EC2 instance before the instance is healthy and capable of accepting traffic. Instances that start and are available for load faster can join the capacity pool more quickly. Furthermore, instances that are more stateless instead of stateful will more gracefully enter and exit an Auto Scaling group.

Rolling Out a Patch at Scale

In large deployments of Amazon EC2 instances, Auto Scaling can be used to make rolling out a patch to your instances easy. The launch configuration associated with the Auto Scaling group may be modified to reference a new AMI and even a new Amazon EC2 instance if needed. Then you can deregister or terminate instances one at a time or in small groups, and the new Amazon EC2 instances will reference the new patched AMI.

Summary

This chapter introduced three services:

- Elastic Load Balancing, which is used to distribute traffic across a group of Amazon EC2 instances in one or more Availability Zones to achieve greater levels of fault tolerance for your applications.
- Amazon CloudWatch, which monitors resources and applications. Amazon CloudWatch is used to collect and track metrics, create alarms that send notifications, and make changes to resources being monitored based on rules you define.
- Auto Scaling, which allows you to automatically scale your Amazon EC2 capacity out and in using criteria that you define.

These three services can be used very effectively together to create a highly available application with a resilient architecture on AWS.

Exam Essentials

Understand what the Elastic Load Balancing service provides. Elastic Load Balancing is a highly available service that distributes traffic across Amazon EC2 instances and includes options that provide flexibility and control of incoming requests to Amazon EC2 instances.

Know the types of load balancers the Elastic Load Balancing service provides and when to use each one. An Internet-facing load balancer is, as the name implies, a load balancer that takes requests from clients over the Internet and distributes them to Amazon EC2 instances that are registered with the load balancer.

An internal load balancer is used to route traffic to your Amazon EC2 instances in VPCs with private subnets.

An HTTPS load balancer is used when you want to encrypt data between your load balancer and the clients that initiate HTTPS sessions and for connections between your load balancer and your back-end instances.

Know the types of listeners the Elastic Load Balancing service provides and the use case and requirements for using each one. A listener is a process that checks for connection requests. It is configured with a protocol and a port for front-end (client to load balancer) connections and a protocol and a port for back-end (load balancer to back-end instance) connections.

Understand the configuration options for Elastic Load Balancing. Elastic Load Balancing allows you to configure many aspects of the load balancer, including idle connection timeout, cross-zone load balancing, connection draining, proxy protocol, sticky sessions, and health checks.

Know what an Elastic Load Balancing health check is and why it is important. Elastic Load Balancing supports health checks to test the status of the Amazon EC2 instances behind an Elastic Load Balancing load balancer.

Understand what the amazon CloudWatch service provides and what use cases there are for using it. Amazon CloudWatch is a service that you can use to monitor your AWS resources and your applications in real time. With Amazon CloudWatch, you can collect and track metrics, create alarms that send notifications, and make changes to the resources being monitored based on rules you define.

For example, you might choose to monitor CPU utilization to decide when to add or remove Amazon EC2 instances in an application tier. Or, if a particular application-specific metric that is not visible to AWS is the best indicator for assessing your scaling needs, you can perform a PUT request to push that metric into Amazon CloudWatch. You can then use this custom metric to manage capacity.

Know the differences between the two types of monitoring—basic and detailed—for Amazon CloudWatch. Amazon CloudWatch offers basic or detailed monitoring for supported AWS products. Basic monitoring sends data points to Amazon CloudWatch every five minutes for a limited number of preselected metrics at no charge. Detailed monitoring sends data points to Amazon CloudWatch every minute and allows data aggregation for an

additional charge. If you want to use detailed monitoring, you must enable it—basic is the default.

Understand Auto Scaling and why it is an important advantage of the AWS Cloud. A distinct advantage of deploying applications to the cloud is the ability to launch and then release servers in response to variable workloads. Provisioning servers on demand and then releasing them when they are no longer needed can provide significant cost savings for workloads that are not steady state.

Know when and why to use Auto Scaling. Auto Scaling is a service that allows you to scale your Amazon EC2 capacity automatically by scaling out and scaling in according to criteria that you define. With Auto Scaling, you can ensure that the number of running Amazon EC2 instances increases during demand spikes or peak demand periods to maintain application performance and decreases automatically during demand lulls or troughs to minimize costs.

Know the supported Auto Scaling plans. Auto Scaling has several schemes or plans that you can use to control how you want Auto Scaling to perform. The Auto Scaling plans are named Maintain Current Instant Levels, Manual Scaling, Scheduled Scaling, and Dynamic Scaling.

Understand how to build an Auto Scaling launch configuration and an Auto Scaling group and what each is used for. A launch configuration is the template that Auto Scaling uses to create new instances and is composed of the configuration name, AMI, Amazon EC2 instance type, security group, and instance key pair.

Know what a scaling policy is and what use cases to use it for. A scaling policy is used by Auto Scaling with CloudWatch alarms to determine when your Auto Scaling group should scale out or scale in. Each CloudWatch alarm watches a single metric and sends messages to Auto Scaling when the metric breaches a threshold that you specify in your policy.

Understand how Elastic Load Balancing, amazon CloudWatch, and Auto Scaling are used together to provide dynamic scaling. Elastic Load Balancing, Amazon CloudWatch, and Auto Scaling can be used together to create a highly available application with a resilient architecture on AWS.

AWS®

Certified Developer

Official Study Guide

Associate (DVA-C01) Exam

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

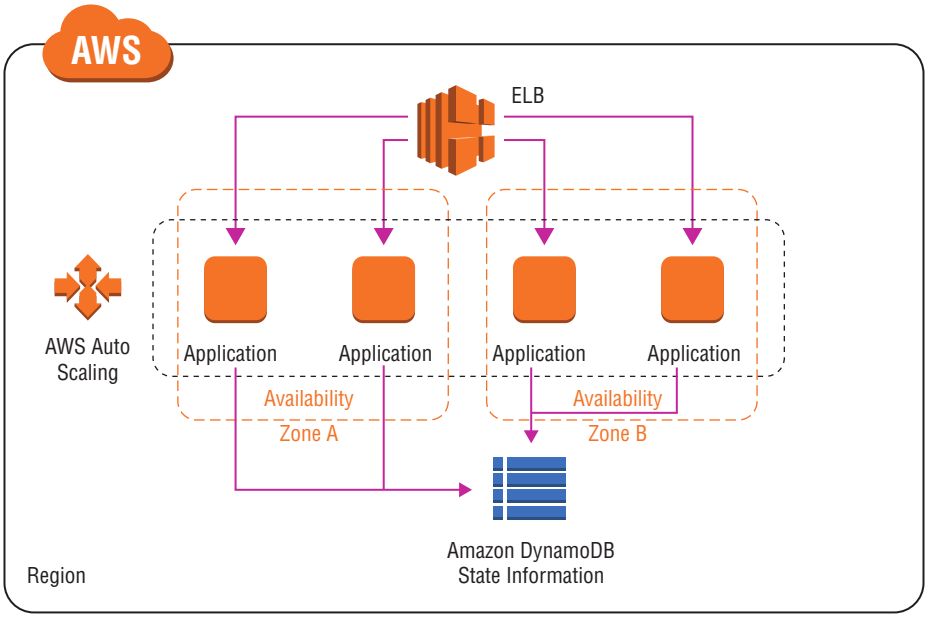
Load balancing is an integral part to directing and managing traffic among your instances. As you launch applications in your environments, you will want them to have high performance and high availability for your users. To enable both of these features, a load balancer will be necessary.

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:

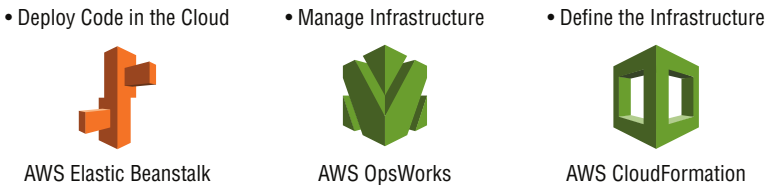
- Multiple Availability Zones and AWS Regions.
- Health check and failover mechanism.
- Stateless application that stores the session state in a cache server or database.
- AWS services that help you to achieve your goal. For example, Auto Scaling helps you maintain high availability and scalability.

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

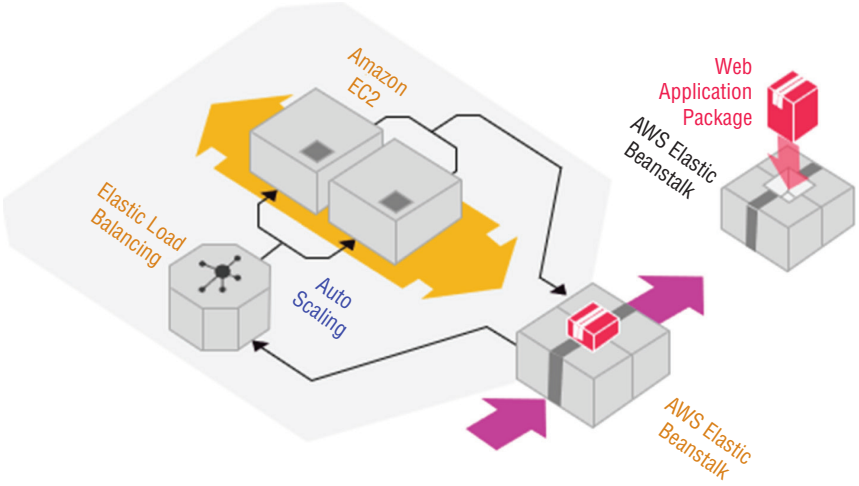
- Environments
- Application versions
- Environment configurations

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.

