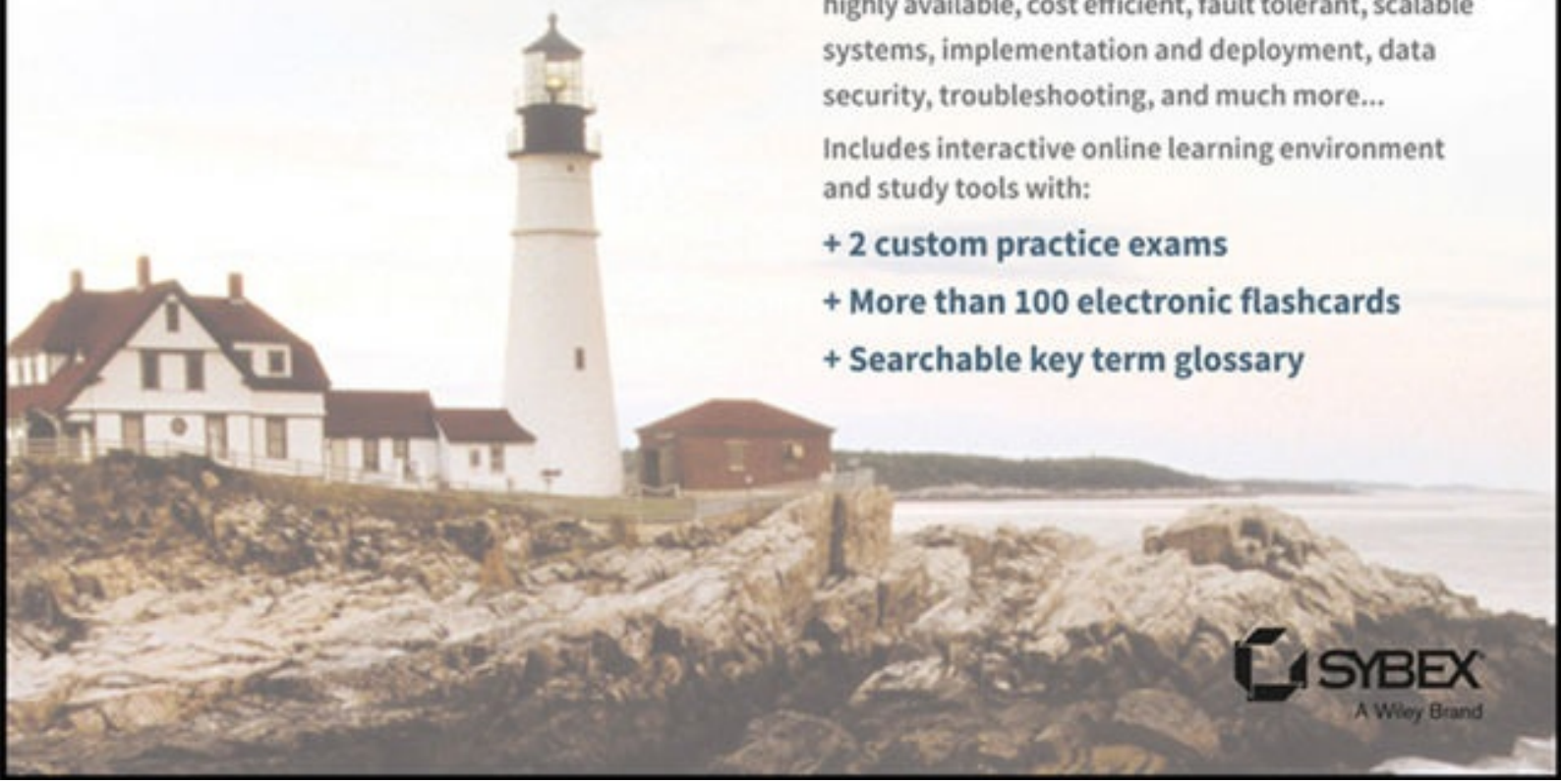
amazon web services

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.