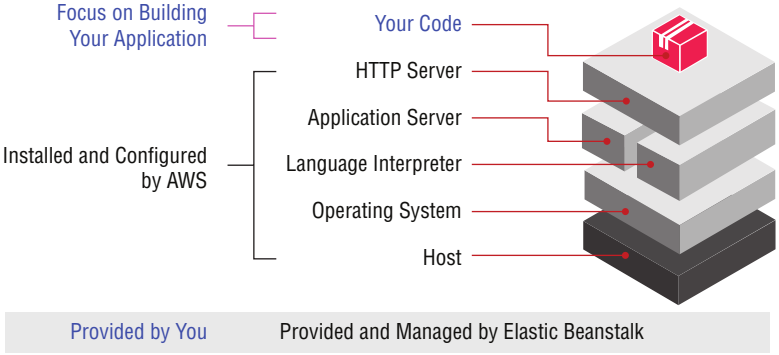
- Single-instance (development, low cost)
- Load balanced, AWS Auto Scaling (production)

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

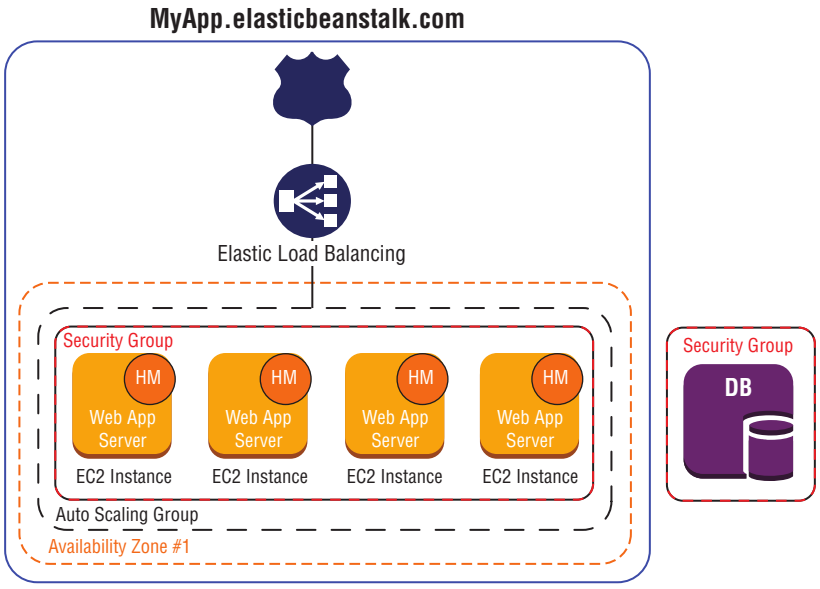
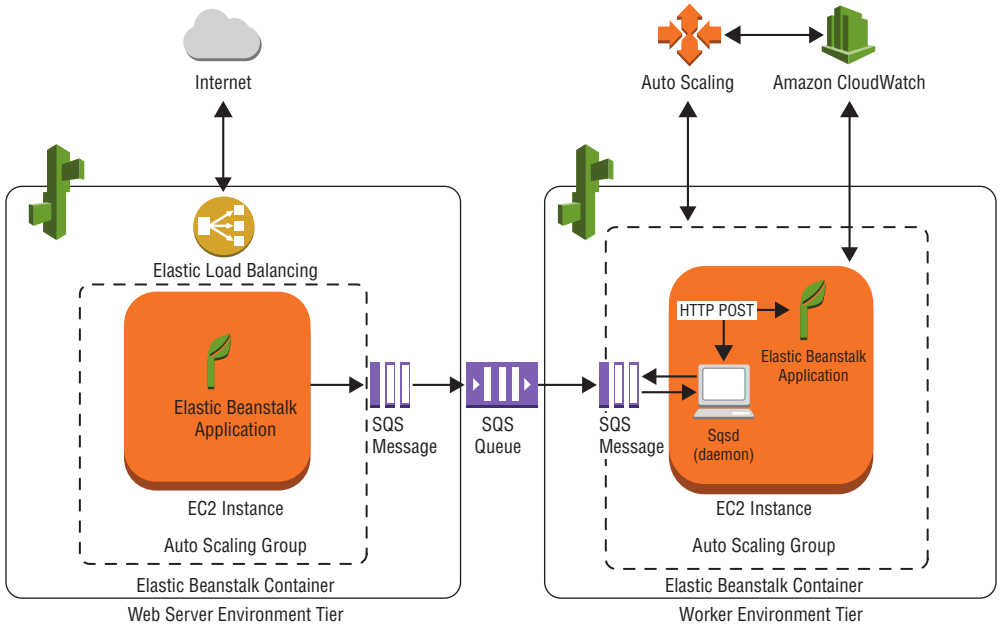


Figure 6.9 shows a worker environment architecture, where AWS resources create configurations, such as Auto Scaling groups, Amazon EC2 instances, and an IAM role, to manage resources for your worker applications.

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

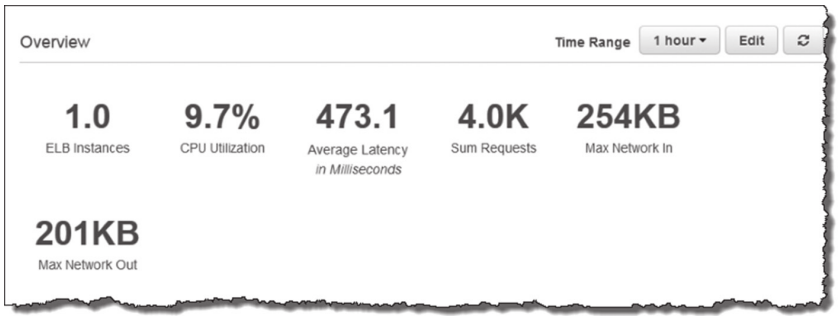


Figure 6.11 shows an example of the graphs that you can view.

FIGURE 6.11 Metrics for monitoring on AWS Elastic Beanstalk

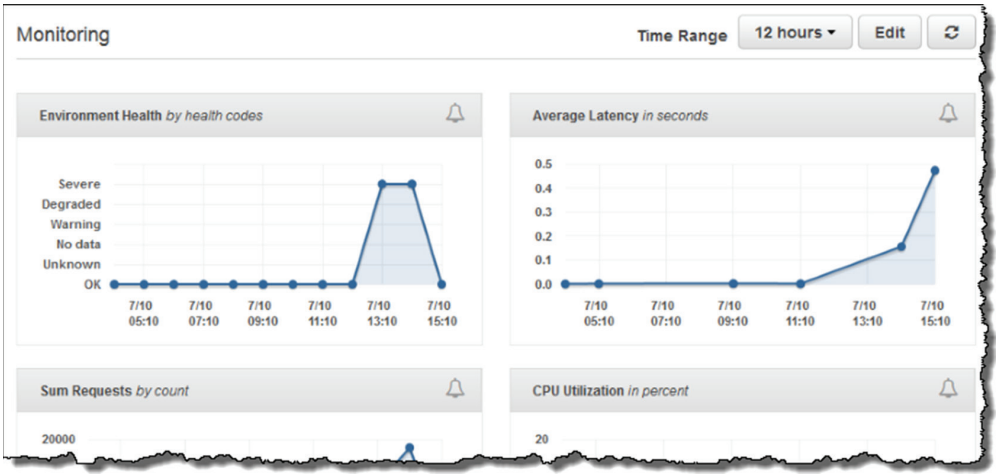


Table 6.3 defines the AWS Elastic Beanstalk Monitoring page colors.

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.

When you package dependencies for multiple cookbooks in the parent directory of the cookbooks, create a `Berksfile` such as this:

```
source "https://supermarket.chef.io"
cookbook "server-app", path: "./server-app"
cookbook "server-utils", path: "./server-utils"
```

After you package the dependencies, run the `berks package` command from this directory to download and dependencies for your cookbooks.