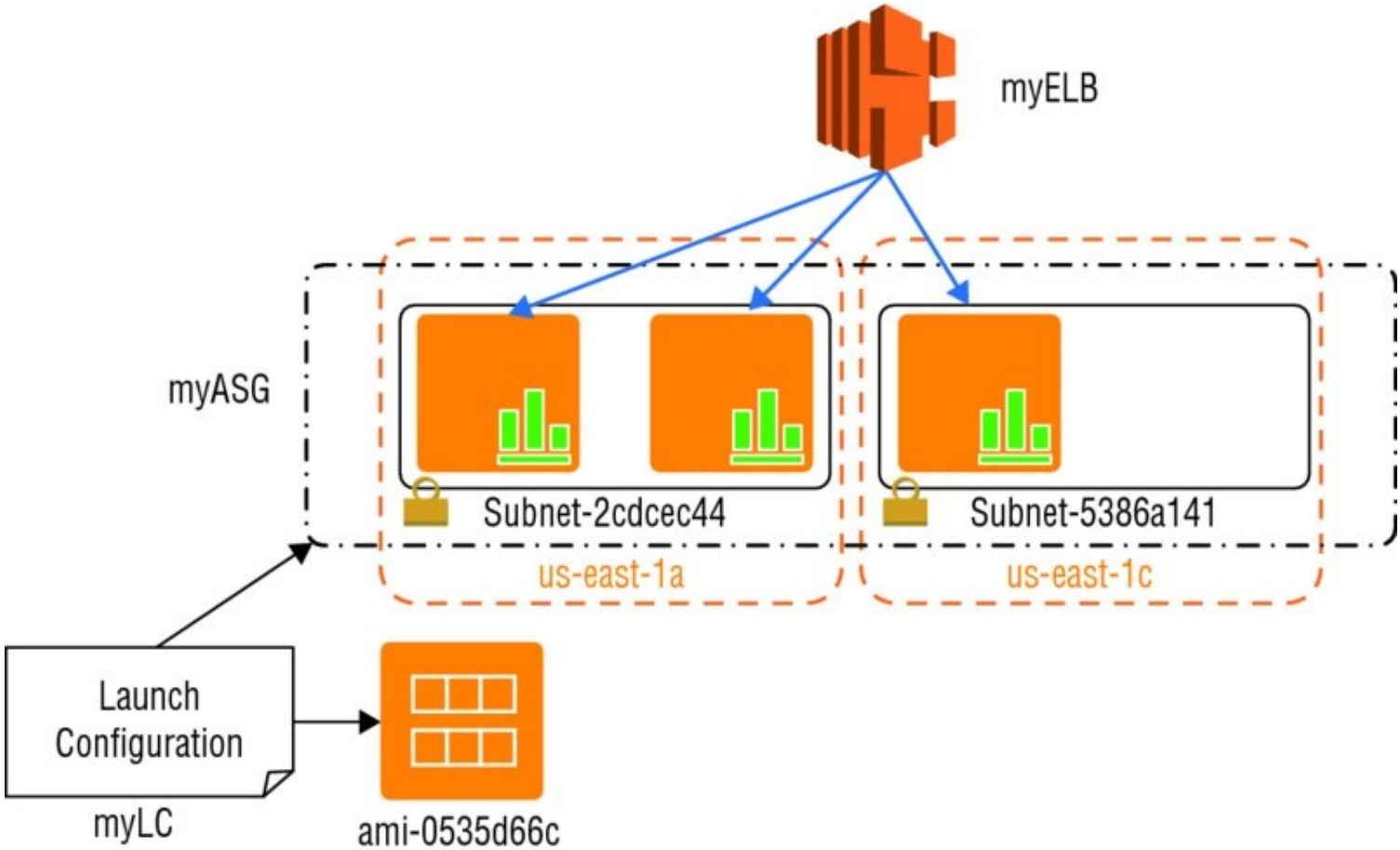


FIGURE 5.1 Auto Scaling group behind an Elastic Load Balancing load balancer

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).