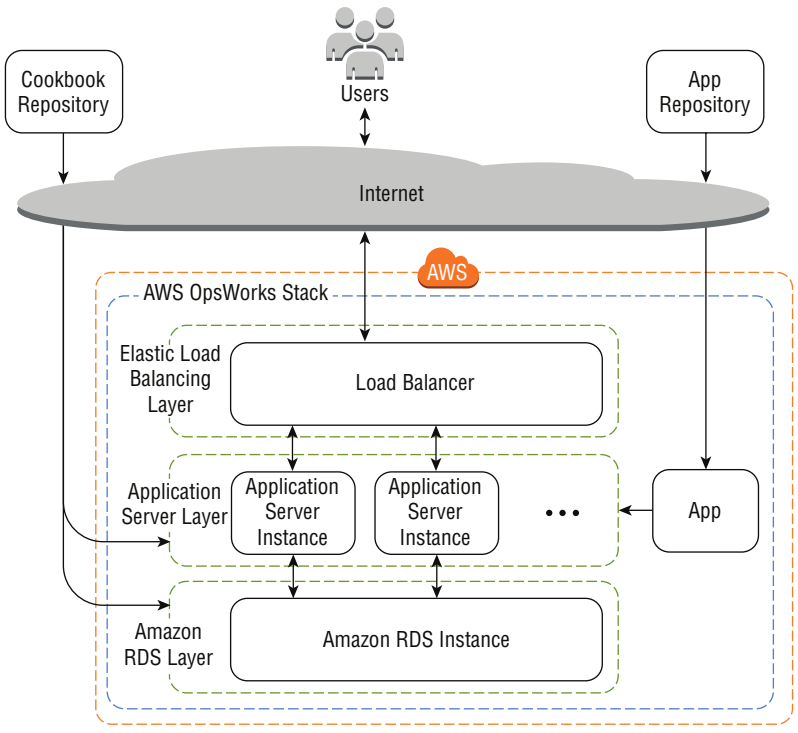
```
berks package cookbooks.tar.gz
```

Stack

A typical workload in AWS will include systems for various purposes, such as load balancers, application servers, proxy servers, databases, and more. The set of Amazon EC2 on-premises instances, Amazon RDS, Elastic Load Balancing, and other systems make up a stack. You can organize stacks across an enterprise.

Suppose you have a single application with dev, test, and production environments. Each of these environments has a stack that enables you to separate resources to ensure stability of changes. You group resources into stacks by logical or functional purposes. The example stack in Figure 9.4 includes three layers, a cookbook repository, an application repository, and one app to deploy to the application server instances. This stack manages a full application available to users over the Internet.

FIGURE 9.4 Example stack structure



When you create a new stack, you will have the option to set the stack's properties.

Stack Name

Stack Name identifies stacks in the AWS OpsWorks Stacks console. Since this name is not unique, AWS OpsWorks assigns a Globally Unique Identifier (GUID) to the stack after you create it.

API Endpoint Region

AWS OpsWorks associates a stack with either a global endpoint or one of multiple regional endpoints. When you create a resource in the stack, such as an instance, it is available only from the endpoint you specify when you create the stack. For example, if a stack is created with the global "classic" endpoint, any instances will be accessible only by AWS OpsWorks Stacks that use the global API endpoint in the US East (N. Virginia) region. Resources are not available across regional endpoints.

Amazon Virtual Private Cloud

Stacks can create and manage instances in Amazon EC2 Classic or an Amazon Virtual Private Cloud (Amazon VPC). When you select an Amazon VPC, you will be able to specify in what subnets to deploy instances when they are created.

Default Operating System

AWS OpsWorks Stacks supports many built-in Linux operating systems and Windows Server (only in Chef 12.2 stacks). If there is a custom Amazon Machine Images (AMI) you want to use, you must configure other tasks on the AMI to make it compatible with AWS OpsWorks Stacks. You must base the custom AMI off an AMI that AWS OpsWorks supports.

- The AMI must support `cloud-init`.
- The AMI must support the instance types you plan to launch.
- The AMI must utilize a 64-bit operating system.

Layer

A *layer* acts as a subset of instances or resources in a stack. Layers act as groups of instances or resources based on a common function. This is especially important, as the Chef recipe code applies to a layer and all instances in a layer. A layer is the point where any configuration of nodes will be set, such as what Chef recipes to execute at each lifecycle hook. A layer can contain any one or more nodes, and a node must be a member of one or more layers. When a node is a member of multiple layers, it will run any recipes you configure for each lifecycle event for both layers in the layer and recipe order you specify.

From the point of view of a Chef Server installation, a layer is synonymous with a Chef Role. In the `node` object, the layer and role data are equivalent. This is primarily to ensure compatibility with open-source cookbooks that are not written specifically for AWS OpsWorks Stacks.

Elastic Load Balancing

After a layer is created, any elastic load balancers in the same region associate with the layer. Any instances that come online in the layer will automatically register with the load balancer. The instances will also deregister from the load balancer when they go offline.

Elastic IP Addresses

You can configure layers to assign public or elastic IP addresses to instances when they come online. For Amazon Elastic Block Store (Amazon EBS) backed instances, the IP address will remain assigned after the instance stops and starts again. For instance-store backed instances, the IP address may not be the same as the original AWS OpsWorks instance.

Amazon EBS Volumes

Linux stacks include the option to assign one or more Amazon EBS volumes to a layer. In the process, you configure the mount point, size, Redundant Array of Independent Disks (RAID) configuration, volume type, Input/Output Operations Per Second (IOPS), and encryption settings. When new instances start in the layer, AWS OpsWorks Stacks will attempt to create an Amazon EBS volume with the configuration and attach it to the instance. Through the instance's setup lifecycle event, AWS OpsWorks Stacks runs a Chef cookbook to mount the volume to the instance. When the volumes add or remove volumes to or from a layer, only new instances will receive the configuration updates. Existing instances' volumes do not change.

Only Chef 11.10 stacks support RAID configurations.

Amazon RDS Layer

Amazon RDS layers pass connection information to an existing Amazon RDS instance. When you associate an Amazon RDS instance to a stack, it is assigned to an app. This passes the connection information to the instances via the app's `deploy` attributes, and you can access the data within your Chef recipes with `node[:deploy][:app_name][:database] hash`.

You can associate a single Amazon RDS instance with multiple apps in the same stack. However, you cannot associate multiple Amazon RDS instances with the same app. If your application needs to connect to multiple databases, use custom JSON to include the connection information for the other database(s).

Amazon ECS Cluster Layer

Amazon ECS cluster layers provide configuration management capabilities to Linux instances in your Amazon ECS cluster. You can associate a single cluster with a single stack at a time. To create this layer, you must register the cluster with the stack. After this, it will appear in the Layer type drop-down list of available clusters from which to create a layer, as shown in Figure 9.5.

Statistics can be used to gain insight into the health of your application and to help you determine the correct settings for various configurations. For example, you may want to implement automatic scaling on your fleet of Amazon EC2 instances in order to avoid having to launch and terminate instances manually. To do so, you must configure an Auto Scaling group. Configuration settings for an Auto Scaling group include the minimum, desired, and maximum number of instances to run in your account. By monitoring statistics over time, you can determine the minimum and maximum number of instances needed to support the average, minimum, and maximum workload.

CloudWatch statistics provide a powerful way to process large amounts of metrics at scale and present insightful data that is easy to consume. Now that you understand how CloudWatch metrics work and are organized, explore the metrics available.

Aggregations

CloudWatch aggregates metrics according to the period of time you specify when retrieving statistics. When you request this statistic, you also can have CloudWatch filter the data points based on the dimensions of the metrics. For example, in Amazon DynamoDB, metrics are fetched across all DynamoDB operations. You can specify a filter on the dimension `operations` to exclude specific operations, such as `GetItem` requests. CloudWatch does not aggregate data across regions.

Available Metrics

Table 15.1 describes the available metrics for Elastic Load Balancing resources. To discover all of the available metrics, refer to the AWS documentation.

TABLE 15.1 Elastic Load Balancing Metrics

Namespace	`AWS/ELB`
	`AWS/ApplicationELB`
	`AWS/NetworkELB`

Dimensions	`LoadBalancerName`: name of the load balancer
Key metrics	▪ `HealthyHostCount`: number of responding backend servers ▪ `RequestCount`: number of IPv4 and IPv6 requests ▪ `ActiveConnectionCount`: total number of concurrent active connections from clients

Table 15.2 describes the available Amazon EC2 metrics.

applications that have stable demand patterns and for ones that experience hourly, daily, or weekly variability in usage. AWS Auto Scaling is useful for applications that show steady demand patterns and that experience frequent variations in usage.

Amazon EC2 Auto Scaling

Amazon EC2 Auto Scaling helps you scale your Amazon EC2 instances and Spot Fleet capacity up or down automatically according to conditions that you define. AWS Auto Scaling is generally used with Elastic Load Balancing to distribute incoming application traffic across multiple Amazon EC2 instances in an AWS Auto Scaling group. AWS Auto Scaling is triggered using scaling plans that include policies that define how to scale (manual, schedule, and demand spikes) and the metrics and alarms to monitor in Amazon CloudWatch.

CloudWatch metrics are used to trigger the scaling event. These metrics can be standard Amazon EC2 metrics, such as CPU utilization, network throughput, Elastic Load Balancing observed request and response latency, and even custom metrics that might originate from application code on your Amazon EC2 instances.

You can use Amazon EC2 Auto Scaling to increase the number of Amazon EC2 instances automatically during demand spikes to maintain performance and decrease capacity during lulls to reduce costs.

Dynamic Scaling

The *dynamic scaling* capabilities of Amazon EC2 Auto Scaling refers to the functionality that automatically increases or decreases capacity based on load or other metrics. For example, if your CPU spikes above 80 percent (and you have an alarm set up), Amazon EC2 Auto Scaling can add a new instance dynamically, reducing the need to provision Amazon EC2 capacity manually in advance. Alternatively, you could set a target value by using the new `Request Count Per Target` metric from Application Load Balancer, a load balancing option for the Elastic Load Balancing service. Amazon EC2 Auto Scaling will then automatically adjust the number of Amazon EC2 instances as needed to maintain your target.