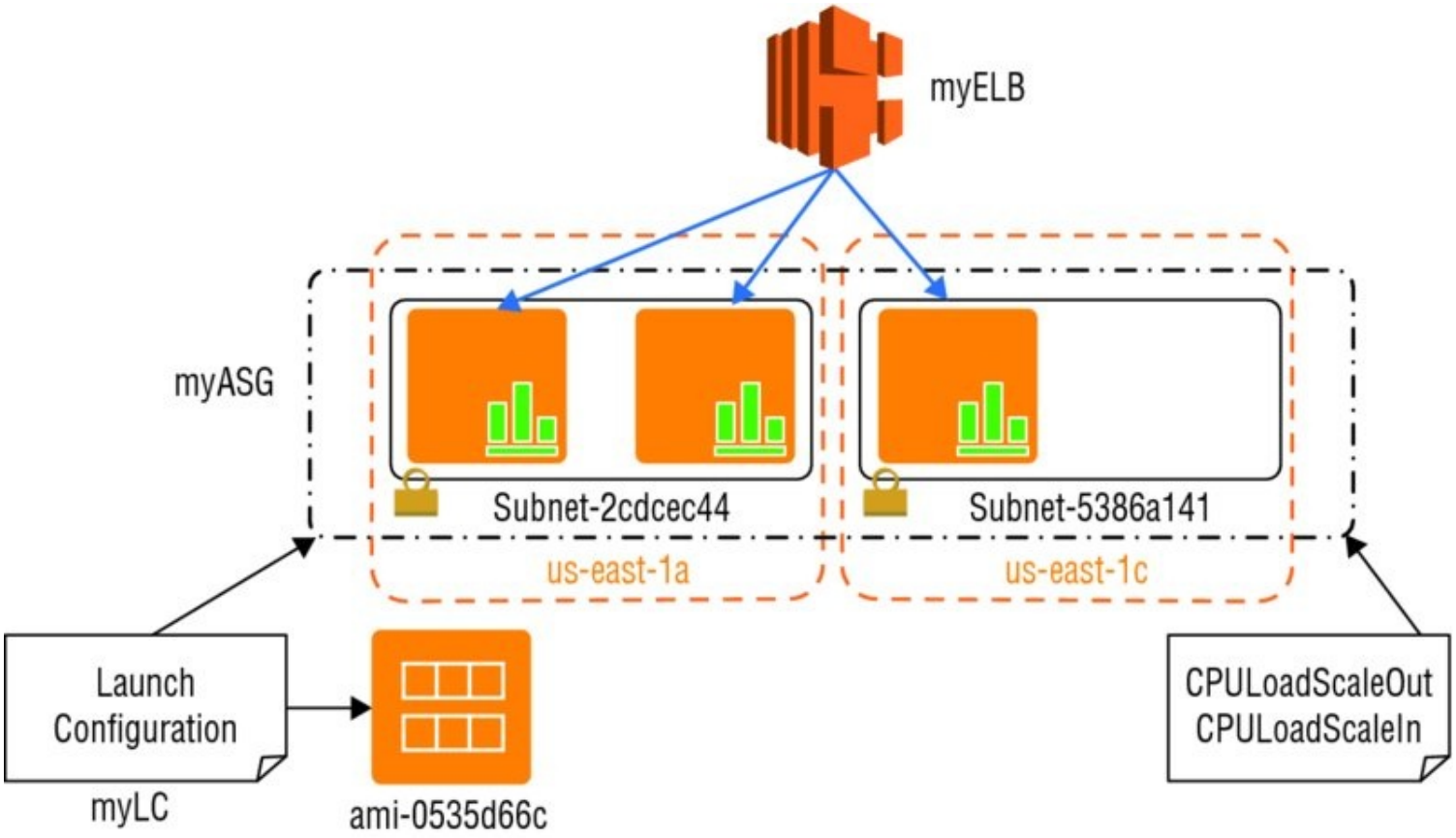


FIGURE 5.2 Auto Scaling group with policy

```
> 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`.