Scheduled Scaling

Scaling based on a schedule allows you to scale your application ahead of known load changes, such as the start of business hours, thus ensuring that resources are available when users arrive, or in typical development or test environments that run only during defined business hours or periods of time.

You can use APIs to scale the size of resources within an environment (vertical scaling). For example, you could scale up a production system by changing the instance size or class. This can be achieved by stopping and starting the instance and selecting the different instance size or class. You can also apply this technique to other resources, such as EBS volumes, which can be modified to increase size, adjust performance (IOPS), or change the volume type while in use.

Fleet Management

Fleet management refers to the functionality that automatically replaces unhealthy instances in your application, maintains your fleet at the desired capacity, and balances instances across Availability Zones. Amazon EC2 Auto Scaling fleet management ensures that your application is able to receive traffic and that the instances themselves are working properly. When AWS Auto Scaling detects a failed `health check`, it can replace the instance automatically.

Instances Purchasing Options

With Amazon EC2 Auto Scaling, you can provision and automatically scale instances across purchase options, Availability Zones, and instance families in a single application to optimize scale, performance, and cost. You can include Spot Instances with On-Demand and Reserved Instances in a single AWS Auto Scaling group to save up to 90 percent on compute. You have the option to define the desired split between On-Demand and Spot capacity, select which instance types work for your application, and specify preferences for how Amazon EC2 Auto Scaling should distribute the AWS Auto Scaling group capacity within each purchasing model.

Golden Images

A *golden image* is a snapshot of a particular state of a resource, such as an Amazon EC2 instance, Amazon EBS volumes, and an Amazon RDS DB instance. You can customize an Amazon EC2 instance and then save its configuration by creating an Amazon Machine Image (AMI). You can launch as many instances from the AMI as you need, and they will all include those customizations. A golden image results in faster start times and removes dependencies to configuration services or third-party repositories. This is important in auto-scaled environments in which you want to be able to launch additional resources in response to changes in demand quickly and reliably.

AWS Auto Scaling

AWS Auto Scaling monitors your applications and automatically adjusts capacity of all scalable resources to maintain steady, predictable performance at the lowest possible cost. Using AWS Auto Scaling, you can set up application scaling for multiple resources across multiple services in minutes.

AWS Auto Scaling automatically scales resources for other AWS services, including Amazon ECS, Amazon DynamoDB, Amazon Aurora, Amazon EC2 Spot Fleet requests, and Amazon EC2 Scaling groups.

If you have an application that uses one or more scalable resources and experiences variable load, use AWS Auto Scaling. A good example would be an ecommerce web application that receives variable traffic throughout the day. It follows a standard three-tier architecture with Elastic Load Balancing for distributing incoming traffic, Amazon EC2 for the compute layer, and Amazon DynamoDB for the data layer. In this case, AWS Auto Scaling scales one or more Amazon EC2 Auto Scaling groups and DynamoDB tables that are powering the application in response to the demand curve.

AWS Auto Scaling continually monitors your applications to make sure that they are operating at your desired performance levels. When demand spikes, AWS Auto Scaling automatically increases the capacity of constrained resources so that you maintain a high quality of service.

AWS Auto Scaling bases its scaling recommendations on the most popular scaling metrics and thresholds used for AWS Auto Scaling. It also recommends safe guardrails for scaling by providing recommendations for the minimum and maximum sizes of the resources. This way, you can get started quickly and then fine-tune your scaling strategy over time, allowing you to optimize performance, costs, or balance between them.

The *predictive scaling* feature uses machine learning algorithms to detect changes in daily and weekly patterns, automatically adjusting their forecasts. This removes the need for the manual adjustment of AWS Auto Scaling parameters as cyclicality changes over time, making AWS Auto Scaling simpler to configure, and provides more accurate capacity provisioning. Predictive scaling results in lower cost and more responsive applications.

DynamoDB Auto Scaling

DynamoDB automatic scaling uses the AWS Auto Scaling service to adjust provisioned throughput capacity dynamically on your behalf in response to actual traffic patterns. This enables a table or a global secondary index to increase its provisioned read and write capacity to handle sudden increases in traffic without throttling. When the workload decreases, AWS Auto Scaling decreases the throughput so that you don't pay for unused provisioned capacity.

Amazon Aurora Auto Scaling

Amazon Aurora automatic scaling dynamically adjusts the number of Aurora Replicas provisioned for an Aurora DB cluster. Aurora automatic scaling is available for both Aurora MySQL and Aurora PostgreSQL. Aurora automatic scaling enables your Aurora DB cluster to handle sudden increases in connectivity or workload. When the connectivity or workload decreases, Aurora automatic scaling removes unnecessary Aurora Replicas so that you don't pay for unused provisioned DB instances.

Amazon Aurora Serverless is an on-demand, automatic scaling configuration for the MySQL-compatible edition of Amazon Aurora. An Aurora Serverless DB cluster automatically starts up, shuts down, and scales capacity up or down based on your application's needs. Aurora Serverless provides a relatively simple, cost-effective option for infrequent, intermittent, or unpredictable workloads.

Accessing AWS Auto Scaling

There are several ways to get started with AWS Auto Scaling. You can set up AWS Auto Scaling through the AWS Management Console, with the AWS CLI, or with AWS SDKs.

You can access the features of AWS Auto Scaling using the AWS CLI, which provides commands to use with Amazon EC2 and Amazon CloudWatch and Elastic Load Balancing.

To scale a resource other than Amazon EC2, you can use the Application Auto Scaling API, which allows you to define scaling policies to scale your AWS resources automatically or schedule one-time or recurring scaling actions.

Using Containers

Containers provide a standard way to package your application's code, configurations, and dependencies into a single object. Containers share an operating system installed on the server and run as resource-isolated processes, ensuring quick, reliable, and consistent deployments, regardless of environment.

Containers provide process isolation that lets you granularly set CPU and memory utilization for better use of compute resources.

Containerize Everything

Containers are a powerful way for developers to package and deploy their applications. They are lightweight and provide a consistent, portable software environment for applications to run and scale effortlessly anywhere.

Use Amazon Elastic Container Service (Amazon ECS) to build all types of containerized applications easily, from long-running applications and microservices to batch jobs and machine learning applications. You can migrate legacy Linux or Windows applications from on-premises to the AWS Cloud and run them as containerized applications using Amazon ECS.

Amazon ECS enables you to use containers as building blocks for your applications by eliminating the need for you to install, operate, and scale your own cluster management infrastructure. You can schedule long-running applications, services, and batch processes using Docker containers. Amazon ECS maintains application availability and allows you to scale your containers up or down to meet your application's capacity requirements. Amazon ECS is integrated with familiar features like Elastic Load Balancing, EBS volumes, virtual private cloud (VPC), and AWS Identity and Access Management (IAM). Use APIs to integrate and use your own schedulers or connect Amazon ECS into your existing software delivery process.

Containers without Servers

AWS Fargate technology is available with Amazon ECS. With Fargate, you no longer have to select Amazon EC2 instance types, provision and scale clusters, or patch and update each server. You do not have to worry about task placement strategies, such as binpacking or host spread, and tasks are automatically balanced across Availability Zones. Fargate manages the availability of containers for you. You define your application's requirements,

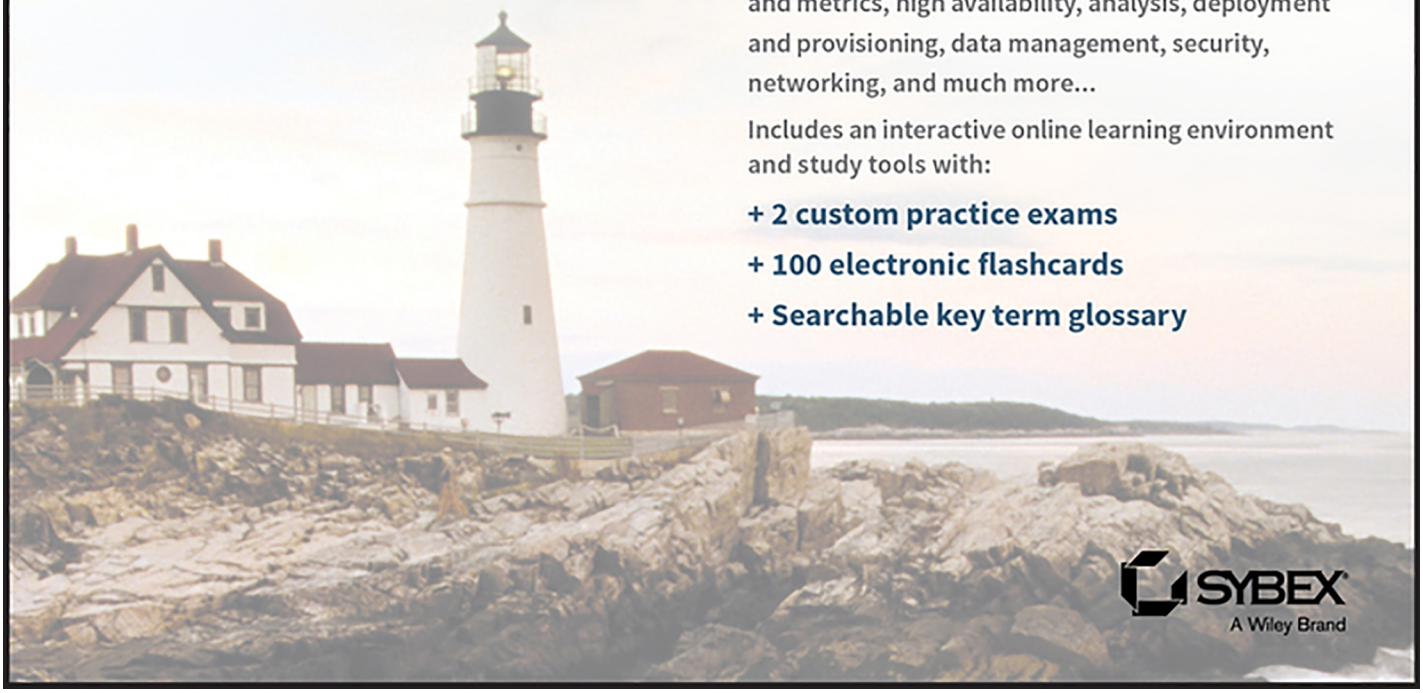
amazon web services

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:

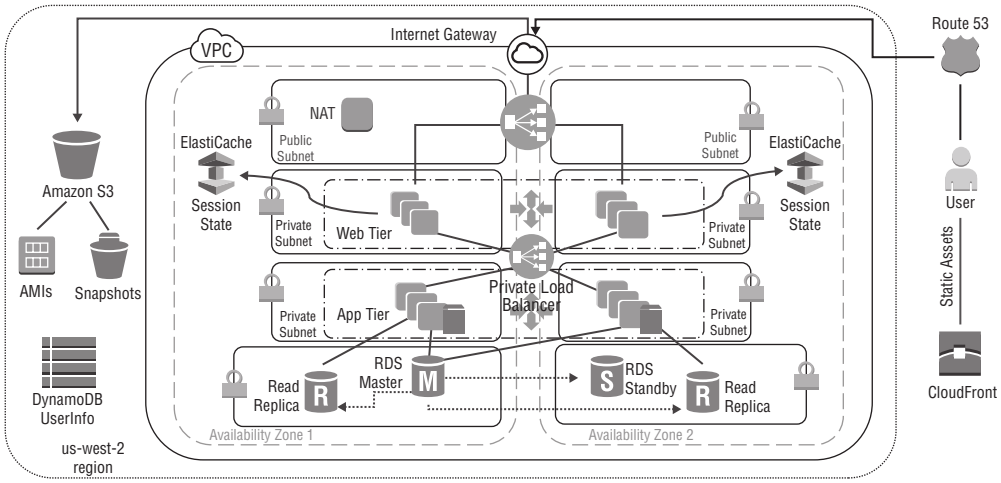
+ 2 custom practice exams

+ 100 electronic flashcards

+ Searchable key term glossary

SYBEX
A Wiley Brand

The Solution



As we examine the pieces of the solution, we start by breaking down the components of the architecture. Then we focus on how systems operators interact with the individual pieces and begin thinking about how those pieces fit into the certification exam.

Environment

Architectures live inside *AWS Regions*; in this scenario, in us-west-2 (Oregon, United States). Regions are made up of multiple *Availability Zones*, which provide the foundation for highly available architectures. Although this is a systems operation exam, it is critical to understand the nature of AWS Regions and Availability Zones.

Each AWS Region is a separate geographic area. Each AWS Region has multiple, isolated locations known as *Availability Zones*. *AWS Regions* and *Availability Zones* are discussed in Chapter 5, "Networking."

Networking

Networking components start inside the AWS Region with Amazon Virtual Private Cloud (Amazon VPC). *Amazon VPC* is a private network in the AWS Region that isolates all traffic from the millions of other applications running in AWS. A deep dive into Amazon VPC (and the rest of its components) is found in Chapter 5.

Amazon VPC is divided into *subnets*; all assets running in your Amazon VPC are assigned to a subnet. Unlike on-premises subnetting decisions that can affect latency between servers, Amazon VPC subnets only affect access. Access between subnets is

controlled through *network Access Control Lists (nACLs)*, and access in and out of Amazon VPC is controlled through attached gateways. In this scenario, the only gateway is the *Internet Gateway (IGW)*, and it allows traffic to and from external (public IP) sources.

By granting route table access to the gateway only to specific subnets, ingress and egress can be tightly controlled. In this scenario, public subnets indicate IGW access. Without IGW access, the subnets become private; that is, they are accessible only to private IP networks.

To learn about the other gateways that could be leveraged to create hybrid or other private architectures, refer to Chapter 5.

Security groups are often part of the networking discussion. They provide stateful firewalls that operate at the Hypervisor levels for all individual*Amazon Elastic Compute Cloud (Amazon EC2)* instances and other Amazon VPC objects. In this scenario, we potentially have seven different security groups:

Public Elastic Load Balancing The only security group that allows full public access

Web Tier Amazon EC2 This accepts traffic only from public Elastic Load Balancing.

Private Elastic Load Balancing This accepts traffic only from Web Tier Amazon EC2.

Application Tier Amazon EC2 This accepts traffic only from private Elastic Load Balancing.

Amazon ElastiCache This accepts traffic only from Application Tier Amazon EC2.

Amazon Relational Database Service (Amazon RDS) This accepts traffic only from Application Tier Amazon EC2.

Network Address Translation (NAT) This is used only for internally initiated outbound traffic.

By specifically stacking security groups in this manner, you can provide layers of network security that surround the database portion of the three-tier design.

Compute

In this scenario, you use traditional compute methods, such as Linux servers running on Amazon EC2. Amazon EC2 comes in many sizes (how many CPUs, how much memory, how much network capacity, and so on), known as *instances*. Based on the Amazon Machine Image (AMI), each Amazon EC2 instance can run a wide range of Linux- or Windows-based operating systems as well as preinstalled software packages. Amazon EC2 instances also support runtime configuration as required.

The requirements for the scenario include scalable solutions. AWS provides Auto Scaling as an engine that can take predefined launch configurations and dynamically add or remove instances from the web or the Application Tier based on metrics.

as that right is explicitly reserved for the AWS account that created the volume. An Amazon EBS snapshot is a block-level view of an entire Amazon EBS volume. Note that data that is not visible through the file system on the volume, such as files that have been deleted, may be present in the Amazon EBS snapshot. If you want to create shared snapshots, you should do so carefully. If a volume has held sensitive data or has had files deleted from it, you should create a new Amazon EBS volume to share. The data to be contained in the shared snapshot should be copied to the new volume and the snapshot created from the new volume.

Amazon EBS volumes are presented to you as raw unformatted *block devices*, which have been wiped prior to being made available for use. Wiping occurs immediately before reuse so that you can be assured that the wipe process completed. If you have procedures requiring that all data be wiped via a specific method, you have the ability to do so on Amazon EBS. You should conduct a specialized wipe procedure prior to deleting the volume for compliance with your established requirements.

Encryption of sensitive data is generally a good security practice, and AWS provides the ability to encrypt Amazon EBS volumes and their snapshots with AES-256. The encryption occurs on the servers that host the Amazon EC2 instances, providing encryption of data as it moves between Amazon EC2 instances and Amazon EBS storage. In order to be able to do this efficiently and with low latency, the Amazon EBS encryption feature is only available on Amazon EC2's more powerful instance types.

Networking

AWS provides a range of networking services that enable you to create a logically isolated network that you define, establish a private network connection to the AWS Cloud, use a highly available and scalable Domain Name System (DNS) service, and deliver content to your end users with low latency at high data transfer speeds with a content delivery web service.

Elastic Load Balancing Security

Elastic Load Balancing is used to manage traffic on a fleet of Amazon EC2 instances, distributing traffic to instances across all Availability Zones within a region. Elastic Load Balancing has all of the advantages of an on-premises load balancer, plus several security benefits:

- Takes over the encryption and decryption work from the Amazon EC2 instances and manages it centrally on the load balancer
- Offers clients a single point of contact and can also serve as the first line of defense against attacks on the customer's network
- When used in an Amazon VPC, supports creation and management of security groups associated with Elastic Load Balancing to provide additional networking and security options
- Supports end-to-end traffic encryption using TLS (previously SSL) on those networks that use HTTPS connections. When TLS is used, the TLS server certificate used to terminate client connections can be managed centrally on the load balancer, instead of on every individual instance.

HTTPS/TLS uses a long-term secret key to generate a short-term session key to be used between the server and the browser to create the encrypted message. Elastic Load Balancing configures your load balancer with a predefined cipher set that is used for TLS negotiation when a connection is established between a client and your load balancer. The predefined cipher set provides compatibility with a broad range of clients and uses strong cryptographic algorithms. However, some customers may have requirements for allowing only specific ciphers and protocols (for example, PCI DSS, Sarbanes-Oxley Act [SOX]) from clients to ensure that standards are met. In these cases, Elastic Load Balancing provides options for selecting different configurations for TLS protocols and ciphers. You can choose to enable or disable the ciphers depending on your specific requirements.

To help ensure the use of newer and stronger cipher suites when establishing a secure connection, you can configure the load balancer to have the final say in the cipher suite selection during the client-server negotiation. When the server order preference option is selected, the load balancer will select a cipher suite based on the server's prioritization of cipher suites rather than the client's. This gives you more control over the level of security that clients use to connect to your load balancer.

For even greater communication privacy, Elastic Load Balancing allows the use of perfect forward secrecy, which uses session keys that are ephemeral and not stored anywhere. This prevents the decoding of captured data, even if the secret long-term key itself is compromised.

Elastic Load Balancing allows you to identify the originating IP address of a client connecting to your servers, whether you're using HTTPS or TCP load balancing. Typically, client connection information, such as IP address and port, is lost when requests are proxied through a load balancer. This is because the load balancer sends requests to the server on behalf of the client, making your load balancer appear as though it is the requesting client. Having the originating client IP address is useful if you need more information about visitors to your applications in order to gather connection statistics, analyze traffic logs, or manage whitelists of IP addresses.

Elastic Load Balancing access logs contain information about each HTTP and TCP request processed by your load balancer. This includes the IP address and port of the requesting client, the back-end IP address of the instance that processed the request, the size of the request and response, and the actual request line from the client (for example, `GET http://www.example.com: 80/HTTP/1.1`). All requests sent to the load balancer are logged, including requests that never make it to back-end instances (more on Elastic Load Blancing can be found in Chapter 5).

Amazon Virtual Private Cloud (Amazon VPC) Security

Normally, each Amazon EC2 instance you launch is randomly assigned a public IP address in the Amazon EC2 address space. *Amazon VPC* enables you to create an isolated portion of the AWS Cloud and launch Amazon EC2 instances that have private (RFC 1918) addresses in the range of your choice (for example, 10.0.0.0/16). You can define subnets in your Amazon VPC, grouping similar kinds of instances based on IP address range, and then set up routing and security to control the flow of traffic in and out of the instances and subnets.

groups to control access on a specific interface. With administrative access, you again can control access with placement of Amazon EC2 instances in an Amazon VPC, the use of security groups, and the use of public/private key pairs both to encrypt traffic and control access.

Controlling Administrative Access

AWS provides a number of tools to manage the security of your instances. These tools include the following:

- IAM
- AWS Trusted Advisor
- AWS Service Catalog
- Amazon Inspector

IAM allows you to create policies that control access to APIs and apply those policies to users, groups, and roles.

AWS Trusted Advisor, which comes as part of AWS Support, looks at things like open ports on security groups and level of access allowed to Amazon EC2 instances, and it makes recommendations to improve the security of your AWS infrastructure.

AWS Service Catalog allows IT administrators to create, manage, and distribute portfolios of approved products to end users who can then access the products they need in a personalized portal. AWS Service Catalog allows organizations to manage commonly deployed IT services centrally, and it helps organizations achieve consistent governance and meet compliance requirements while enabling users to deploy quickly only the approved IT services they need within the constraints the organization sets.

Amazon Inspector is a security vulnerability assessment service that helps improve the security and compliance of your AWS resources. Amazon Inspector automatically assesses resources for vulnerabilities or deviations from best practices and then produces a detailed list of security findings prioritized by level of severity.

AWS also has a number of developer documents, whitepapers, articles, and tutorials to help you in securing your environment. These are available on the Cloud Security Resources page of the AWS website.

Amazon EC2 Container Service (Amazon ECS)

Amazon ECS is a highly scalable container management service. Amazon ECS makes it easy to run, stop, and manage Docker containers on Amazon EC2 instances.

With Amazon ECS, you can launch and place containers across your cluster using API calls. You schedule the placement of containers based on your resource needs, isolation

policies, and availability requirements. You can use Amazon ECS both for cluster management and for scheduling deployments of containers onto hosts.

Amazon ECS eliminates the need for you to operate your own cluster management and configuration management systems. Amazon ECS can be used to manage and scale both batch and Extract, Transform, Load (ETL) workloads. It can also be used to build application architectures based on the microservices model.

Amazon ECS is a regionally-based service that can be used to run application containers in a highly available manner across all Availability Zones within an AWS Region.

Implementation

To implement Amazon ECS, you need to install an Amazon ECS agent on your Amazon EC2 instances. If you use Amazon ECS-optimized AMIs, that agent is already installed. Additionally, the container instance needs to have an IAM role that authenticates to your account and will need external network access to communicate with the Amazon ECS service endpoint.

Clusters are the logical grouping of container instances that you can place tasks on. Clusters are region specific and can contain multiple, different instance types (though an instance can only be a member of a single cluster).

Amazon ECS obtains a Docker image for repository. This repository can be on AWS or on other infrastructure.

Deploying a Container

To deploy a container, you need to do the following:

1. **Define a task**. This is where you assign the name, provide the image name (important for locating the correct image), and decide on the amount of memory and CPU needed.
2. **Define the service**. In this step, you decide how many instances of the task you want to run in the cluster and any Elastic Load Balancing load balancers that you want to register the instances with.
3. **Create the Amazon ECS cluster**. This is where the cluster is created and also where you specify the number of instances to run. The cluster can run across multiple Availability Zones.
4. **Create the stack**. A stack of instances is created based on the configuration information provided. You can monitor the creation of the stack in the AWS Management Console. Creation of the stack is usually completed in less than five minutes.

Management

Amazon ECS can be configured using the AWS Management Console, the AWS CLI, or the Amazon ECS CLI.

Monitoring Amazon ECS

The primary tool used for monitoring your Amazon ECS clusters is Amazon CloudWatch. Amazon CloudWatch collects Amazon ECS metric data in one-minute periods and sends them to Amazon CloudWatch. Amazon CloudWatch stores these metrics for a period of two weeks. You can monitor the CPU and memory reservation and utilization across your cluster as a whole and the CPU and memory utilization on the services in your cluster. You can use Amazon CloudWatch to trigger alarms, set notifications, and create dashboards to monitor the services.

Once it's set up, you can view Amazon CloudWatch metrics in both the Amazon ECS console and the Amazon CloudWatch console. The Amazon ECS console provides a maximum 24-hour view while the Amazon CloudWatch console provides a fine-grained and customizable view of running services.

The other tool available is AWS CloudTrail, which will log all Amazon ECS API calls.

AWS Trusted Advisor is another source for monitoring all of your AWS resources, including Amazon ECS, to improve performance, reliability, and security.

There is no additional cost for using Amazon ECS. The only charges are for the Amazon EC2 instances or AWS Lambda requests and compute time.

Security

With Amazon ECS, you need to do the following:

1. Control who can create task definitions.
2. Control who can deploy clusters.
3. Control who can access the Amazon EC2 instances.

IAM is the tool used for the first two necessities. For controlling access to Amazon EC2 instances, the tools described in the Amazon EC2 section still apply. You can use IAM roles, security groups, and (because these Amazon EC2 instances are located in an Amazon VPC) network Access Control Lists (ACLs) and route tables to control access to the Amazon EC2 instances.

AWS Elastic Beanstalk

AWS Elastic Beanstalk enables you to deploy and manage applications in the AWS Cloud without worrying about the infrastructure that runs those applications. 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.

Languages Supported in AWS Elastic Beanstalk

AWS Elastic Beanstalk supports applications developed in the following languages:

- Java
- PHP
- .NET
- Node.js
- Python
- Ruby

Services that AWS Elastic Beanstalk Deploys

AWS Elastic Beanstalk automates the deployment of Amazon VPC, multiple compute instances (across multiple Availability Zones), security groups (both for instances and load balancers), Auto Scaling, Amazon S3 buckets, Amazon CloudWatch alarms, and domain names. Even though AWS Elastic Beanstalk automates the deployment of your environment, you can make modifications as you see fit.

Implementation

There are a number of tools that you can use to create and manage the AWS Elastic Beanstalk environment. These tools are as follows:

- AWS Management Console
- AWS Elastic Beanstalk CLI
- AWS SDK for Java
- AWS Toolkit for Eclipse
- AWS SDK for .NET
- AWS Toolkit for Visual Studio
- AWS SDK for Node.js
- AWS SDK for PHP (requires PHP 5.2 or later)
- Boto (AWS SDK for Python)
- AWS SDK for Ruby

You deploy a new application on AWS Elastic Beanstalk by uploading a source bundle. This source bundle can be a ZIP or a WAR file (you can include multiple WAR files inside of your ZIP file). This source bundle cannot be larger than 512 MB.

Management

AWS Elastic Beanstalk is by definition highly available. It creates multiple compute instances, locates those compute instances in two Availability Zones, and creates a load

balancer (a Classic Elastic Load Balancing load balancer) to spread the compute load across these multiple instances.

You can configure Auto Scaling to establish a minimum number of instances, a maximum number of instances, and what parameters would trigger either an increase or decrease in the number of instances. These triggers can either be time-based or event-based.

When you go into the AWS Elastic Beanstalk console, it displays all of the environments running in that region, sorted by application. You can further drill down by application name to see the environments, application versions, and saved configurations. There are limits on the number of applications, application versions, and environments that you can have.

As with Amazon EC2, you are responsible for patching and updating both the guest operating system and your application with AWS Elastic Beanstalk.