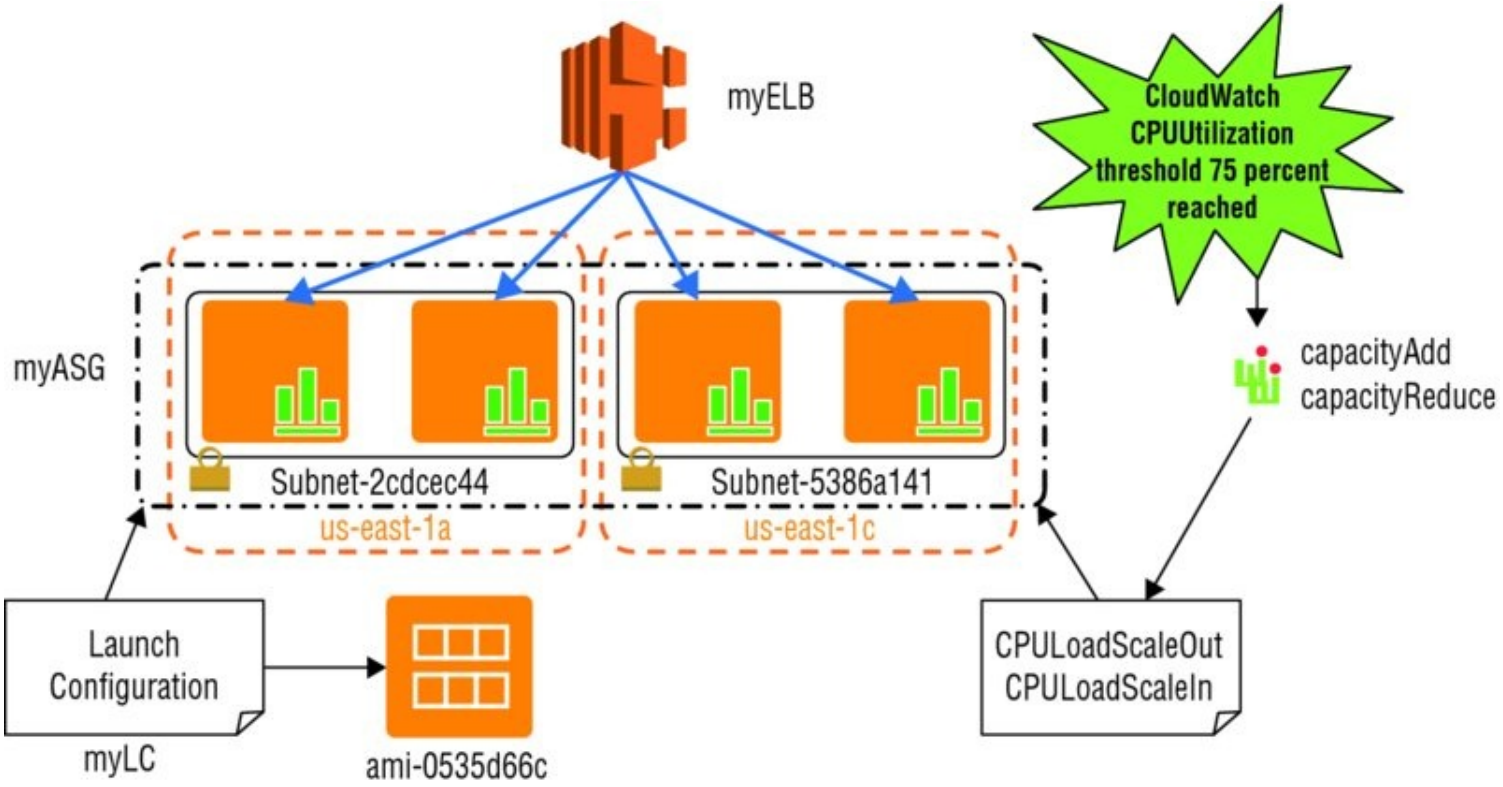


FIGURE 5.3 Amazon CloudWatch alarm triggering scaling out

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.