Making Changes in Your AWS Elastic Beanstalk Environment

It is recommended to use the AWS Elastic Beanstalk console and CLI when making changes to your AWS Elastic Beanstalk environments and configuration. Although it is possible to modify your AWS Elastic Beanstalk environment using other service consoles, CLI commands, or AWS SDKs, those changes will not be reflected in the AWS Elastic Beanstalk console and you will not be able to monitor, change, or save your environment. It will also cause issues when you attempt to terminate your environment without using the AWS Elastic Beanstalk console.

Monitoring Your AWS Elastic Beanstalk Environment

You can monitor the overall health of your environment using either the basic health reporting or enhanced health reporting and monitoring systems. Basic health reporting gives an overall status (via color key) of the environment. It does not publish any metrics to Amazon CloudWatch. Enhanced health reporting and monitoring not only provides status via color key, but it also provides a descriptor that helps indicate the severity and cause of the problem. You can also configure enhanced health reporting and monitoring to publish metrics to Amazon CloudWatch as custom metrics.

You can retrieve log files from the individual instances or have those log files uploaded to an Amazon S3 bucket for more permanent storage.

Security

When you create an environment with AWS Elastic Beanstalk, you are prompted to create two IAM roles:

Service role This is used by AWS Elastic Beanstalk to access other AWS Cloud services.

Instance profile This is applied to the instances created in your environment, and it allows them to upload logs to Amazon S3 and perform other tasks.

In addition to these two roles, you can create polices in IAM and attach them to users, groups, and roles. Just as in Amazon EC2, in AWS Elastic Beanstalk you need to create an Amazon EC2 key pair to access the guest operating system within your compute instances. Access would be via either Port 22 (for SSH) or Port 3389 (for RDP).

AWS Elastic Beanstalk allows web developers to deploy highly scalable and highly available web infrastructure without having to know or understand services like Elastic Load Balancing, Auto Scaling groups, or Availability Zones.

AWS Lambda

AWS Lambda is a serverless compute service. It runs your code in response to events. AWS Lambda automatically manages the underlying compute resources with no server configuration from you. There is no need to provision or manage servers.

The execution of code (called an AWS Lambda function) can be triggered by events internal to your AWS infrastructure or external to it.

AWS Lambda is a regionally-based service running across multiple Availability Zones. This makes it highly available and highly scalable. Because the code is stateless, it is executed almost automatically without deployment or configuration delays.

Implementation

Implementation for AWS Lambda involves the following steps:

1. Configure an AWS Lambda function.
2. Create a deployment package.
3. Upload the deployment package.
4. Create an invocation type.

The languages available to write your code are Node.js, Java, Python, and C#. The language that you choose also controls what tools are available for authoring your code. These tools can include the AWS Lambda console, Visual Studio, .NET Core, Eclipse, or your own authoring environment. You use these languages to create a deployment package.

Once a deployment package has been created, you then upload it to create an AWS Lambda function. The tools you can use to do so are the AWS Lambda console, CLI, and AWS SDKs.

The code that you have written is part of what is called an *AWS Lambda function*. The other elements of an AWS Lambda function are as follows:

- Memory
- Maximum execution time
- IAM role
- Handler name

The amount of memory that you specify controls the CPU power. The default amount of memory allocated per AWS Lambda function is 512 MB, but this can range from 128 MB to 1,536 MB in 64 MB increments.

Load Balancing

Elastic Load Balancing distributes incoming application traffic across multiple Amazon EC2 instances, in multiple Availability Zones within a single AWS Region. Elastic Load Balancing supports two types of load balancers:

- Application Load Balancers
- Classic Load Balancers

The Elastic Load Balancing load balancer serves as a single target, which increases the availability of your application. You can add and remove instances from your load balancer as your needs change without disrupting the overall flow of requests to your application. Elastic Load Balancing, working in conjunction with Auto Scaling, can scale your load balancer as traffic to your application changes over time. Refer to Chapter 10, "High Availability," for details on how Auto Scaling works.

You can configure health checks and send requests only to the healthy instances. You can also offload the work of encryption and decryption to your load balancer. See Table 5.6 for differences between the Classic Load Balancer and the Application Load Balancer.

TABLE 5.6 Classic Load Balancer and Application Load Balancer

Feature	Classic Load Balancer	Application Load Balancer
Protocols	HTTP, HTTPS, TCP, SSL	HTTP, HTTPS
Platforms	Amazon EC2-Classic, Amazon EC2-VPC	Amazon EC2-VPC
Sticky sessions (cookies)	✔	Load balancer generated
Idle connection timeout	✔	✔
Connection draining	✔	✔
Cross-zone load balancing	Can be enabled	Always enabled
Health checks	✔	Improved
Amazon CloudWatch metrics	✔	Improved
Access logs	✔	Improved
Host-based routing		✔
Path-based routing		✔

TABLE 5.6 Classic Load Balancer and Application Load Balancer *(continued)*

Feature	Classic Load Balancer	Application Load Balancer
Route to multiple ports on a single instance		✔
HTTP/2 support		✔
WebSocket support		✔
Load balancer deletion protection		✔

Load Balancing Implementation

Because the Classic Load Balancer and the Application Load Balancer distribute traffic in different ways, implementation is different. Let's start with the Classic Load Balancer. Both Load Balancers will be examined in this section of the chapter.

Classic Load Balancer

With a *Classic Load Balancer*, there are a number of parameters that you need to configure. These include the following:

- Choose VPC
- Choose subnets
- Define protocols and ports
- Choose Internet facing or internal
- Determine security groups
- Configure health check
- Assign Amazon EC2 instances

If you have multiple VPCs, you determine which VPC will contain the Application Load Balancer. Load balancers cannot be shared among VPCs. From there, you are given a list of Availability Zones in which you have created subnets. In order for your application to be considered highly available, you must specify at least two Availability Zones. If you need additional Availability Zones, you need to create subnets in those Availability Zones.

High Availability

You can distribute incoming traffic across your Amazon EC2 instances in a single Availability Zone or multiple Availability Zones. The Classic Load Balancer automatically scales its request handling capacity in response to incoming application traffic.

Health Checks

The Classic Load Balancer can detect the health of Amazon EC2 instances. When it detects unhealthy Amazon EC2 instances, it no longer routes traffic to those instances and spreads the load across the remaining healthy instances.

Security Features

When using Amazon Virtual Private Cloud (Amazon VPC), you can create and manage security groups associated with Classic Load Balancers to provide additional networking and security options. You can also create a Classic Load Balancer without public IP addresses to serve as an internal (non-Internet-facing) load balancer.

SSL Offloading

Classic Load Balancers support SSL termination, including offloading SSL decryption from application instances, centralized management of SSL certificates, and encryption to back-end instances with optional public key authentication.

Flexible cipher support allows you to control the ciphers and protocols the load balancer presents to clients.

Sticky Sessions

Classic Load Balancers support the ability to stick user sessions to specific EC2 instances using cookies. Traffic will be routed to the same instances as the user continues to access your application.

IPv6 Support

Classic Load Balancers support the use of Internet Protocol versions 4 and 6 (IPv4 and IPv6). IPv6 support is currently unavailable for use in VPC.

Layer 4 or Layer 7 Load Balancing

You can load balance HTTP/HTTPS applications and use layer 7-specific features, such as X-Forwarded and sticky sessions. You can also use strict layer 4 load balancing for applications that rely purely on the TCP protocol.

Operational Monitoring

Classic Load Balancer metrics, such as request count and request latency, are reported by Amazon CloudWatch.

Logging

Use the Access Logs feature to record all requests sent to your load balancer and store the logs in Amazon S3 for later analysis. The logs are useful for diagnosing application failures and analyzing web traffic.

You can use AWS CloudTrail to record classic load balancer API calls for your account and deliver log files. The API call history enables you to perform security analysis, resource change tracking, and compliance auditing.

You can assign the Classic Load Balancer to any Amazon EC2 instance that is in your VPC, including Amazon EC2 instances not in the Availability Zones that you specified. If you do so, the Classic Load Balancer will not route traffic to those Amazon EC2 instances.

Application Load Balancer

With an *Application Load Balancer*, there are a number of parameters that you need to configure. These include the following:

- Name of your load balancer
- Security groups
- The VPC
- Availability Zones to be used
- Internet-facing or internal
- IP addressing scheme to be used
- Configure one or more listeners
- Configure listener rules
- Define target group

Table 5.6 demonstrates the features of the Application Load Balancer. In order to become a proficient Systems Operator on AWS, you need to know what these features do. The features of the Application Load Balancer are defined in the following sections.

Content-Based Routing

If your application is composed of several individual services, an Application Load Balancer can route a request to a service based on the content of the request.

Host-Based Routing

You can route a client request based on the Host field of the HTTP header, allowing you to route to multiple domains from the same load balancer.

Path-Based Routing

You can route a client request based on the URL path of the HTTP header.

Containerized Application Support

You can now configure an Application Load Balancer to load balance containers across multiple ports on a single Amazon EC2 instance. Amazon EC2 Container Service (Amazon ECS)

allows you to specify a dynamic port in the ECS task definition, giving the container an unused port when it is scheduled on the EC2 instance. The ECS scheduler automatically adds the task to the ELB using this port.

HTTP/2 Support

HTTP/2 is a new version of the Hypertext Transfer Protocol (HTTP) that uses a single, multiplexed connection to allow multiple requests to be sent on the same connection. It also compresses header data before sending it out in binary format, and it supports TLS connections to clients.

WebSockets Support

WebSockets allows a server to exchange real-time messages with end users without the end users having to request (or poll) the server for an update. The WebSockets protocol provides bi-directional communication channels between a client and a server over a long-running TCP connection.

Native IPv6 Support

Application Load Balancers support native Internet Protocol version 6 (IPv6) in a VPC. This will allow clients to connect to the Application Load Balancer via IPv4 or IPv6.

Sticky Sessions

Sticky sessions are a mechanism to route requests from the same client to the same target. The Application Load Balancer supports sticky sessions using load balancer generated cookies. If you enable sticky sessions, the same target receives the request and can use the cookie to recover the session context. Stickiness is defined at a target group level.

Health Checks

An Application Load Balancer routes traffic only to healthy targets. With an Application Load Balancer, you get improved insight into the health of your applications in two ways:

1. Health check improvements that allow you to configure detailed error codes from 200-399. The health checks allow you to monitor the health of each of your services behind the load balancer.
2. New metrics that give insight into traffic for each of the services running on an Amazon EC2 instance.

High Availability

An Application Load Balancer requires you to specify more than one Availability Zone. You can distribute incoming traffic across your targets in multiple Availability Zones. An Application Load Balancer automatically scales its request-handling capacity in response to incoming application traffic.

Layer-7 Load Balancing

You can load balance HTTP/HTTPS applications and use layer 7-specific features, such as X-Forwarded-For (XFF) headers.

HTTPS Support

An Application Load Balancer supports HTTPS termination between the clients and the load balancer. Application Load Balancers also offer management of SSL certificates through AWS Identity and Access Management (IAM) and AWS Certificate Manager for predefined security policies.

Operational Monitoring

Amazon CloudWatch reports Application Load Balancer metrics, such as request counts, error counts, error types, and request latency.

Logging

You can use the Access Logs feature to record all requests sent to your load balancer and store the logs in Amazon S3 for later analysis. The logs are compressed and have a `.gzip` file extension. The compressed logs save both storage space and transfer bandwidth, and they are useful for diagnosing application failures and analyzing web traffic.

You can also use AWS CloudTrail to record Application Load Balancer API calls for your account and deliver log files. The API call history enables you to perform security analysis, resource change tracking, and compliance auditing.

Delete Protection

You can enable deletion protection on an Application Load Balancer to prevent it from being accidentally deleted.

Request Tracing

The Application Load Balancer injects a new custom identifier "X-Amzn-Trace-Id" HTTP header on all requests coming into the load balancer. Request tracing allows you to track a request by its unique ID as the request makes its way across various services that make up your websites and distributed applications. You can use the unique trace identifier to uncover any performance or timing issues in your application stack at the granularity of an individual request.

AWS Web Application Firewall

You can now use AWS WAF to protect your web applications on your Application Load Balancers. AWS WAF is a web application firewall that helps protect your web applications from common web exploits that could affect application availability, compromise security, or consume excessive resources.

Load Balancing Management

Elastic Load Balancing is a highly available and scalable service.

You can create, access, and manage your load balancers using any of the following interfaces:

AWS Management Console Provides a web interface that you can use to access Elastic Load Balancing.

AWS CLI Provides commands for a broad set of AWS Cloud services, including Elastic Load Balancing, and it is supported on Windows, Mac, and Linux.

AWS SDKs Provides language-specific APIs and takes care of many of the connection details, such as calculating signatures, handling request retries, and error handling.

Query API Provides low-level API actions that you call using HTTPS requests. Using the Query API is the most direct way to access Elastic Load Balancing, but it requires that your application handle low-level details such as generating the hash to sign the request and error handling.

There are three tools for managing both Classic Load Balancers and Application Load Balancers. These are Amazon CloudWatch, AWS CloudTrail, and access logs. In addition, Application Load Balancers has a feature that allows request tracing to track HTTP requests from clients to targets.

Amazon CloudWatch

Amazon CloudWatch provides a number of metrics available to track the performance of the Elastic Load Balancing load balancers. These metrics include number of health hosts, average latency, number of requests, and number of connections, among others. These metrics are updated on a 60-second interval.

AWS CloudTrail

AWS CloudTrail can capture the API calls used by the Elastic Load Balancing load balancers and deliver those logs to an Amazon S3 bucket that you specify. From there you can analyze the logs as needed. The information these logs will include are things like the time the API was invoked, what invoked the API, and the parameters of the request.

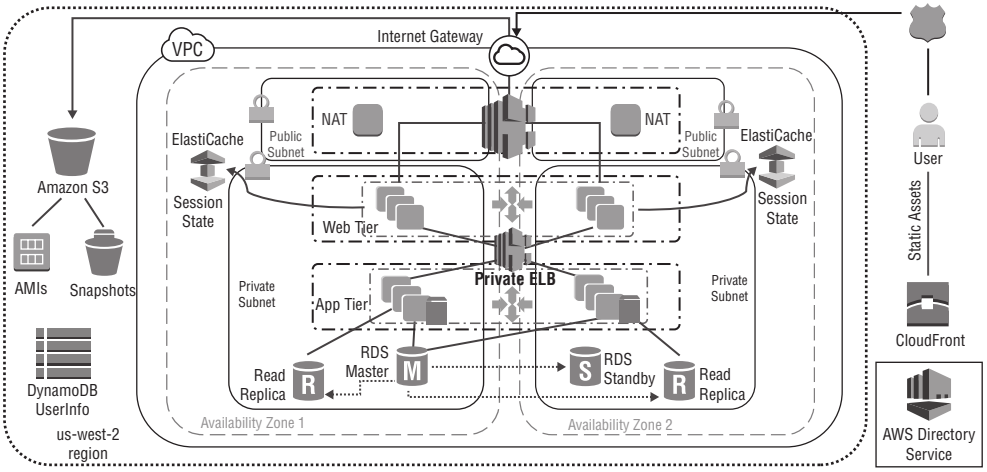
Access Logs

Access logs are an optional feature of Elastic Load Balancing. Access logs capture greater detailed information than AWS CloudTrail does, including such information as latencies, request paths, IP addresses of clients making the requests, and the instance processing the request, among other information.

Request Tracing

Request tracing is used to track HTTP requests from clients to targets or other services. When the load balancer receives a request from a client, it adds or updates the X-Amzn-Trace-Id header before sending the request to the target. Any services or applications between the load balancer and the target can also add or update this header. The contents of the header are logged in access logs.

FIGURE 10.5 Highly available three-tier architecture



Network Address Translation (NAT) Gateways

Network Address Translation (NAT) gateways were covered in Chapter 5. NAT servers allow traffic from private subnets to traverse the Internet or connect to other AWS Cloud services. Individual NAT servers can be a single point of failure. The NAT gateway is a managed device, and each NAT gateway is created in a specific Availability Zone and implemented with redundancy in that Availability Zone. To achieve optimum availability, use NAT gateways in each Availability Zone.

If you have resources in multiple Availability Zones, they share one NAT gateway. When the NAT gateway's Availability Zone is down, resources in the other Availability Zones will lose Internet access. To create an Availability Zone-independent architecture, create a NAT gateway in each Availability Zone and configure your routing to ensure that resources use the NAT gateway in the same Availability Zone.

Elastic Load Balancing

As discussed in Chapter 5, Elastic Load Balancing comes in two types: Application Load Balancer and Classic Load Balancer. *Elastic Load Balancing* allows you to decouple an application's web tier (or front end) from the application tier (or back end). In the event of a node failure, the Elastic Load Balancing load balancer will stop sending traffic to the affected Amazon EC2 instance, thus keeping the application available, although in a deprecated state. Self-healing can be achieved by using Auto Scaling. For a refresher on Elastic Load Balancing, refer to Chapter 5.

JON BONSO AND KENNETH SAMONTE

AWS CERTIFIED DEVOPS ENGINEER PROFESSIONAL

Tutorials Dojo
Study Guide and Cheat Sheets

AWS Elastic Load Balancing (ELB)

- Distributes incoming application or network traffic across multiple targets, such as **EC2 instances**, **containers (ECS)**, **Lambda functions,** and **IP addresses**, in multiple Availability Zones.
- When you create a load balancer, you must specify one public subnet from at least two Availability Zones. You can specify only one public subnet per Availability Zone.

General features

- Accepts incoming traffic from clients and routes requests to its registered targets.
- Monitors the health of its registered targets and routes traffic only to healthy targets.
- **Cross Zone Load Balancing** - when enabled, each load balancer node distributes traffic across the registered targets in all enabled AZs.

Three Types of Load Balancers

- **Application Load Balancer**
- **Network Load Balancer**
- **Classic load Balancer**
- **Slow Start Mode** gives targets time to warm up before the load balancer sends them a full share of requests.
- **Sticky sessions** route requests to the same target in a target group. You enable sticky sessions at the target group level. You can also set the duration for the stickiness of the load balancer-generated cookie, in seconds. Useful if you have stateful applications.
- **Health checks** verify the status of your targets. The statuses for a registered target are: