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

CloudWatch Alarms

A CloudWatch alarm watches over the value of a single metric. If the metric crosses a threshold that you specify (and stays there), the alarm will take an action. For example, you might configure an alarm to take an action when the average CPU utilization for an instance exceeds 80% for five minutes. The action can be one of the following:

Notification using Simple Notification Service The *Simple Notification Service* (SNS) allows applications, users, and devices to send and receive notifications from AWS. SNS uses a publisher-subscriber model, wherein a publisher such as an AWS service generates a notification and a subscriber such as an end user receives it. The communication channel that SNS uses to map publishers and subscribers is called a *topic*. SNS can send notifications to subscribers via a variety of protocols including the following:

- HTTP(S)
- Simple Queue Service (SQS)
- Lambda
- Mobile push notification
- Email
- Email-JSON
- Short Message Service (SMS) text messages

Auto Scaling action By specifying an EC2 Auto Scaling action, the EC2 Auto Scaling service can add or remove EC2 instances in response to changing demand. For example, if a metric indicates that instances are overburdened, you can have EC2 Auto Scaling respond by adding more instances.

EC2 action If you're monitoring a specific instance that's having a problem, you can use an EC2 action to stop, terminate, or recover the instance. Recovering an instance migrates the instance to a new EC2 host, something you may need to do if there's a physical hardware problem on the hardware hosting the instance.

CloudWatch Dashboards

CloudWatch dashboards are your one-stop shop for keeping an eye on all of your important metrics. You can create multiple dashboards and add to them metric graphs, the latest values for a metric, and CloudWatch alarms. You can save your dashboards for future use and share them with others. Dashboards can also visualize metrics from multiple AWS Regions, so you can keep an eye on the global health of your infrastructure. Check out Figure 6.17 for a sample CloudWatch dashboard.

EC2 Auto Scaling

EC2 Auto Scaling automatically launches preconfigured EC2 instances. The goal of Auto Scaling is to ensure you have just enough computing resources to meet user demand without over-provisioning.

Launch Configurations and Launch Templates

Auto Scaling works by spawning instances from either a launch configuration or a launch template. For the purposes of Auto Scaling, both achieve the same basic purpose of defining the instance's characteristics, such as AMI, disk configuration, and instance type. You can also install software and make custom configurations by placing commands into a Userdata script that automatically runs when Auto Scaling launches a new instance.

But there are some differences between launch configurations and launch templates. Launch templates are newer and can be used to spawn EC2 instances manually, even without Auto Scaling. You can also modify them after you create them. Launch configurations, on the other hand, can be used only with Auto Scaling. And once you create a launch configuration, you can't modify it.

Auto Scaling Groups

Instances created by Auto Scaling are organized into an Auto Scaling group. All instances in a group can be automatically registered with an application load balancer target group. The application load balancer distributes traffic to the instances, spreading the demand out evenly among them.

Desired Capacity

When you configure an Auto Scaling group, you define a desired capacity—the number of instances that you want Auto Scaling to create. Auto Scaling creates this many instances and strives to maintain the desired capacity. If you raise or lower the capacity, Auto Scaling launches or terminates instances to match.

Self-Healing

Failed instances are self-healing. If an instance fails or terminates, Auto Scaling re-creates a replacement. This way you always get the number of instances you expect.

Auto Scaling can use EC2 or elastic load balancing (ELB) health checks to determine whether an instance is healthy. EC2 health checks consider the basic health of an instance, whether it's running, and whether it has network connectivity. ELB health checks look at the health of the application running on an instance. An unhealthy instance is treated as a failed instance and is terminated, and Auto Scaling creates another in its place.

Scaling Actions

Scaling actions control when Auto Scaling launches or terminates instances. You control how many instances Auto Scaling launches or terminates by specifying a minimum and maximum group size. Auto Scaling will ensure the number of instances never goes outside of this range. Naturally, the desired capacity must rest within the minimum and maximum bounds.

Dynamic Scaling

With dynamic scaling, Auto Scaling launches new instances in response to increased demand using a process called *scaling out*. It can also scale in, terminating instances when demand ceases. You can scale in or out according to a metric, such as average CPU utilization of your instances, or based on the number of concurrent application users.

Scheduled Scaling

Instead of or in addition to dynamic scaling, Auto Scaling can scale in or out according to a schedule. This is particularly useful if your demand has predictable peaks and valleys.

Predictive scaling is a feature that looks at historic usage patterns and predicts future peaks. It then automatically creates a scheduled scaling action to match. It needs at least one day's worth of traffic data to create a scaling schedule.

EC2 Auto Scaling fulfills one of the core principles of sound cloud architecture design: don't guess your capacity needs. Auto Scaling can save money by reducing your capacity when you don't need it and improve performance by increasing it when you do.

Configuration Management

Configuration management is an approach to ensuring accurate and consistent configuration of your systems. While automation is concerned with carrying out tasks, configuration management is primarily concerned with enforcing and monitoring the internal configuration state of your instances to ensure they're what you expect. Such configuration states primarily include but aren't limited to operating system configurations and what software is installed.

As with automation in general, configuration management tools use either imperative or declarative approaches. AWS offers both approaches using two tools to help you achieve configuration management of your EC2 and on-premises instances: Systems Manager and OpsWorks.

Systems Manager

Systems Manager uses the imperative approach to get your instances and AWS environment into the state that you want.

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.

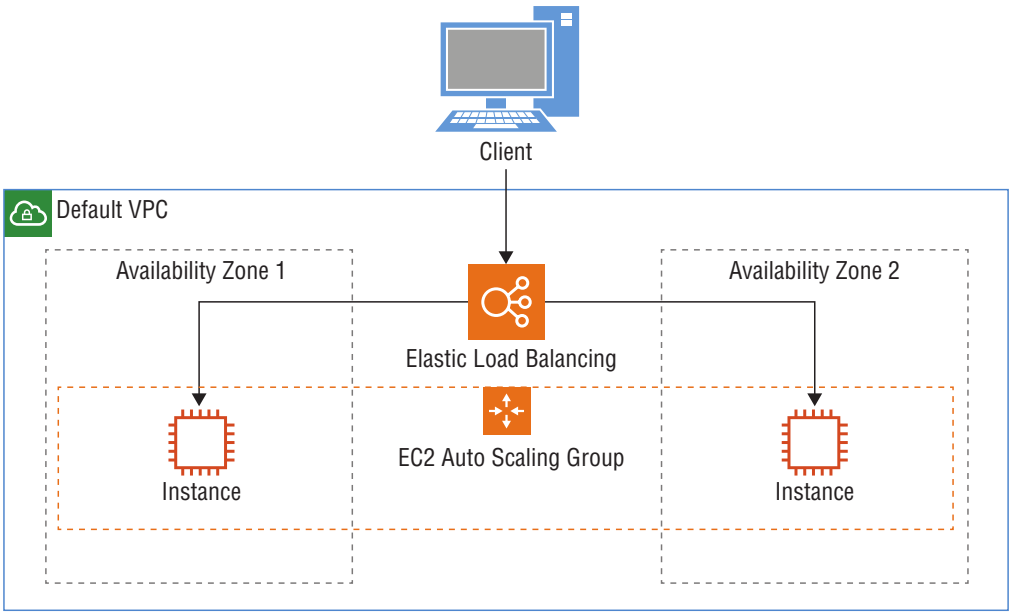
Cost optimization Automatically shut down or decommission unused resources. For example, implement S3 object life cycle configurations to delete unneeded objects. Or if you have EC2 instances that are used for testing only during business hours, you can automatically shut them down at the end of the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

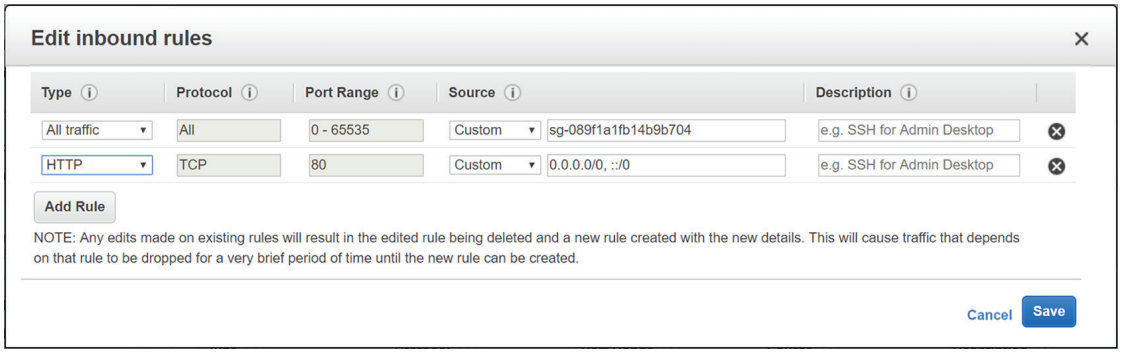
EXERCISE 12.1

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

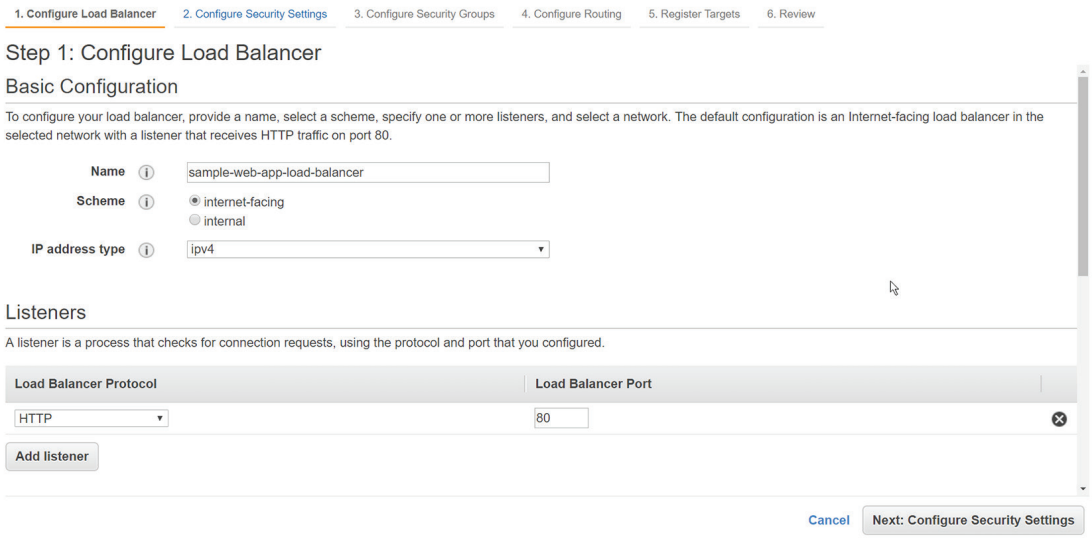
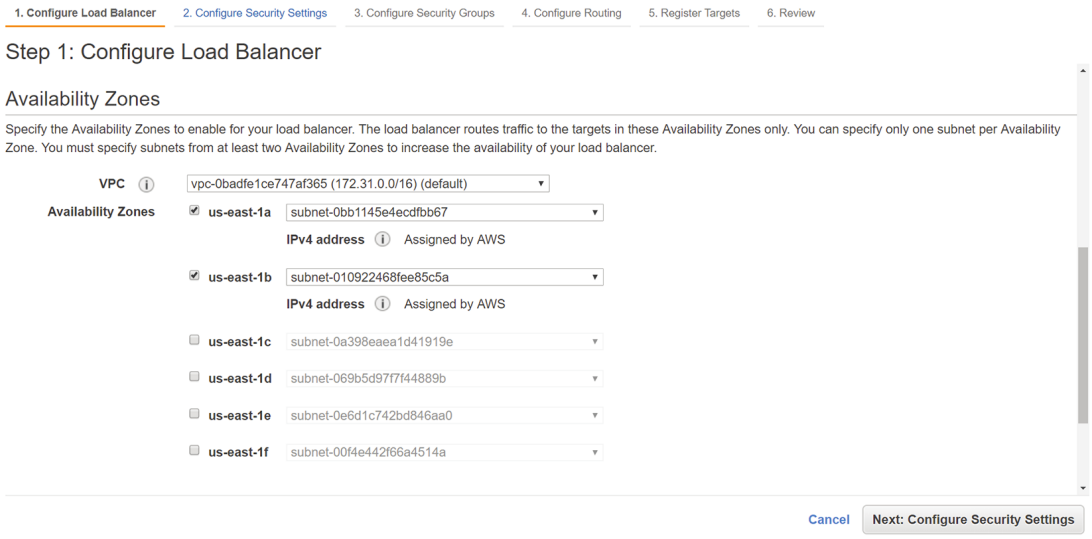


FIGURE 12.4 Application load balancer Availability Zones configuration



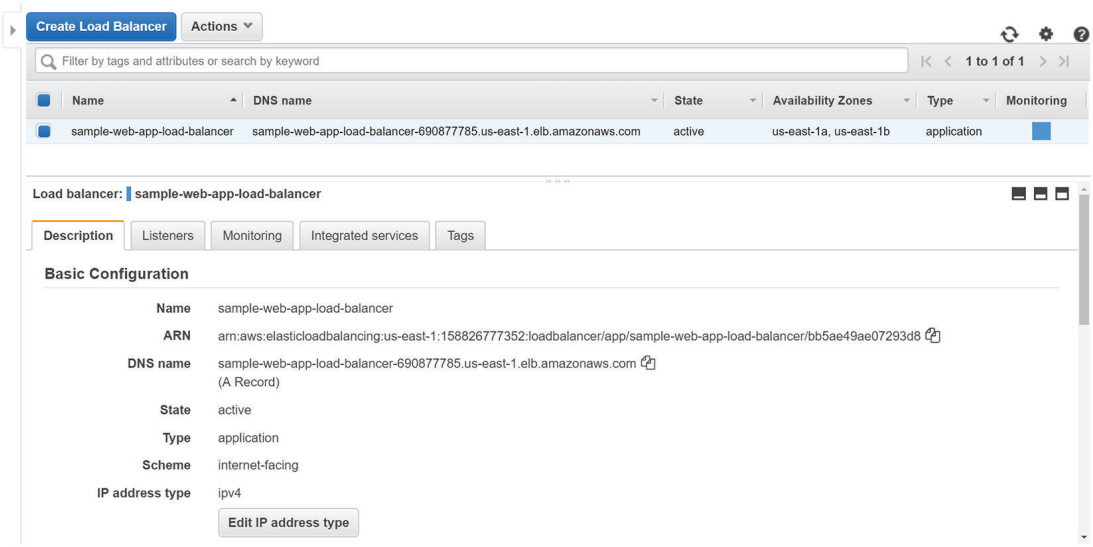
13. Select the button titled Next: Configure Routing.
14. Under Target Group, in the Target Group drop-down list, select New Target Group.
15. In the Name field, type **sample-web-app-target-group**.
16. Next to Target Type, select the Instance radio button.

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.

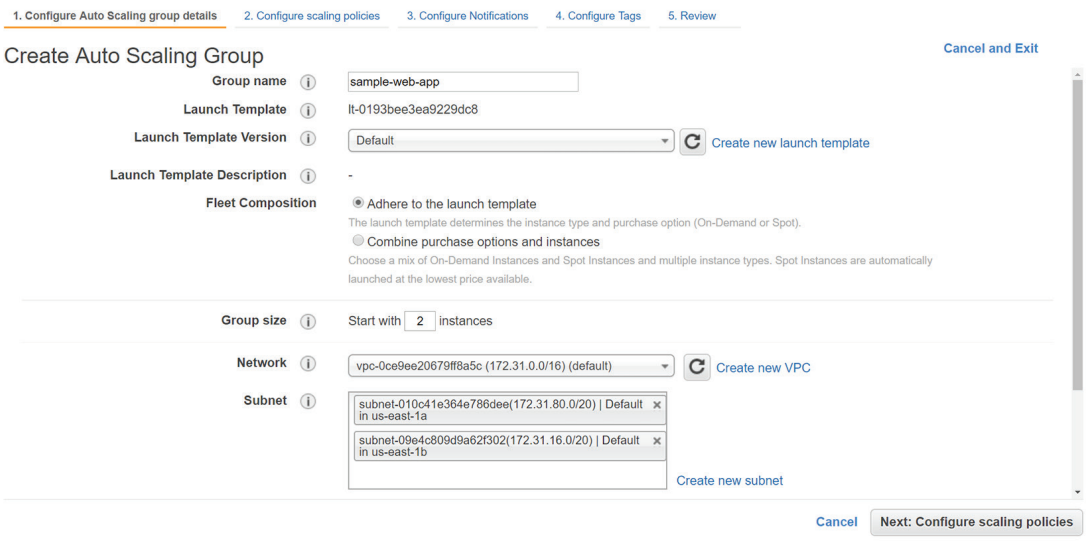
EXERCISE 12.4

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

When you're done experimenting, remember to delete the resources you created, including the Auto Scaling group, the instances it created, and the application load balancer.

Static Website Hosting Using S3

The next scenario you'll consider is a static website hosted on S3. Despite the term, a static website doesn't mean one that never changes. *Static* refers to the fact that the site's assets—HTML files, graphics, and other downloadable content such as PDF files—are just static files sitting in an S3 bucket. You can update these files at any time and as often as you want, but what's delivered to the end user is the same content that's stored in S3. This is in contrast to a dynamic website that uses server-side processing to modify the content on the fly just before sending it to the user. Some examples of dynamic websites are web-based email applications, database-backed ecommerce sites (like Amazon), and WordPress blogs. As a rule of thumb, if a website uses a database for storing any information, it's a dynamic website.

Setting up S3 to host a static website involves the following steps:

- Creating an S3 bucket
- Configuring it for static website hosting
- Uploading the web assets that contain the content you want to serve

By default, files in S3 buckets are not public and are accessible only by the account owner and any users the account owner has authorized. This has always been the case, but after several high-profile incidents of users unwittingly making files in their S3 buckets publicly available, AWS has taken steps to reduce this risk. This poses a challenge when you want everyone on the internet to be able to access your static website. Therefore, many of the steps you'll complete in Exercise 12.5 are there to overcome some of these important safeguards and ensure that your static website is in fact available to everyone.

EXERCISE 12.5

Create a Static Website Hosted Using S3

In this exercise, you're going to create a simple static website hosted by S3.

1. Browse to the S3 service console.
2. Choose the Create Bucket button.
3. Give the bucket a name of your choice. Try for something that's easy to type but unique (for this example, we used **benpiper2020**). Then choose the Next button.
4. Leave object versioning and encryption disabled by default. Enabling either won't impact your ability to use the bucket for hosting a static website. Choose the Next button.

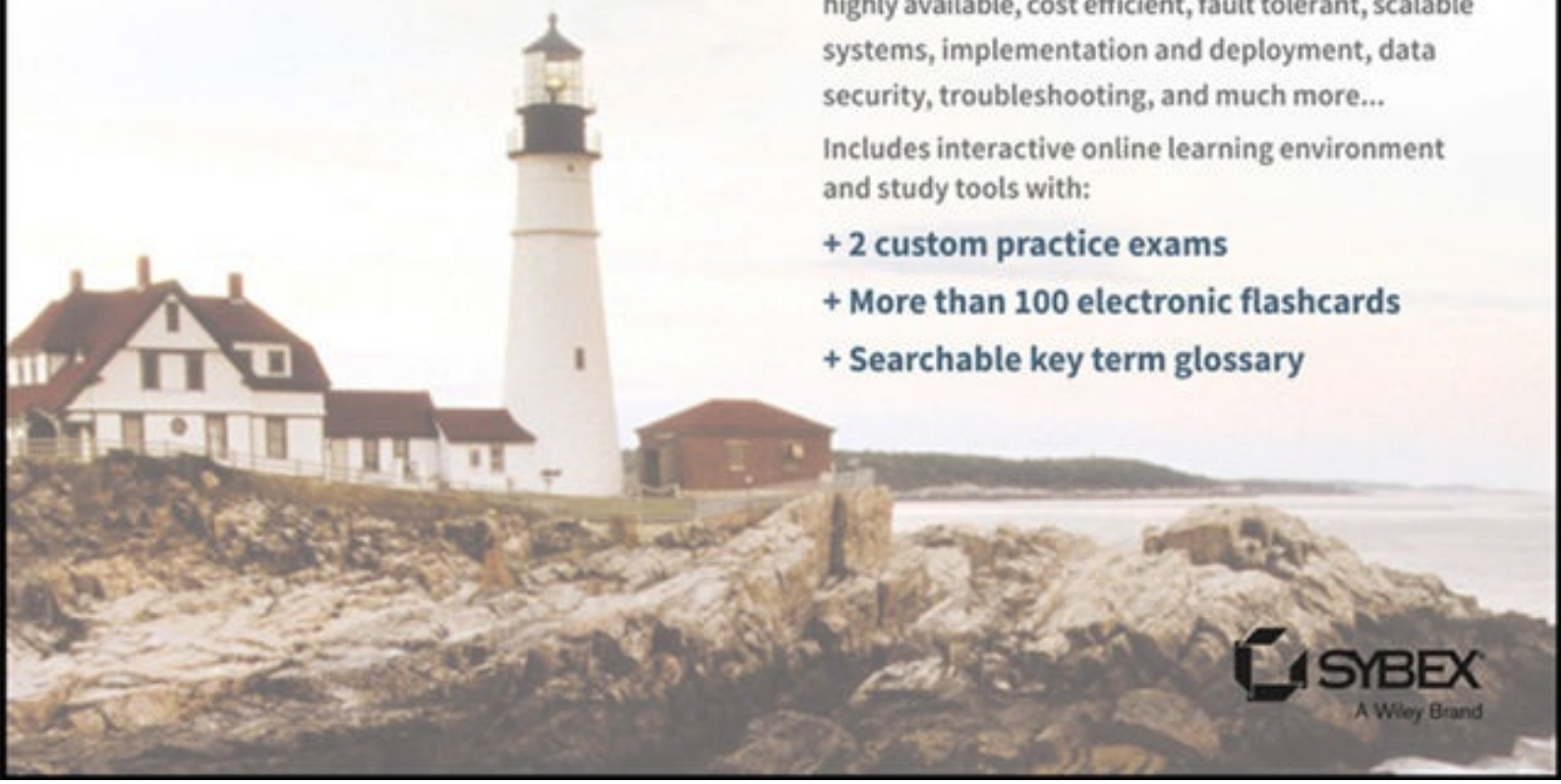
amazon web services

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

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.

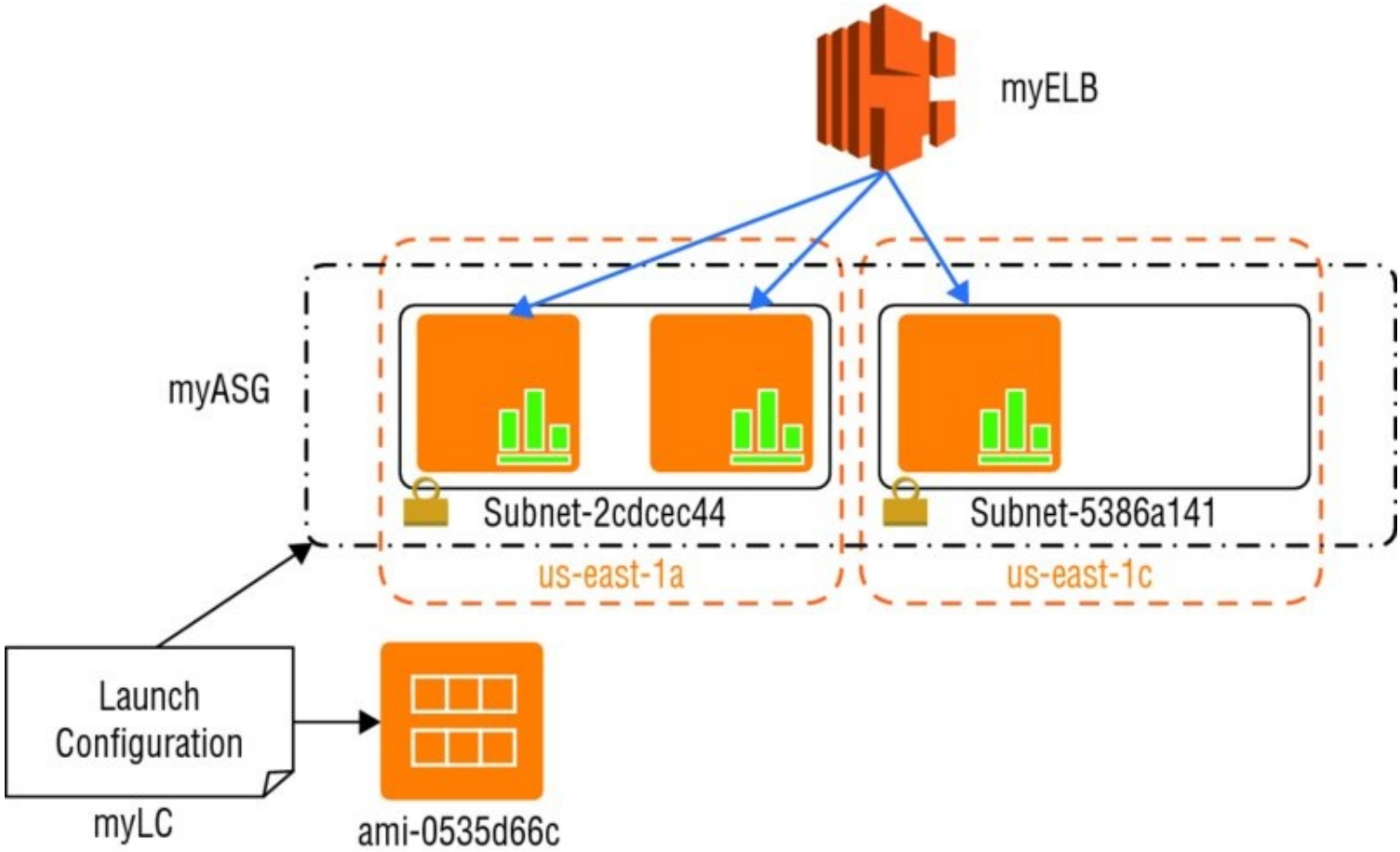


FIGURE 5.1 Auto Scaling group behind an Elastic Load Balancing load balancer

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).

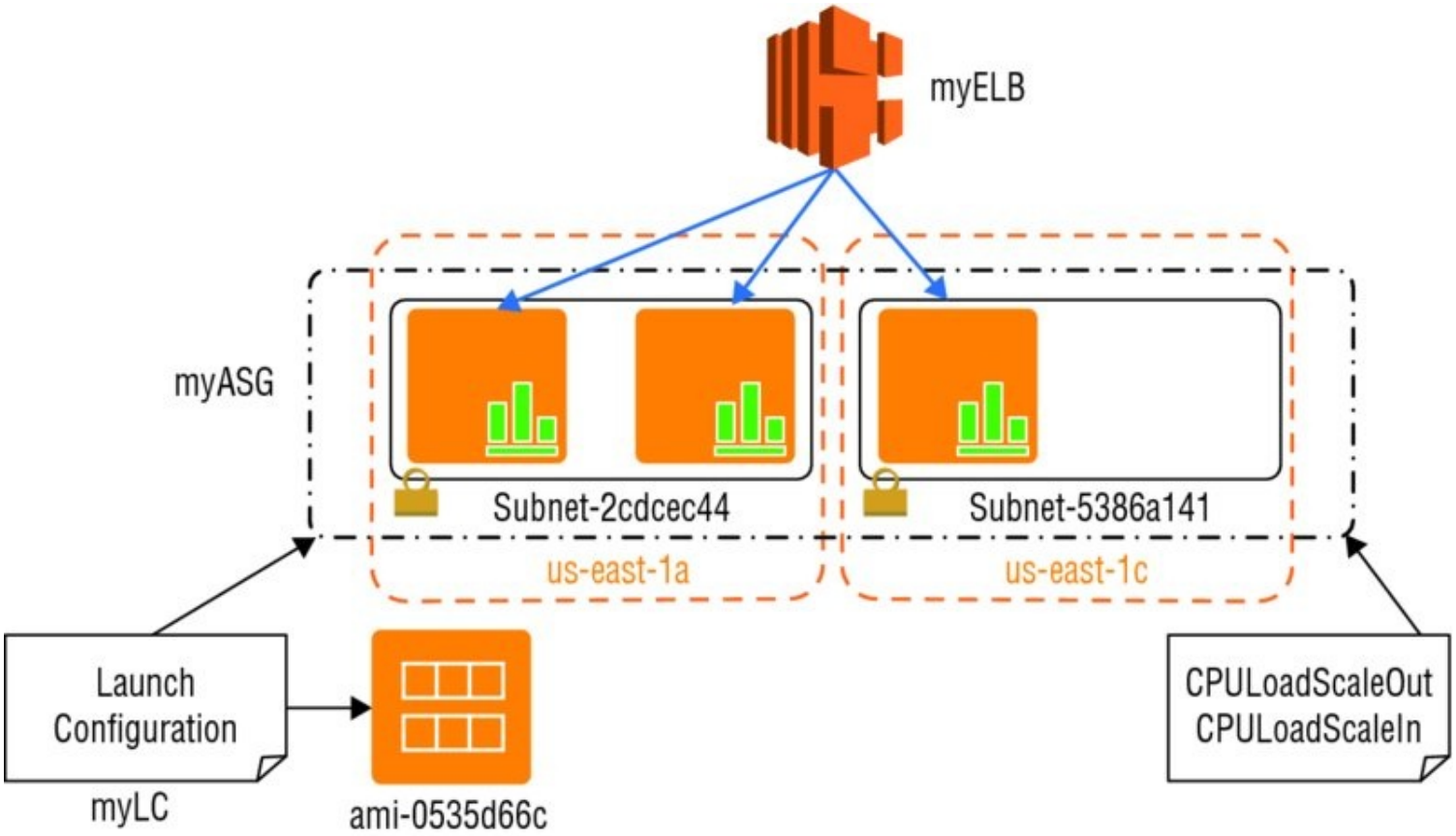


FIGURE 5.2 Auto Scaling group with policy

```
> 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`.

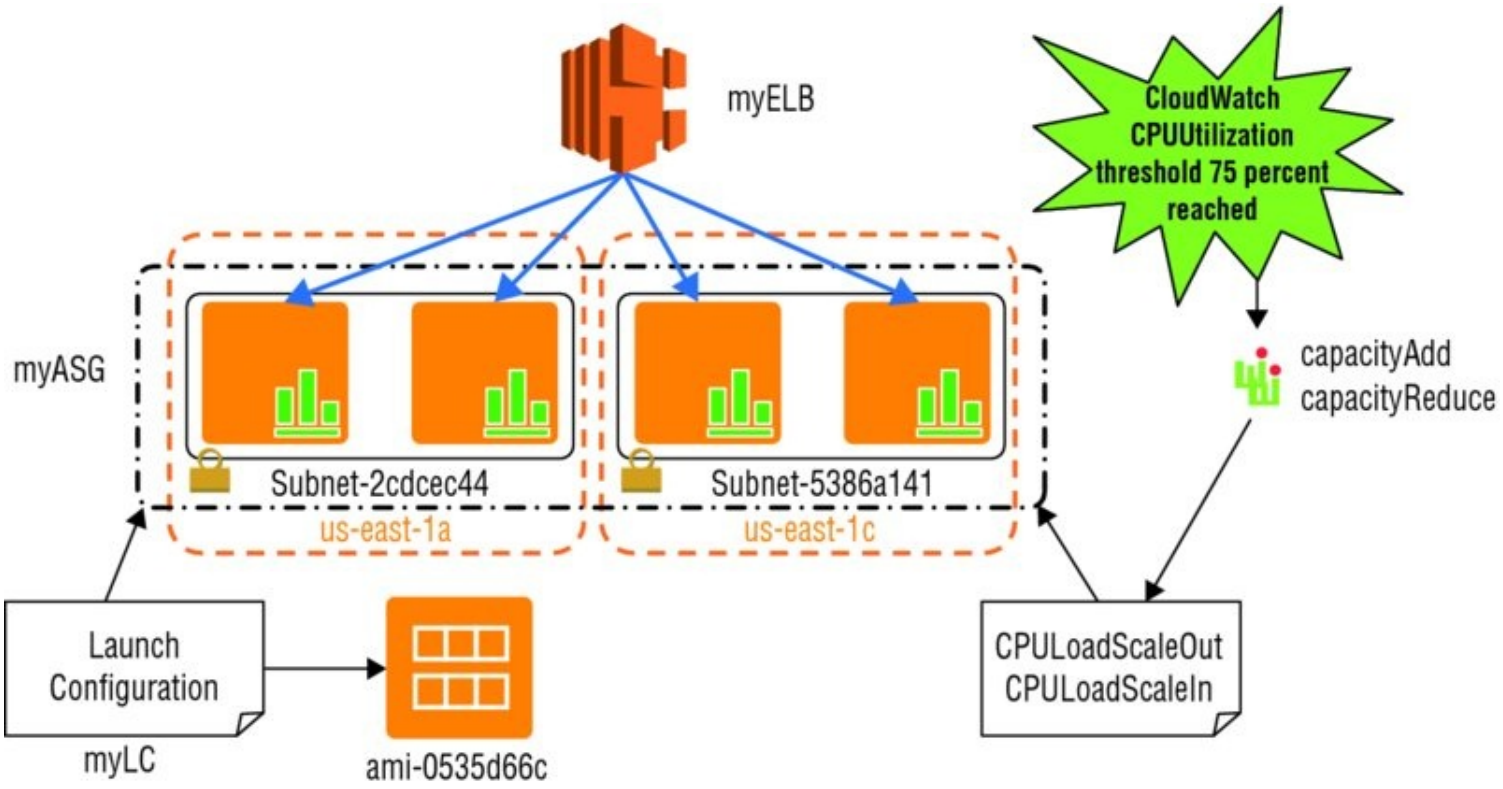


FIGURE 5.3 Amazon CloudWatch alarm triggering scaling out

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.

and predictable fashion. When you use AWS CloudFormation, you work with *templates* and *stacks*.

You create AWS CloudFormation templates to define your AWS resources and their properties. A *template* is a text file whose format complies with the JSON standard. AWS CloudFormation uses these templates as blueprints for building your AWS resources.

When you use AWS CloudFormation, you can reuse your template to set up your resources consistently and repeatedly. Just describe your resources once, and then provision the same resources over and over in multiple regions.

When you use AWS CloudFormation, you manage related resources as a single unit called a stack. You create, update, and delete a collection of resources by creating, updating, and deleting stacks. All of the resources in a stack are defined by the stack's AWS CloudFormation template. Suppose you created a template that includes an Auto Scaling group, Elastic Load Balancing load balancer, and an Amazon RDS database instance. To create those resources, you create a stack by submitting your template that defines those resources, and AWS CloudFormation handles all of the provisioning for you. After all of the resources have been created, AWS CloudFormation reports that your stack has been created. You can then start using the resources in your stack. If stack creation fails, AWS CloudFormation rolls back your changes by deleting the resources that it created.

Often you will need to launch stacks from the same template, but with minor variations, such as within a different Amazon VPC or using AMIs from a different region. These variations can be addressed using parameters. You can use parameters to customize aspects of your template at runtime, when the stack is built. For example, you can pass the Amazon RDS database size, Amazon EC2 instance types, database, and web server port numbers to AWS CloudFormation when you create a stack. By leveraging template parameters, you can use a single template for many infrastructure deployments with different configuration values. For example, your Amazon EC2 instance types, Amazon CloudWatch alarm thresholds, and Amazon RDS read-replica settings may differ among AWS regions if you receive more customer traffic in the United States than in Europe. You can use template parameters to tune the settings and thresholds in each region separately and still be sure that the application is deployed consistently across the regions.

Figure 11.8 depicts the AWS CloudFormation workflow for creating stacks.

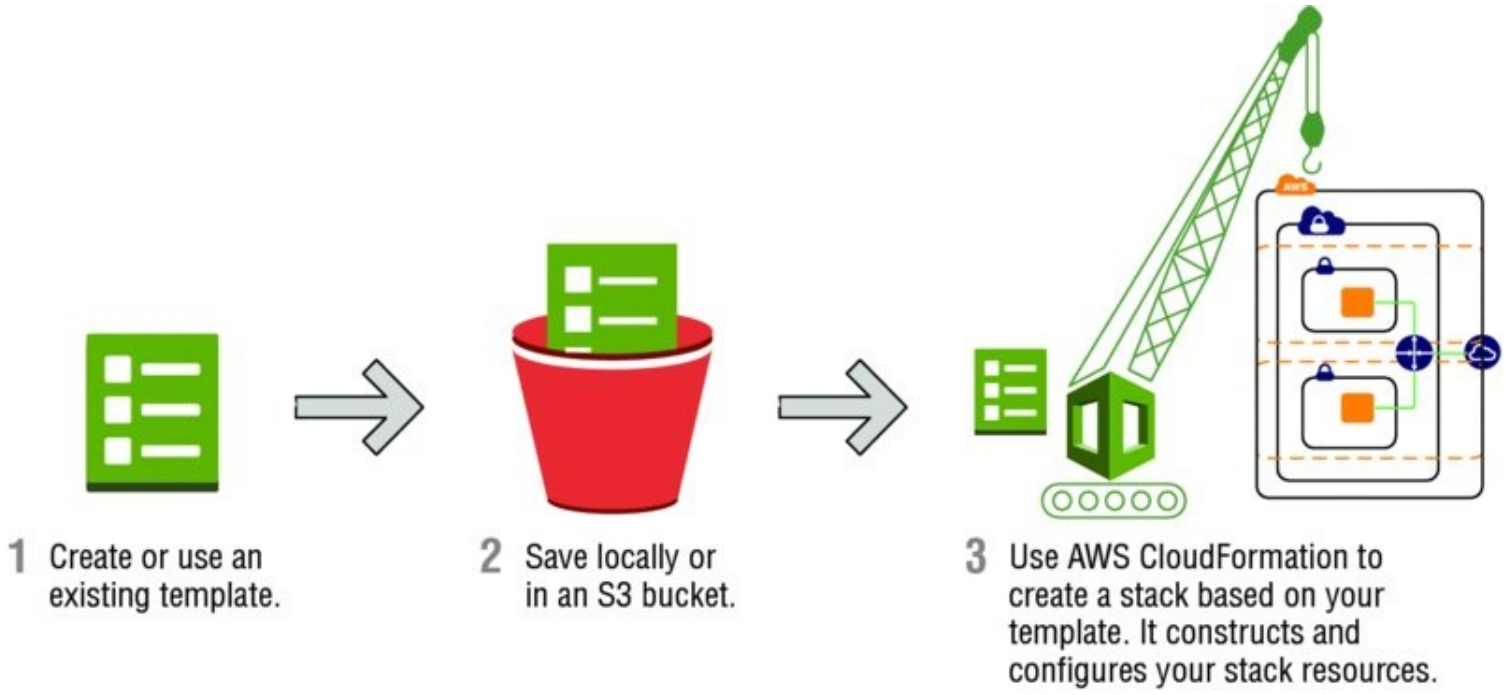


FIGURE 11.8 Creating a stack workflow

Because environments are dynamic in nature, you inevitably will need to update your stack's resources from time to time. There is no need to create a new stack and delete the old one; you can simply modify the existing stack's template. To update a stack, create a *change set* by submitting a modified version of the original stack template, different input parameter values, or both. AWS CloudFormation compares the modified template with the original template and generates a change set. The change set lists the proposed changes. After reviewing the changes, you can execute the change set to update your stack. Figure 11.9 depicts the workflow for updating a stack.

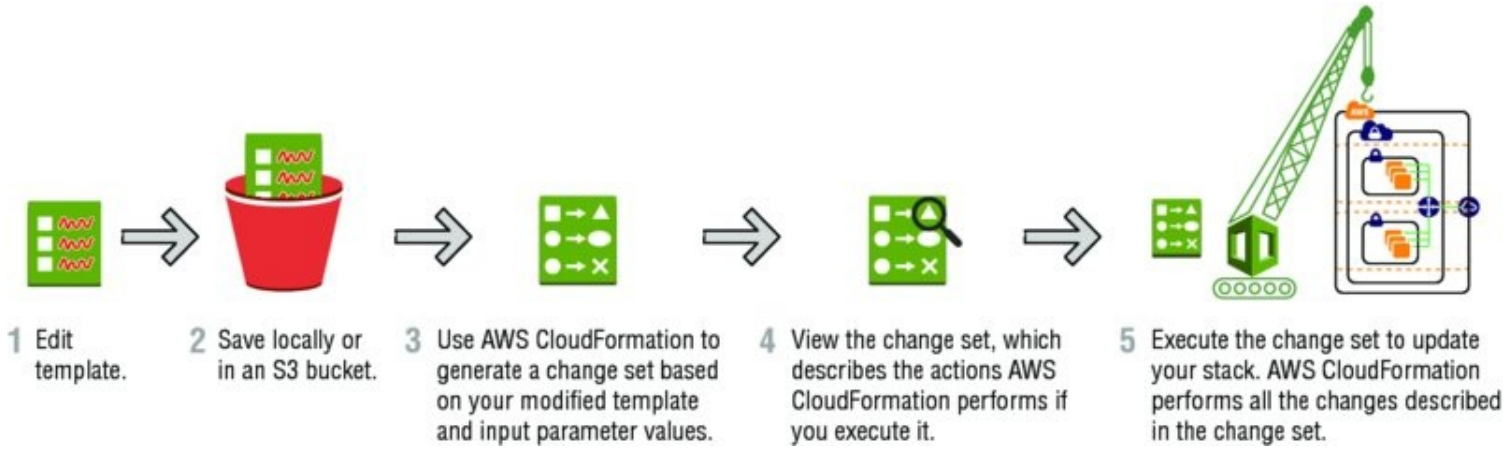


FIGURE 11.9 Updating a stack workflow

When the time comes and you need to delete a stack, AWS CloudFormation deletes the stack and all of the resources in that stack.

If you want to delete a stack but still retain some resources in that stack, you can use a deletion policy to retain those resources. If a resource has no deletion policy, AWS CloudFormation deletes the resource by default.

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

Use Case

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

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

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

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

AWS Elastic Beanstalk

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

Overview

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

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

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

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

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

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

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

Use Cases

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

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

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

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

Key Features

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

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

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

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

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

AWS Trusted Advisor

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

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

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

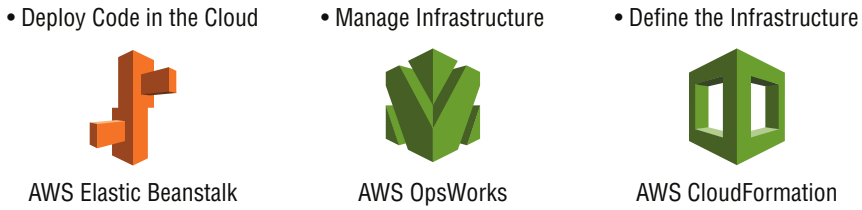
AWS®

Certified Developer

Official Study Guide

Associate (DVA-C01) Exam

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

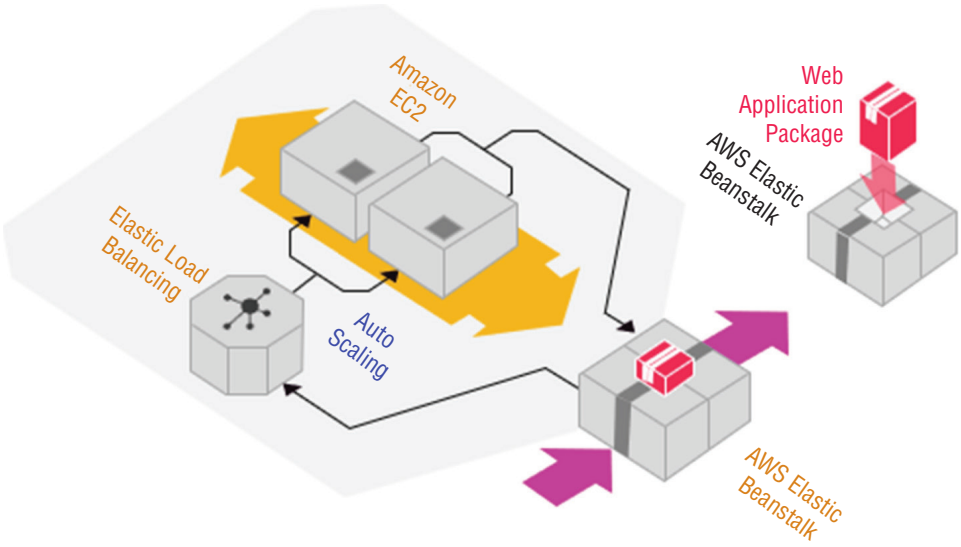
- Environments
- Application versions
- Environment configurations

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.

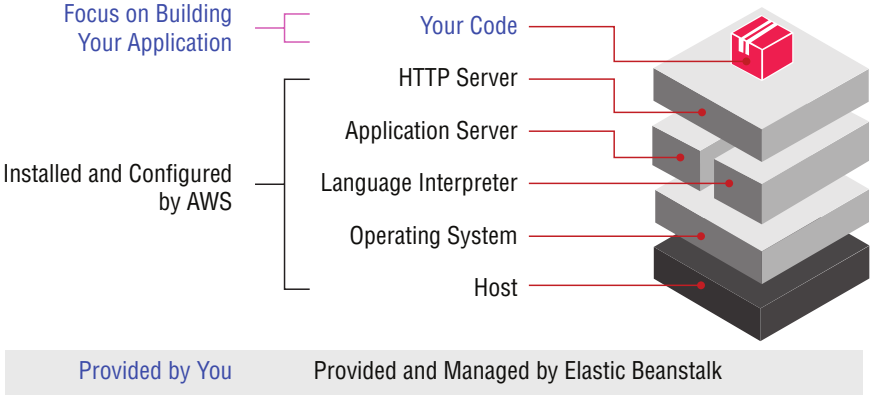
- Single-instance (development, low cost)
- Load balanced, AWS Auto Scaling (production)

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

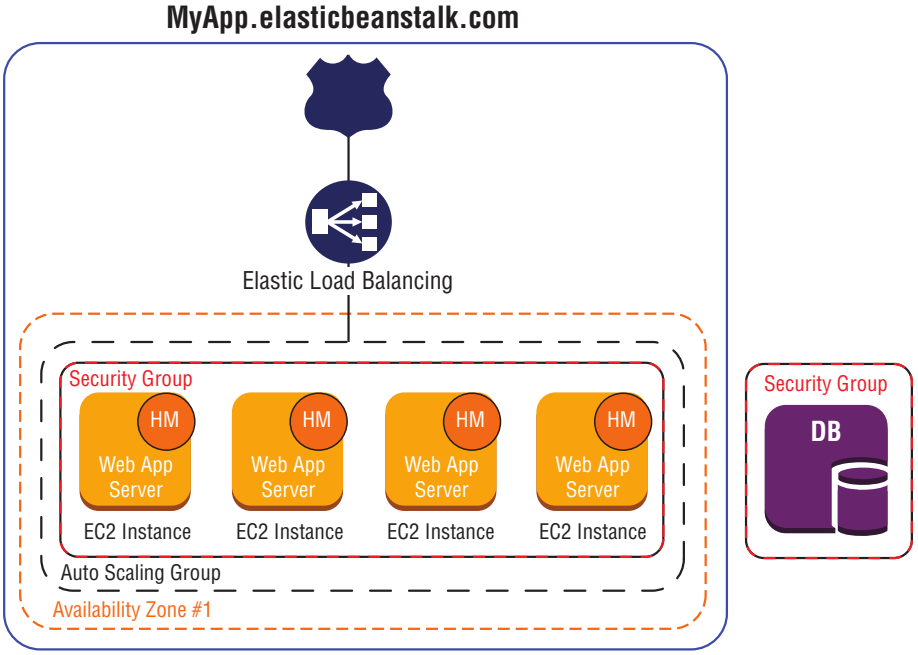
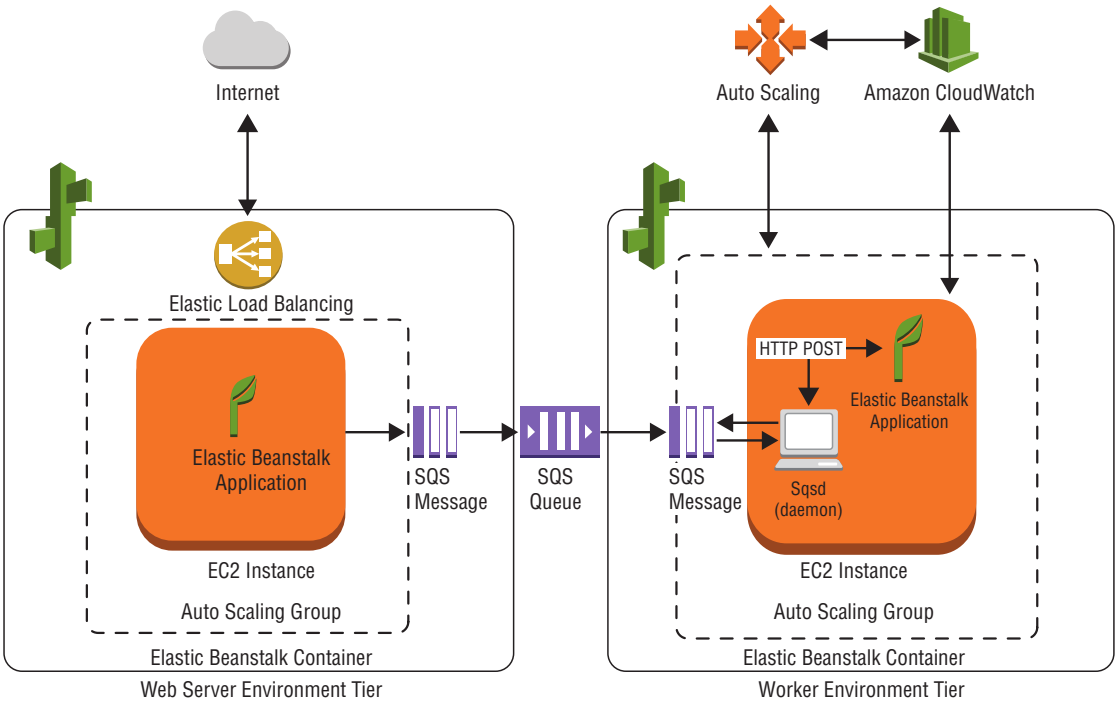


Figure 6.9 shows a worker environment architecture, where AWS resources create configurations, such as Auto Scaling groups, Amazon EC2 instances, and an IAM role, to manage resources for your worker applications.

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.

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.

Deploy to Amazon EC2 Auto Scaling Groups

When you deploy to Amazon EC2 Auto Scaling groups, AWS CodeDeploy will automatically run the latest successful deployment on any new instances created when the group scales out. If the deployment fails on an instance, it updates to maintain the count of healthy instances. For this reason, AWS does not recommend that you associate the same Auto Scaling group with multiple deployment groups (for example, you want to deploy multiple applications to the same Auto Scaling group). If both deployment groups perform a deployment at roughly the same time and the first deployment fails on the new instance, it terminates by AWS CodeDeploy. The second deployment, unaware that the instance terminated, will not fail until the deployment times out (*the default timeout value is 1 hour*). Instead, you should combine your application deployments into one or consider the use of multiple Auto Scaling groups with smaller instance types.

Deployment Configuration

You use *deployment configurations* to drive how quickly Amazon EC2 on-premises instances update by AWS CodeDeploy. You can configure deployments to deploy to all instances in a deployment group at once or subgroups of instances at a time, or you can create an entire new group of instances (*blue/green deployment*). A deployment configuration also specifies the fault tolerance of deployments, so you can roll back changes if a specified number or percentage of instances or functions in your deployment group fail to complete their deployments and signal success back to AWS CodeDeploy.

Amazon EC2 On-Premises Deployment Configurations

When you deploy to Amazon EC2 on-premises instances, you can configure either in-place or blue/green deployments.

In-Place deployments These deployments recycle currently running instances and deploy revisions on existing instances.

Blue/Green deployments These deployments replace currently running instances with sets of newly created instances.

In both scenarios, you can specify wait times between groups of deployed instances (batches). Additionally, if you register the deployment group with an *elastic load balancer*, newly deployed instances also register with the load balancer and are subject to its health checks.

The deployment configuration specifies success criteria for deployments, such as the minimum number of healthy instances that must pass health checks during the deployment process. This is done to maintain required availability during application updates. AWS CodeDeploy provides three built-in deployment configurations.

CodeDeployDefault.AllAtOnce

For in-place deployments, AWS CodeDeploy will attempt to deploy to all instances in the deployment group at the same time. The success criteria for this deployment configuration

Know how to use the credential helper to connect to repositories. It is possible to connect to AWS CodeCommit repositories using IAM credentials. The AWS CodeCommit credential helper translates an IAM access key and secret access key into valid Git credentials. This requires the AWS CLI and a Git configuration file that specifies the credential helper.

Understand the different strategies for migrating to AWS CodeCommit. You can migrate an existing Git repository by cloning to your local workstation and adding a new remote, pointing to the AWS CodeCommit repository you create. You can push the repository contents to the new remote. You can migrate unversioned content in a similar manner; however, you must create a new local Git repository (instead of cloning an existing one). Large repositories can be migrated incrementally because large pushes may fail because of network issues.

Know the basics of AWS CodeBuild. AWS CodeBuild allows you to perform long-running build tasks repeatedly and reliably without having to manage the underlying infrastructure. You are responsible only for specifying the build environment settings and the actual tasks to perform.

Know the basics of AWS CodeDeploy. AWS CodeDeploy standardizes and automates deployments to Amazon EC2 instances, on-premises servers, and AWS Lambda functions. Deployments can include application/static files, configuration tasks, or arbitrary scripts to execute. For Amazon EC2 on-premises deployments, a lightweight agent is required.

Understand how AWS CodeDeploy works with Amazon EC2 Auto Scaling groups. When you deploy to Amazon EC2 Auto Scaling groups, AWS CodeDeploy will automatically run the last successful deployment on any new instances that you add to the group. If the deployment fails on the instance, it will be terminated and replaced (to maintain the desired count of healthy instances). If two deployment groups for separate AWS CodeDeploy applications specify the same Auto Scaling group, issues can occur. If both applications deploy at roughly the same time and one fails, the instance will be terminated before success/failure can be reported for the second application deployment. This will result in AWS CodeDeploy waiting until the timeout period expires before taking any further action.

Resources to Review

What is DevOps?

```
https://aws.amazon.com/devops/what-is-devops/
```

AWS DevOps Blog:

```
https://aws.amazon.com/blogs/devops/
```

Introduction to DevOps on AWS:

```
https://d1.awsstatic.com/whitepapers/AWS_DevOps.pdf
```

Practicing Continuous Integration and Continuous Delivery on AWS:

```
https://d1.awsstatic.com/whitepapers/DevOps/practicing-continuous-
integration-continuous-delivery-on-AWS.pdf
```

You can configure the creation policy to require a specific number of signals in a certain amount of time; otherwise, the resource will show `CREATE_FAILED`. Signals sent to a resource are visible events in the AWS CloudFormation stack logs.

You can define creation policies with this syntax. When you configure creation policies for AWS Auto Scaling groups, you must specify the `MinSuccessfulInstancesPercent` property so that a certain percentage of instances in the group setup successfully complete before the group itself shows `CREATE_COMPLETE`. You can also configure creation policies to require a certain number of signals (`Count`) in a certain amount of time (`Timeout`). The following code example displays an AWS Auto Scaling group resource with a creation policy. This policy specifies that at least three signals must be received in 15 minutes for the group to create successfully.

```
"AutoScalingGroup": {
  "Type": "AWS::AutoScaling::AutoScalingGroup",
  "Properties": {
    "AvailabilityZones": { "Fn::GetAZs": "" },
    "LaunchConfigurationName": { "Ref": "LaunchConfig" },
    "DesiredCapacity": "3",
    "MinSize": "1",
    "MaxSize": "4"
  },
  "CreationPolicy": {
    "ResourceSignal": {
      "Count": "3",
      "Timeout": "PT15M"
    }
  }
}
```

Wait Conditions

You can use the *`WaitCondition`* property to insert arbitrary pauses until resources complete. If you require additional tracking of stack creation, you can use the `WaitCondition` property to add pauses to wait for external configuration tasks. An example of this would be if you create an Amazon DynamoDB table with a custom resource associated with AWS Lambda to load data into the table and then install software on an Amazon EC2 instance that reads data from the table. You can insert a `WaitCondition` into this template to prevent the creation of the instance until the custom resource function signals that data has been successfully loaded.

For Amazon EC2 instances and AWS Auto Scaling groups, we recommend that you use creation policies instead of wait conditions.

Wait conditions consist of two resources in a template, an `AWS::CloudFormation::WaitCondition` (wait condition) and an `AWS::CloudFormation::WaitConditionHandle` (wait condition handle).

The first resource, the wait condition, is similar to a creation policy. It requires a signal count and timeout value. However, it also requires a reference to a wait condition handle. The wait condition handle acts as a reference to a presigned URL where signals are sent to AWS CloudFormation, which it monitors.

Wait condition handles should never be reused between stack creation and subsequent updates, as it may result in signals from previous stack actions being evaluated. Instead, create new wait conditions for each stack action.

In the following example, the `WebServerGroup` resource creates an AWS Auto Scaling group with a count equal to the `WebServerCapacity` parameter. The example also creates a wait condition and wait condition handle, where the wait condition handle expects a number of signals equal to the `WebServerCapacity` parameter.

```
"WebServerGroup" : {
   "Type" : "AWS::AutoScaling::AutoScalingGroup",
   "Properties" : {
      "AvailabilityZones" : { "Fn::GetAZs" : "" },
      "LaunchConfigurationName" : { "Ref" : "LaunchConfig" },
      "MinSize" : "1",
      "MaxSize" : "5",
      "DesiredCapacity" : { "Ref" : "WebServerCapacity" },
      "LoadBalancerNames" : [ { "Ref" : "ElasticLoadBalancer" } ]
   }
},
"WaitHandle" : {
   "Type" : "AWS::CloudFormation::WaitConditionHandle"
},
"WaitCondition" : {
   "Type" : "AWS::CloudFormation::WaitCondition",
   "DependsOn" : "WebServerGroup",
   "Properties" : {
      "Handle"  : { "Ref" : "WaitHandle" },
      "Timeout" : "300",
      "Count"   : { "Ref" : "WebServerCapacity" }
   }
}
```

With this approach, you need to ensure that the signal is sent to the wait condition handle. This is done in the Amazon EC2 instance's user data, which you define in the launch configuration for AWS Auto Scaling groups. In this case, the `LaunchConfig` resource must include a signal to the wait condition handle. To do this, you reference the wait condition handle within the launch configuration's `UserData` script.

```
"UserData" : {
   "Fn::Base64" : {
       "Fn::Join" : [ "", ["SignalURL=", { "Ref" : "myWaitHandle" } ] ]
   }
}
```

Within `UserData`, you can use a `curl` command to send the success signal back to AWS CloudFormation.

```
curl -T /tmp/a "WAIT_CONDITION_HANDLE_URL"
```

The file `/tmp/a` must be in the following format:

```
{
   "Status" : "SUCCESS",
   "Reason" : "Configuration Complete",
   "UniqueId" : "ID1234",
   "Data" : "Application has completed configuration."
}
```

The `Data` section of the JSON response can include arbitrary data about the signal. You can make it accessible in the AWS CloudFormation template with the `Fn::GetAtt` intrinsic function.

```
"Outputs": {
    "WaitConditionData" : {
        "Value" : { "Fn::GetAtt" : [ "mywaitcondition", "Data" ]},
        "Description" : "The data passed back as part of signalling the
WaitCondition"
    }
}
```

Stack Create, Update, and Delete Statuses

Whenever you perform an action on an AWS CloudFormation stack, the end result will bring the stack into one of three possible statuses: Create, Update, and Delete. These statuses are visible in the AWS CloudFormation console, or if you use the `DescribeStacks` action.

CREATE_COMPLETE

The stack has created successfully.

CREATE_IN_PROGRESS

The stack is currently undergoing creation. No error has been detected.

CREATE_FAILED

One or more resources has failed to create successfully, causing the entire stack creation to fail. Review the stack failure messages to determine which resource(s) failed to create.

DELETE_COMPLETE

The stack has deleted successfully and will remain visible for 90 days.

DELETE_IN_PROGRESS

The stack is currently deleting.

DELETE_FAILED

The stack delete action has failed because of one or more underlying resources failing to delete. Review the stack output events to determine which resource(s) failed to delete. There you can manually delete the resource to prevent the stack delete from failing again.

ROLLBACK_COMPLETE

If a stack creation action fails to complete, AWS CloudFormation will automatically attempt to roll the stack back and delete any created resources. This status is achieved when the resources have been removed.

ROLLBACK_IN_PROGRESS

The stack has failed to create and is currently rolling back.

ROLLBACK_FAILED

If AWS CloudFormation is not able to delete resources that were provisioned during a failed stack create action, the stack will enter `ROLLBACK_FAILED`. The remaining resources will not be deleted until the error condition is corrected. Other than attempting to continue deleting the stack, no other actions can be performed on the stack itself. To resolve this, review the stack events to determine which resource(s) failed to delete.

UPDATE_COMPLETE

The stack has updated successfully.

UPDATE_IN_PROGRESS

The stack is currently performing an update.

UPDATE_COMPLETE_CLEANUP_IN_PROGRESS

When AWS CloudFormation updates certain resources, the type of update may require a replacement of the original physical resource. In these situations, AWS CloudFormation will first create the replacement resource and verify that the provision was successful. After all resources update, the stack will enter this phase and remove any previous resources. For example, when you update

the `Name` property of an `AWS::S3::Bucket` resource, AWS CloudFormation will create a bucket with the new name value and then delete the previous bucket during the cleanup phase.

UPDATE_ROLLBACK_COMPLETE

If a stack update fails, AWS CloudFormation will attempt to roll the stack back to the last working state. Once complete, the stack will enter the `UPDATE_ROLLBACK_COMPLETE` state.

UPDATE_ROLLBACK_IN_PROGRESS

After a stack update fails, AWS CloudFormation begins to roll back any changes to bring the stack back to the last working state.

UPDATE_ROLLBACK_COMPLETE_CLEANUP_IN_PROGRESS

A failed rollback will require a cleanup of any newly created resources that would have originally replaced existing ones. During this phase, replacement resources are deleted in place of the originals.

UPDATE_ROLLBACK_FAILED

If the stack update fails and the rollback is unable to return it to a working state, it will enter `UPDATE_ROLLBACK_FAILED`. You can delete the entire stack. Otherwise, you can review to determine what failed to roll back and continue the update rollback again.

Stack Updates

You do not need to re-create stacks any time you need to update an underlying resource. You can modify and resubmit the same template, and AWS CloudFormation will parse it for changes and apply the modifications to the resources. This can include the ability to add new resources or modify and delete existing ones. You can perform stack updates when you create a new template or parameters directly, or you can create a change set with the updates.

Some template sections, such as `Metadata`, require you to modify one or more resources when the stack updates, as you cannot change them on their own. You can change parameters without modifying the stack's template.

When performing a stack action, such as an update, one or more stack events are created. The event contains information such as the resource being modified, the action being performed, and resource IDs. One critical piece of information in the stack event is the `ClientRequestToken`.

All events triggered by a single stack action are assigned the same token value. For example, if a stack update modifies an Amazon S3 bucket and Amazon EC2 instance, the corresponding Amazon S3 and Amazon EC2 API calls will contain the same request token. This lets you easily track what API activity corresponds to particular stack actions. This API activity can be tracked in AWS CloudTrail and stored in Amazon S3 for later review.

When you update a stack, underlying resources can exhibit one of several behaviors. This depends on the update to the resource property or properties. Resource property changes can cause one of update types to occur, as shown in Table 8.1.

TABLE 8.1 AWS CloudFormation Update Types

Update Type	Resource Downtime	Resource Replacement
Update with No Interruption	No	No
Update with Some Interruption	Yes	No
Replacing Update	Yes	Yes

For resource properties that require replacement, the resource's physical ID will change.

Some resource properties do not support updates. In these cases, you must create new resources first. After this, you can remove the original resource from the stack.

Update Policies

You use the AWS CloudFormation *`UpdatePolicy`* to determine how to respond to changes to `AWS::AutoScaling::AutoScalingGroup` and `AWS::Lambda::Alias` resources.

For AWS Auto Scaling group update policies, there are policies that you can enforce. These depend on the type of change you make and whether you configure the AWS Auto Scaling scheduled actions. Table 8.2 displays the types of policies that take effect under each scenario.

TABLE 8.2 AWS Auto Scaling Update Types in AWS CloudFormation

AWS Auto Scaling Update Type	Change AWS Auto Scaling group Launch Configuration	Change AWS Auto Scaling group `VPCZoneIdentifier` Property	AWS Auto Scaling group Has a Scheduled Action
`AutoScalingReplacingUpdate`	X	X	
`AutoScalingRollingUpdate`	X	X	
`AutoScalingScheduledAction`			X

You can configure the `WillReplace` property for an `UpdatePolicy` to `true` and give precedence to the `AutoScalingReplacingUpdate` settings.

The `AutoScalingReplacingUpdate` policy defines how to replace updates. You can replace the entire AWS Auto Scaling group or only instances inside.

```
"UpdatePolicy" : {
  "AutoScalingReplacingUpdate" : {
    "WillReplace" : Boolean
  }
}
```

The `AutoScalingRollingUpdate` policy allows you to define the update process for instances in an AWS Auto Scaling group. This lets you configure the group to update instances all at once, in batches, or with an additional batch.

SuspendProcesses Attribute

The `SuspendProcesses` attribute can define whether to suspend AWS Auto Scaling scheduled actions or those you invoke by alarms, which can otherwise cause the update to fail.

```
"UpdatePolicy" : {
  "AutoScalingRollingUpdate" : {
    "MaxBatchSize" : Integer,
    "MinInstancesInService" : Integer,
    "MinSuccessfulInstancesPercent" : Integer
    "PauseTime" : String,
    "SuspendProcesses" : [ List of processes ],
    "WaitOnResourceSignals" : Boolean
  }
}
```

Lastly, for AWS Auto Scaling groups, the `AutoScalingScheduledAction` property defines whether to adhere to the group sizes (minimum, maximum, and desired counts) you define in your template. If your AWS Auto Scaling group has enabled scheduled actions, there is a possibility that the actual group sizes no longer reflect those in the template. If you run an update without this policy set, it can cause the group to be reverted to its original size. If you configure the `IgnoreUnmodifiedGroupSizeProperties` property to `true`, it will cause

AWS CloudFormation to ignore different group sizes when it compares the template to the actual AWS Auto Scaling group.

```
"UpdatePolicy" : {
  "AutoScalingScheduledAction" : {
    "IgnoreUnmodifiedGroupSizeProperties" : Boolean
  }
}
```

For changes to an `AWS::Lambda::Alias` resource, you can define the `CodeDeployLambdaAliasUpdate` policy. This controls whether a deployment is made with AWS CodeDeploy whenever it detects version changes.

```
"UpdatePolicy" : {
  "CodeDeployLambdaAliasUpdate" : {
    "AfterAllowTrafficHook" : String,
    "ApplicationName" : String,
    "BeforeAllowTrafficHook" : String,
    "DeploymentGroupName" : String
  }
}
```

In the previous examples, you only require the `ApplicationName` and `DeploymentGroupName` properties. These refer to the AWS CodeDeploy application and deployment group, which should update when the alias changes.

Deletion Policies

When you delete a stack, by default all underlying stack resources are also deleted. If this behavior is not desirable, apply the *`DeletionPolicy`* to resources in the stack to modify their behavior when the stack is deleted. You use deletion policies to preserve resources when you delete a stack (set `DeletionPolicy` to `Retain`). Some resources can instead have a snapshot or backup taken before you delete the resource (set `DeletionPolicy` to `Snapshot`). The following resource types support snapshots:

- `AWS::EC2::Volume`
- `AWS::ElastiCache::CacheCluster`
- `AWS::ElastiCache::ReplicationGroup`
- `AWS::RDS::DBInstance`
- `AWS::RDS::DBCluster`
- `AWS::Redshift::Cluster`

The following template example creates an Amazon S3 bucket with a deletion policy set to retain the bucket when you delete the stack.

When you create a stack, you can submit a template from a local file or via a URL that points to an object in Amazon S3. If you submit the template as a local file, it uploads to Amazon S3 on your behalf.

Two key benefits of AWS CloudFormation are that your infrastructure is repeatable and that it is versionable.

A change set is a description of the changes that will occur to a stack should you submit the template and/or parameter updates. When you process stack updates, the template snippets you reference in any transforms pull from their Amazon S3 locations. If a snippet updates without your knowledge, the updated snippet will import into the template. Use a change set where there is a potential for data loss.

If values input into a template cannot be determined until the stack or change set is actually created, intrinsic functions resolve this by adding dynamic functionality into AWS CloudFormation templates. Condition functions are intrinsic functions to create resources or set resource properties that evaluate true or false conditions.

AWS CloudFormation Designer is a web-based graphical interface to design and deploy AWS CloudFormation templates. You can create connections to make relationships between resources that automatically update dependencies between them.

AWS CloudFormation uses custom resource providers to handle the provisioning and configuration of custom resources with AWS Lambda functions or Amazon SNS topics. You must provide a service token along with any optional input parameters. *The service token acts as a reference to where custom resource requests are sent*. This can be an AWS Lambda function or Amazon SNS topic. Custom resources can provide outputs back to AWS CloudFormation, which are made accessible as properties of the custom resource.

Custom resources associated with AWS Lambda invoke functions whenever create, update, or delete actions are sent to the resource provider. This resource type is incredibly useful to reference other AWS services and resources that may not support AWS CloudFormation.

You can use custom resources associated with Amazon SNS for any long-running custom resource tasks, such as transcoding a large video file.

By default, AWS CloudFormation will track most dependencies between resources. *A resource can have a dependency on one or more other resources in a stack, in which case you create a resource relationship to control the order of resource creation, updates, and deletion.*

Whenever you perform an action on an AWS CloudFormation stack, the end result will bring the stack into one of several possible statuses. These actions can complete or fail. In the case of a failed event, you can roll back the release based on your update or deletion policies.

To update stacks, you can modify and resubmit the same template or create a change set; AWS CloudFormation will parse it for changes (add, modify, or delete) and apply the modifications to the resources. You use the AWS CloudFormation `UpdatePolicy` to determine how to respond to changes. When you delete a stack, by default all underlying stack resources also delete. *You use deletion policies to preserve resources when you delete a stack*.

AWS Auto Scaling group update policies enforce the behavior that will occur when an update is performed on an AWS Auto Scaling group. This depends on the type of change you make and whether you configure the AWS Auto Scaling scheduled actions. You can replace the entire AWS Auto Scaling group or only instances inside it. When you delete a stack, all underlying stack resources are deleted. You can apply the `DeletionPolicy` to resources in the stack to modify their behavior when the stack deletes.

Managing Throughput Capacity Automatically with AWS Auto Scaling

Many database workloads are cyclical in nature or are difficult to predict in advance. In a social networking application, where most of the users are active during daytime hours, the database must be able to handle the daytime activity. But there is no need for the same levels of throughput at night. If a new mobile gaming app is experiencing rapid adoption and becomes too popular, the app could exceed the available database resources, resulting in slow performance and unhappy customers. These situations often require manual intervention to scale database resources up or down in response to varying usage levels.

DynamoDB uses the *AWS Application Auto Scaling* service to adjust provisioned throughput capacity dynamically 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, Application Auto Scaling decreases the throughput so that you do not pay for unused provisioned capacity.

Application Auto Scaling does not scale down your provisioned capacity if the consumed capacity of your table becomes zero. To scale down capacity manually, perform one of the following actions:

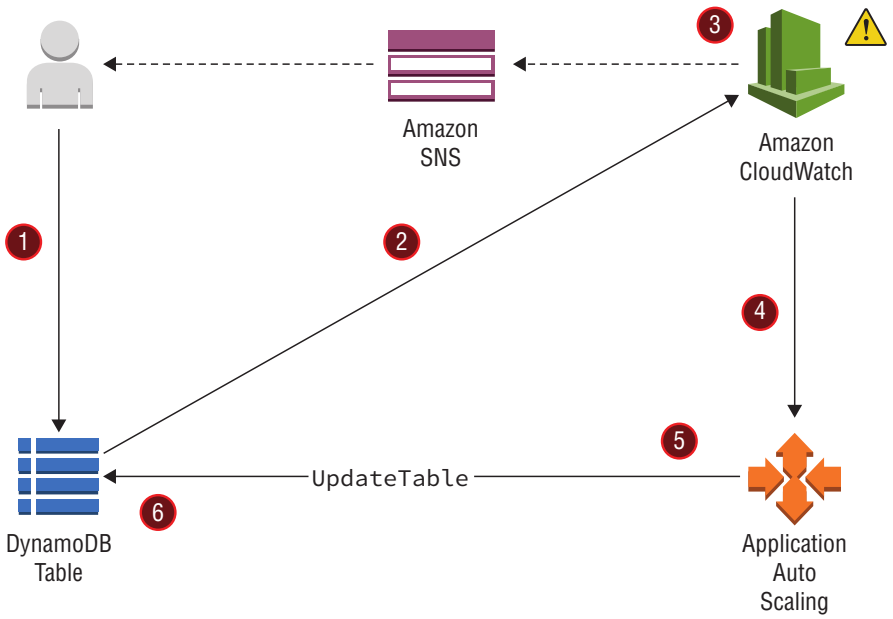
- Send requests to the table until automatic scaling scales down to the minimum capacity.
- Change the policy and reduce the maximum provisioned capacity to the same size as the minimum provisioned capacity.

With Application Auto Scaling, you can create a scaling policy for a table or a global secondary index. The scaling policy specifies whether you want to scale read capacity or write capacity (or both), and the minimum and maximum provisioned capacity unit settings for the table or index.

The scaling policy also contains a target utilization that is the percentage of consumed provisioned throughput at a point in time. Application Auto Scaling uses a target tracking algorithm to adjust the provisioned throughput of the table (or index) upward or downward in response to actual workloads so that the actual capacity utilization remains at or near your target utilization.

DynamoDB automatic scaling also supports global secondary indexes. Every global secondary index has its own provisioned throughput capacity, separate from that of its base table. When you create a scaling policy for a global secondary index, Application Auto Scaling adjusts the provisioned throughput settings for the index to ensure that its actual utilization stays at or near your desired utilization ratio, as shown in Figure 14.8.

FIGURE 14.8 DynamoDB Auto Scaling



How DynamoDB Auto Scaling Works

The steps in Figure 14.8 summarize the automatic scaling process:

1. Create an Application Auto Scaling policy for your DynamoDB table.
2. DynamoDB publishes consumed capacity metrics to Amazon CloudWatch.
3. If the table's consumed capacity exceeds your target utilization (or falls below the target) for a specific length of time, CloudWatch triggers an alarm. You can view the alarm on the AWS Management Console and receive notifications using Amazon Simple Notification Service (Amazon SNS).
4. The CloudWatch alarm invokes Application Auto Scaling to evaluate your scaling policy.
5. Application Auto Scaling issues an `UpdateTable` request to adjust your table's provisioned throughput.
6. DynamoDB processes the `UpdateTable` request, increasing or decreasing the table's provisioned throughput capacity dynamically so that it approaches your target utilization.

DynamoDB automatic scaling modifies provisioned throughput settings only when the actual workload stays elevated or depressed for a sustained period of several minutes. The

Application Auto Scaling target tracking algorithm seeks to keep the target utilization at or near your chosen value over the long term. Sudden, short-duration spikes of activity are accommodated by the table's built-in burst capacity.

Burst Capacity

DynamoDB provides some flexibility in your per-partition throughput provisioning by providing *burst capacity*. Whenever you are not fully using a partition's throughput, Amazon DynamoDB reserves a portion of that unused capacity for later bursts of throughput to handle usage spikes.

DynamoDB currently retains up to 5 minutes (300 seconds) of unused read and write capacity. During an occasional burst of read or write activity, these extra capacity units can be consumed quickly—even faster than the per-second provisioned throughput capacity that you have defined for your table. However, do not rely on burst capacity being available at all times, as DynamoDB can also consume burst capacity for background maintenance and other tasks without prior notice.

To enable DynamoDB automatic scaling, you create a scaling policy. This *scaling policy* specifies the table or global secondary index that you want to manage, which capacity type to manage (read or write capacity), the upper and lower boundaries for the provisioned throughput settings, and your target utilization.

When you create a scaling policy, Application Auto Scaling creates a pair of CloudWatch alarms on your behalf. Each pair represents your upper and lower boundaries for provisioned throughput settings. These CloudWatch alarms are triggered when the table's actual utilization deviates from your target utilization for a sustained period of time.

When one of the CloudWatch alarms is triggered, Amazon SNS sends you a notification (if you have enabled it). The CloudWatch alarm then invokes Application Auto Scaling, which notifies DynamoDB to adjust the table's provisioned capacity upward or downward, as appropriate.

Considerations for DynamoDB Auto Scaling

Before you begin using DynamoDB automatic scaling, be aware of the following:

- DynamoDB automatic scaling can increase read capacity or write capacity as often as necessary in accordance with your automatic scaling policy. All DynamoDB limits remain in effect.
- DynamoDB automatic scaling does not prevent you from manually modifying provisioned throughput settings. These manual adjustments do not affect any existing CloudWatch alarms that are related to DynamoDB automatic scaling.
- If you enable DynamoDB automatic scaling for a table that has one or more global secondary indexes, AWS highly recommends that you also apply automatic scaling uniformly to those indexes. You can apply this by choosing *Apply same settings* `to global secondary indexes` in the AWS Management Console.

TABLE 15.2 Amazon EC2 Metrics

Namespace	`AWS/EC2`
Dimensions	▪ `InstanceId`: identifier of a particular Amazon EC2 instance ▪ `InstanceType`: type of Amazon EC2 instance, such as `t2.micro`, `m4.large`
Key metrics	▪ `CPUUtilization`: percentage of vCPU utilization on the instance ▪ `DiskReadOps`, `DiskWriteOps`: number of operations per second on attached disk ▪ `DiskReadBytes`, `DiskWriteBytes`: volume of bytes to transfer on attached disk ▪ `NetworkIn`, `NetworkOut`: number of bytes sent or received by network interfaces ▪ `NetworkPacketsIn`, `NetworkPacketsOut`: number of packets sent or received by network interfaces

Amazon EC2 does not report memory utilization to CloudWatch. This is because memory is allocated in full to an instance by the underlying host. Memory consumption is visible only to the guest operating system (OS) of the instance. However, you can report memory utilization to CloudWatch using the CloudWatch agent.

Table 15.3 describes the AWS Auto Scaling group metrics.

TABLE 15.3 AWS Auto Scaling Groups

Namespace	`AWS/AutoScaling`
Dimensions	`AutoScalingGroupName`: name of the Auto Scaling group
Key metrics	▪ `GroupMinSize`, `GroupMaxSize`, `GroupDesiredCapacity`: minimum, maximum, and desired size of the Auto Scaling group ▪ `GroupInServiceInstances`: number of instances up and running in the Auto Scaling group ▪ `GroupTotalInstances`: total number of instances in the Auto Scaling group, regardless of state

Relevant applications on Spot Instances should poll for the termination notice at 5-second intervals, which gives the application almost the entire 2 minutes to complete any needed processing before the instance is terminated and taken back by AWS.

Using Persistent Requests

You can set your request to remain open so that a new instance is launched in its place when the instance is interrupted. You can also have your Amazon EBS–backed instance stopped upon interruption and restarted when Spot has capacity at your preferred price.

Using Block Durations

You can also launch Spot Instances with a fixed duration (Spot blocks, 1–6 hours), which are not interrupted as the result of changes in the Spot price. Spot blocks can provide savings of up to 50 percent.

You submit a Spot Instance request and use the new `BlockDuration` parameter to specify the number of hours that you want your instances to run, along with the maximum price that you are willing to pay.

You can submit a request of this type by running the following command:

```
$ aws ec2 request-spot-instances \
      block-duration-minutes 360 \
      instance-count 2 \
      spot-price "0.25"
:
```

Alternatively, you can call the `RequestSpotInstances` function.

Minimizing the Impact of Interruptions

Because the Spot service can terminate Spot Instances without warning, it is important to build your applications in a way that allows you to make progress even if your application is interrupted. There are many ways to accomplish this, including the following:

Adding checkpoints Add checkpoints that save your work periodically, for example, to an Amazon EBS volume. Another approach is to launch your instances from Amazon EBS–backed AMI.

Splitting up the work By using Amazon Simple Queue Service (Amazon SQS), you can queue up work increments and track work that has already been done.

Using AWS Auto Scaling

Using AWS Auto Scaling, you can scale workloads in your architecture. It automatically increases the number of resources during the demand spikes to maintain performance and decreases capacity when demand lulls to reduce cost. AWS Auto Scaling is well-suited for

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,

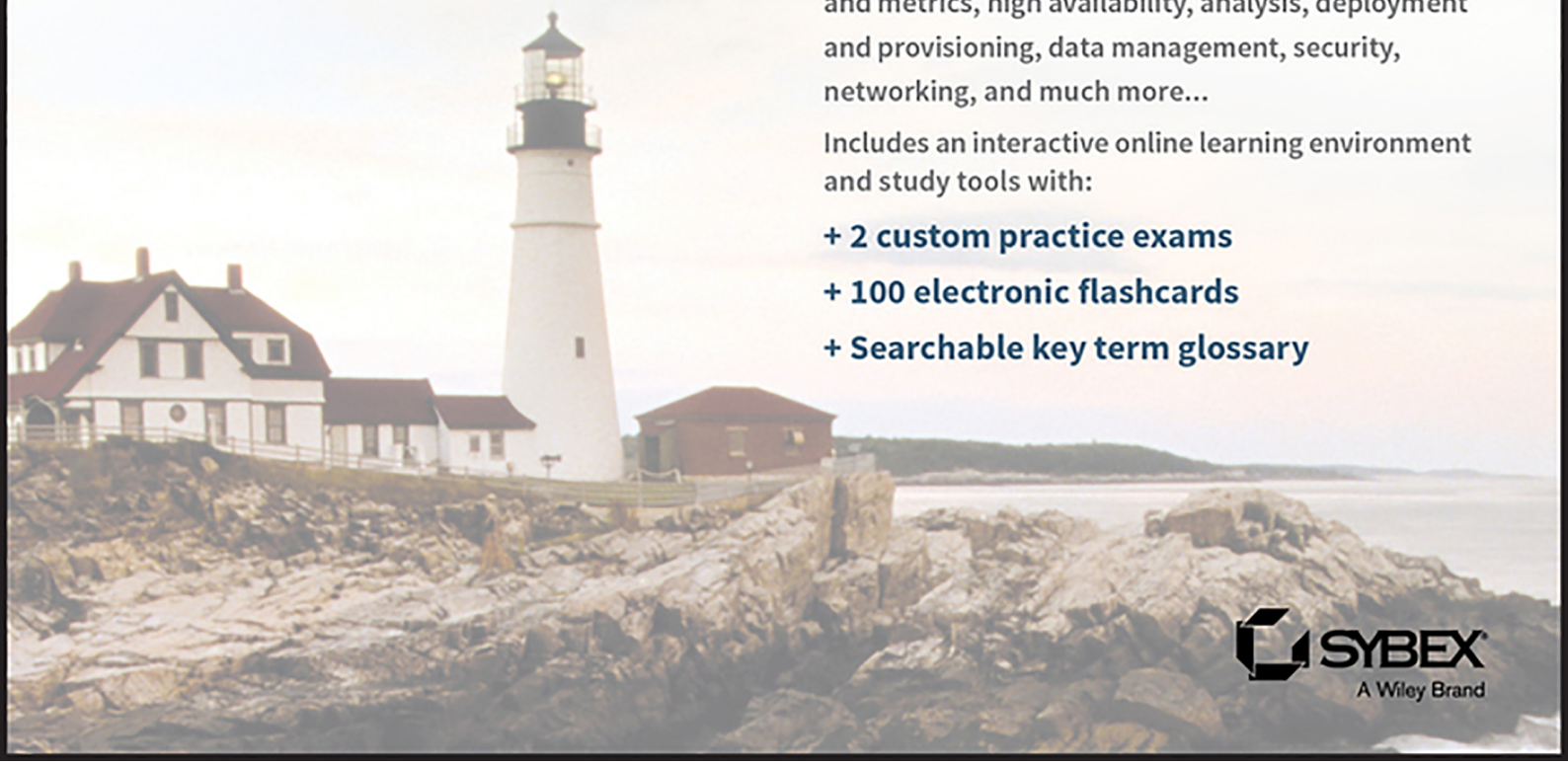
amazon web services

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

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.

Because you pay for the resources that are used to run your AWS Lambda function, the purpose of a timeout is to prevent your AWS Lambda function from running indefinitely. You can choose a value ranging between 1 and 300 seconds (5 minutes). When the specified timeout is reached, the AWS Lambda function is terminated.

The IAM role controls the level of access that the specific AWS Lambda function has. This is important when using AWS Lambda to access other AWS resources.

The handler refers to the method in your code where AWS Lambda begins execution. AWS Lambda passes any event information that triggered the invocation (further information on invocation and invocation types follows) as a parameter to the handler method.

When an AWS Lambda function is invoked, AWS Lambda launches a container based on your configuration specifications. Because it is launching a container, there may be some latency the first time that the AWS Lambda function is invoked. AWS Lambda will maintain the container for some time after it has been invoked to speed up response time for subsequent invocations.

How AWS Lambda Functions Are Invoked

AWS Lambda functions can be invoked either synchronously or asynchronously. If you are using AWS Cloud services to invoke AWS Lambda functions, those AWS Cloud services control whether the function is invoked synchronously or asynchronously. For example, if Amazon S3 invokes an AWS Lambda function, it does so asynchronously. In contrast, Amazon Cognito invokes an AWS Lambda function synchronously.

AWS Lambda functions are invoked by the following:

Push event model Some event sources can publish events to AWS Lambda and directly invoke your AWS Lambda function. The push model applies to Amazon S3, Amazon SNS, Amazon Cognito, Amazon Echo, and user applications, where each event triggers the AWS Lambda function.

Pull event model Some event sources publish events, but AWS Lambda must poll the event source and invoke your AWS Lambda function when events occur. This model applies when AWS Lambda is used with streaming event sources such as Amazon Kinesis and Amazon DynamoDB Streams. For example, AWS Lambda polls your Amazon Kinesis stream, or Amazon DynamoDB Stream, and it invokes your AWS Lambda function when it detects new records on the stream.

Direct invocation model This invocation type causes AWS Lambda to execute the function synchronously and returns the response immediately to the calling application. This invocation type is available for custom applications.

One of the recommendations for writing code for AWS Lambda is to make sure that the code is stateless (that is, it does not make reference to any underlying infrastructure).

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

Deploying Applications

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

Configuration Management

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

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

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

Tagging

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

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

Custom Variables

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

Baking Amazon Machine Images (AMIs)

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

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

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

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

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

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

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

Logging

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

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

Instance Profiles

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

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

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

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

Scalability Capabilities

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

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

Monitoring Resources

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

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

Continuous Deployment

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

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

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

Deployment Methods

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

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

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

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

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

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

In-Place Upgrade

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

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

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

Replacement Upgrade

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

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

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

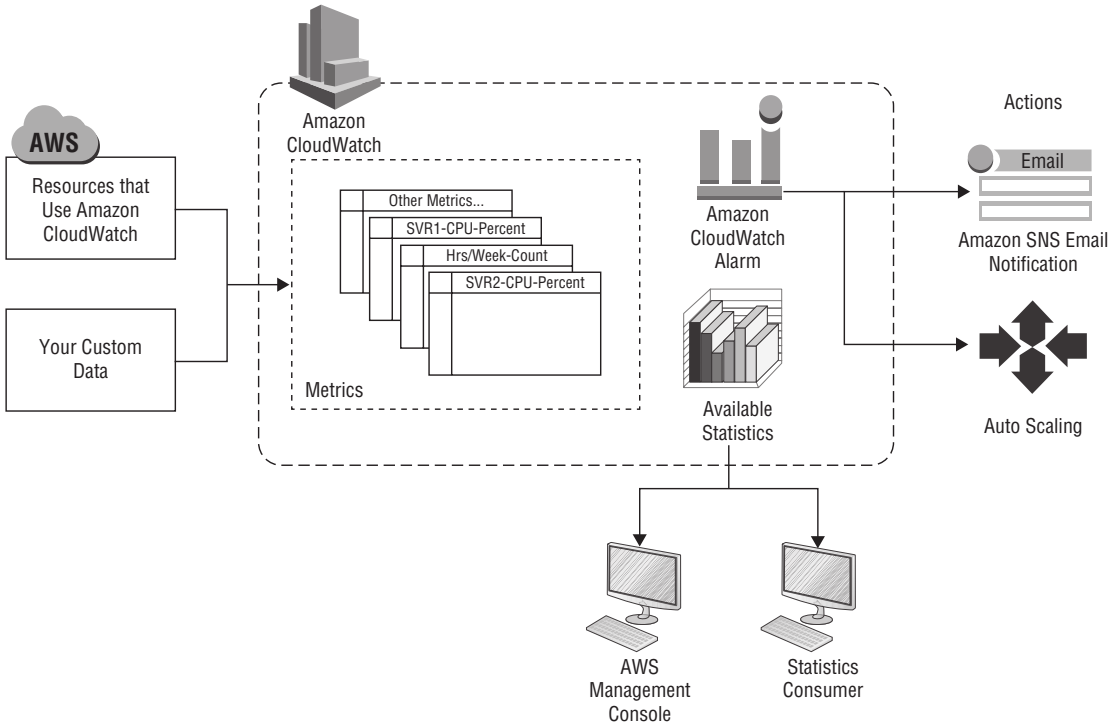
Blue/Green Deployments

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

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

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

FIGURE 9.1 Amazon CloudWatch integration



Amazon CloudWatch can be accessed using the following methods:

- Amazon CloudWatch console: `https://console.aws.amazon.com/cloudwatch/`
- The AWS CLI
- The Amazon CloudWatch API
- AWS SDKs

Amazon CloudWatch Services

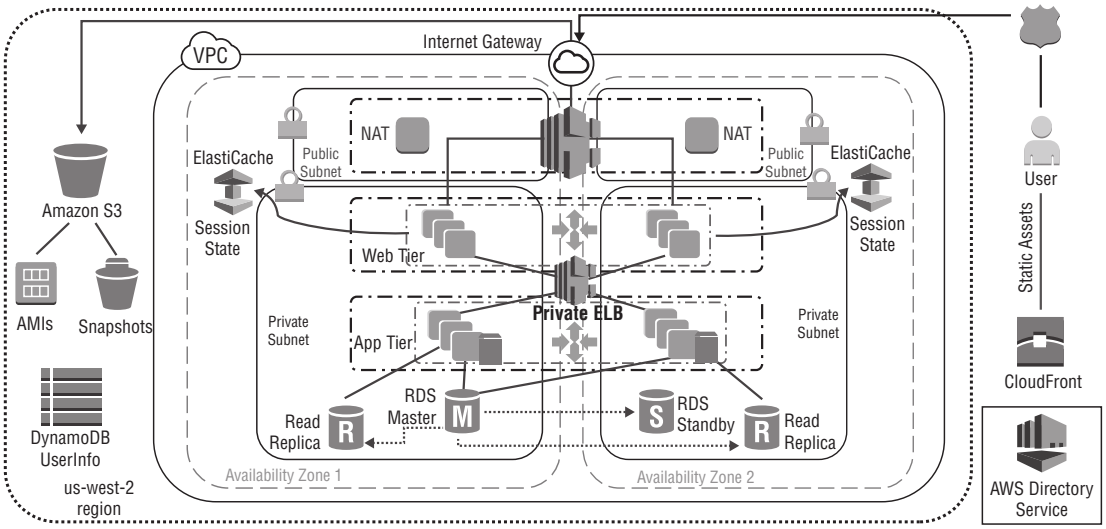
The following services are used in conjunction with Amazon CloudWatch:

Amazon SNS Amazon SNS coordinates and manages the delivery or sending of messages to subscribing endpoints or clients. Use Amazon SNS with Amazon CloudWatch to send messages when an alarm threshold has been reached.

Auto Scaling Auto Scaling automatically launches or terminates Amazon EC2 instances based on user-defined policies, health status checks, and schedules. Use an Amazon CloudWatch Alarm with Auto Scaling to scale Amazon EC2 instances in or out based on demand.

AWS CloudTrail AWS CloudTrail can monitor calls made to the Amazon CloudWatch API for an account. It includes calls made by the AWS Management Console, AWS CLI, and other

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.

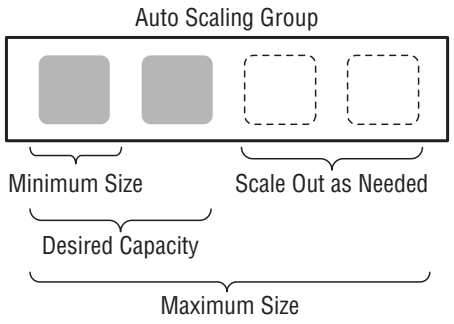
Auto Scaling

Auto Scaling is a web service designed to launch or terminate Amazon EC2 instances automatically based on user-defined policies, schedules, and health checks. Application Auto Scaling automatically scales supported AWS Cloud services with an experience similar to Auto Scaling for Amazon EC2 resources. Application Auto Scaling works with Amazon EC2 Container Service (Amazon ECS) and will not be covered in this guide.

Auto Scaling helps to ensure that you have the correct number of Amazon EC2 instances available to handle the load for your application. You create collections of Amazon EC2 instances, called *Auto Scaling groups*. You can specify the minimum number of instances in each Auto Scaling group, and Auto Scaling ensures that your group never goes below this size. Likewise, you can specify the maximum number of instances in each Auto Scaling group, and Auto Scaling ensures that your group never goes above this size. If you specify the desired capacity, either when you create the group or at any time thereafter, Auto Scaling ensures that your group has that many instances. If you specify scaling policies, then Auto Scaling can launch or terminate instances on demand as your application needs increase or decrease.

For example, the Auto Scaling group shown in Figure 10.6 has a minimum size of one instance, a desired capacity of two instances, and a maximum size of four instances. The scaling policies that you define adjust the number of instances, within your minimum and maximum number of instances, based on the criteria that you specify.

FIGURE 10.6 Auto Scaling group



Auto Scaling is set in motion by Amazon CloudWatch. Amazon CloudWatch is covered in Chapter 9, "Monitoring and Metrics."

Auto Scaling Components

- Launch configuration
- Group
- Scaling policy (optional)
- Scheduled action (optional)

Launch configuration Launch configuration defines how Auto Scaling should launch your Amazon EC2 instances. Auto Scaling provides you with an option to create a new launch configuration using the attributes from an existing Amazon EC2 instance. When you use this option, Auto Scaling copies the attributes from the specified instance into a template from which you can launch one or more Auto Scaling groups.

Auto Scaling group Your Auto Scaling group uses a launch configuration to launch Amazon EC2 instances. You create the launch configuration by providing information about the image that you want Auto Scaling to use to launch Amazon EC2 instances. The information can be the image ID, instance type, key pairs, security groups, and block device mapping.

Auto Scaling policy An Auto Scaling group uses a combination of policies and alarms to determine when the specified conditions for launching and terminating instances are met. An alarm is an object that watches over a single metric (for example, the average CPU utilization of your Amazon EC2 instances in an Auto Scaling group) over a time period that you specify. When the value of the metric breaches the thresholds that you define over a number of time periods that you specify, the alarm performs one or more actions. An action can be sending messages to Auto Scaling. A *policy* is a set of instructions for Auto Scaling that tells the service how to respond to alarm messages. These alarms are defined in Amazon CloudWatch (refer to Chapter 9 for more details).

Scheduled action Scheduled action is scaling based on a schedule, allowing you to scale your application in response to predictable load changes. To configure your Auto Scaling group to scale based on a schedule, you need to create scheduled actions. A *scheduled action* tells Auto Scaling to perform a scaling action at a certain time in the future. To create a scheduled scaling action, you specify the start time at which you want the scaling action to take effect, along with the new minimum, maximum, and desired size you want for that group at that time. At the specified time, Auto Scaling will update the group to set the new values for minimum, maximum, and desired sizes, as specified by your scaling action.

Session State Management

In order to scale out or back in, your application needs to be stateless. Consider storing session-related information off of the instance. Amazon DynamoDB or Amazon ElastiCache can be used for session state management. For more information on Amazon ElastiCache and Amazon DynamoDB, refer to Chapter 7.

Amazon Elastic Compute Cloud Auto Recovery

Auto Recovery is an Amazon EC2 feature that is designed to increase instance availability. It allows you to recover supported instances automatically when a system impairment is detected.

JON BONSO AND KENNETH SAMONTE

AWS CERTIFIED

DEVOPS ENGINEER PROFESSIONAL

Tutorials Dojo
Study Guide and Cheat Sheets

	to AWS SNS or a Lambda function to send results to Slack channel
Need to run an AWS CodePipeline every day for updating the development progress status	Create CloudWatch Events rule to run on schedule every day and set a target to the AWS CodePipeline ARN
Automate deployment of a Lambda function and test for only 10% of traffic for 10 minutes before allowing 100% traffic flow.	Use CodeDeploy and select deployment configuration CodeDeployDefault.LambdaCanary10Percent10Minutes
Deployment of Elastic Beanstalk application with absolutely no downtime. The solution must maintain full compute capacity during deployment to avoid service degradation.	Choose the “Rolling with additional Batch” deployment policy in Elastic Beanstalk
Deployment of Elastic Beanstalk application where the new version must not be mixed with the current version.	Choose the “Immutable deployments” deployment policy in Elastic Beanstalk
Configuration Management and Infrastructure-as-Code	
The resources on the parent CloudFormation stack needs to be referenced by other nested CloudFormation stacks	Use Export on the Output field of the main CloudFormation stack and use Fn::ImportValue function to import the value on the other stacks
On which part of the CloudFormation template should you define the artifact zip file on the S3 bucket?	The artifact file is defined on the AWS::Lambda::Function code resource block
Need to define the AWS Lambda function inline in the CloudFormation template	On the AWS::Lambda::Function code resource block, the inline function must be enclosed inside the ZipFile section.
Use CloudFormation to update Auto Scaling Group and only terminate the old instances when the newly launched instances become fully operational	Set AutoScalingReplacingUpdate : WillReplace property to TRUE to have CloudFormation retain the old ASG until the instances on the new ASG are healthy.
You need to scale-down the EC2 instances at night when there is low traffic using OpsWorks.	Create *Time-based* instances for automatic scaling of predictable workload.

Can't install an agent on on-premises servers but need to collect information for migration	Deploy the Agentless Discovery Connector VM on your on-premises data center to collect information.
Syntax for CloudFormation with an Amazon ECS cluster with ALB	Use the AWS::ECS::Service element for the ECS Cluster, AWS::ECS::TaskDefinition element for the ECS Task Definitions and the AWS::ElasticLoadBalancingV2::LoadBalancer element for the ALB.
Monitoring and Logging	
Need to centralize audit and collect configuration setting on all regions of multiple accounts	Setup an Aggregator on AWS Config.
Consolidate CloudTrail log files from multiple AWS accounts	Create a central S3 bucket with bucket policy to grant cross-account permission. Set this as destination bucket on the CloudTrail of the other AWS accounts.
Ensure that CloudTrail logs on the S3 bucket are protected and cannot be tampered with.	Enable Log File Validation on CloudTrail settings
Need to collect/investigate application logs from EC2 or on-premises server	Install CloudWatch Logs Agent to send the logs to CloudWatch Logs for storage and viewing.
Need to review logs from running ECS Fargate tasks	Enable awslogs log driver on the Task Definition and add the required logConfiguration parameter.
Need to run real-time analysis for collected application logs	Send logs to CloudWatch Logs, create a Lambda subscription filter, Elasticsearch subscription filter, or Kinesis stream filter.
Need to be automatically notified if you are reaching the limit of running EC2 instances or limit of Auto Scaling Groups	Track service limits with Trusted Advisor on CloudWatch Alarms using the ServiceLimitUsage metric.
Policies and Standards Automation	
Need to secure the buildspec.yml file which contains the AWS keys and database password stored in plaintext.	Store these values as encrypted parameter on SSM Parameter Store

Using default IAM policies for AWSCodeCommitPowerUser but must be limited to a specific repository only	Attach additional policy with Deny rule and custom condition if it does not match the specific repository or branch
You need to secure an S3 bucket by ensuring that only HTTPS requests are allowed for compliance purposes.	Create an S3 bucket policy that Deny if checks for condition aws:SecureTransport is **false**
Need to store a secret, database password, or variable, in the most cost-effective solution	Store the variable on SSM Parameter Store and enable encryption
Need to generate a secret password and have it rotated automatically at regular intervals	Store the secret on AWS Secrets Manager and enable key rotation.
Several team members, with designated roles, need to be granted permission to use AWS resources	Assign AWS managed policies on the IAM accounts such as, ReadOnlyAccess, AdministratorAccess, PowerUserAccess
Apply latest patches on EC2 and automatically create an AMI	Use Systems Manager automation to execute an Automation Document that installs OS patches and creates a new AMI.
Need to have a secure SSH connection to EC2 instances and have a record of all commands executed during the session	Install SSM Agent on EC2 and use SSM Session Manager for the SSH access. Send the session logs to S3 bucket or CloudWatch Logs for auditing and review.
Ensure that the managed EC2 instances have the correct application version and patches installed.	Use SSM Inventory to have a visibility of your managed instances and identify their current configurations.
Apply custom patch baseline from a custom repository and schedule patches to managed instances	Use SSM Patch Manager to define a custom patch baseline and schedule the application patches using SSM Maintenance Windows
Incident and Event Response	
Need to get a notification if somebody deletes files in your S3 bucket	Setup Amazon S3 Event Notifications to get notifications based on specified S3 events on a particular bucket.
Need to be notified when an RDS Multi-AZ failover happens	Setup Amazon RDS Event Notifications to detect specific events on RDS.

Get a notification if somebody uploaded IAM access keys on any public GitHub repositories	Create a CloudWatch Events rule for the AWS_RISK_CREDENTIALS_EXPOSED event from AWS Health Service. Use AWS Step Functions to automatically delete the IAM key.
Get notified on Slack when your EC2 instance is having an AWS-initiated maintenance event	Create a CloudWatch Events rule for the AWS Health Service to detect EC2 Events. Target a Lambda function that will send a notification to the Slack channel
Get notified of any AWS maintenance or events that may impact your EC2 or RDS instances	Create a CloudWatch Events rule for detecting any events on AWS Health Service and send a message to an SNS topic or invoke a Lambda function.
Monitor scaling events of your Amazon EC2 Auto Scaling Group such as launching or terminating an EC2 instance.	Use Amazon EventBridge or CloudWatch Events for monitoring the Auto Scaling Service and monitor the EC2 Instance-Launch Successful and EC2 Instance-Terminate Successful events.
View object-level actions of S3 buckets such as upload or deletion of object in CloudTrail	Set up Data events on your CloudTrail trail to record object-level API activity on your S3 buckets.
Execute a custom action if a specific CodePipeline stage has a FAILED status	Create CloudWatch Event rule to detect failed state on the CodePipeline service, and set a target to SNS topic for notification or invoke a Lambda function to perform custom action.
Automatically rollback a deployment in AWS CodeDeploy when the number of healthy instances is lower than the minimum requirement.	On CodeDeploy, create a deployment alarm that is integrated with Amazon CloudWatch. Track the MinimumHealthyHosts metric for the threshold of EC2 instances and trigger the rollback if the alarm is breached.
Need to complete QA testing before deploying a new version to the production environment	Add a Manual approval step on AWS CodePipeline, and instruct the QA team to approve the step before the pipeline can resume the deployment.
Get notified for OpsWorks auto-healing events	Create a CloudWatch Events rule for the OpsWorks Service to track the auto-healing events

High Availability, Fault Tolerance, and Disaster Recovery	
Need to ensure that both the application and the database are running in the event that one Availability Zone becomes unavailable.	Deploy your application on multiple Availability Zones and set up your Amazon RDS database to use Multi-AZ Deployments.
In the event of an AWS Region outage, you have to make sure that both your application and database will still be running to avoid any service outages.	Create a copy of your deployment on the backup AWS region. Set up an RDS Read-Replica on the backup region.
Automatically switch traffic to the backup region when your primary AWS region fails	Set up Route 53 Failover routing policy with health check enabled on your primary region endpoint.
Need to ensure the availability of a legacy application running on a single EC2 instance	Set up an Auto Scaling Group with MinSize=1 and MaxSize=1 configuration to set a fixed count and ensure that it will be replaced when the instance becomes unhealthy
Ensure that every EC2 instance on an Auto Scaling group downloads the latest code first before being attached to a load balancer	Create an Auto Scaling Lifecycle hook and configure the Pending:Wait hook with the action to download all necessary packages.
Ensure that all EC2 instances on an Auto Scaling group upload all log files in the S3 bucket before being terminated.	Use the Auto Scaling Lifecycle and configure the Terminating:Wait hook with the action to upload all logs to the S3 bucket.

Validate Your Knowledge

After your review, you should take some practice tests to measure your preparedness for the real exam. AWS offers a sample practice test for free which you can find here. You can also opt to buy the longer AWS sample practice test at aws.training, and use the discount coupon you received from any previously taken certification exams. Be aware though that the sample practice tests do not mimic the difficulty of the real DevOps Pro exam.

Therefore, we highly encourage using other mock exams such as our very own **AWS Certified DevOps Engineer Professional Practice Exam** course which contains high-quality questions with complete explanations on correct and incorrect answers, visual images and diagrams, YouTube videos as needed, and also contains reference links to official AWS documentation as well as our cheat sheets and study guides. You can also pair

AutoScalingReplacingUpdate vs AutoScalingRollingUpdate Policy

When using AWS CloudFormation to provision your Auto Scaling groups, you can control how CloudFormation handles updates for your Auto Scaling Group. You need to define the proper **UpdatePolicy** attribute for your ASG depending on your desired behavior during an update.

DevOps Exam Notes:

You can use **AWS::AutoScaling::AutoScalingGroup** resource type on CloudFormation if you want to create an AutoScaling group for your fleet of EC2 instances. Going into the exam, you will need to distinguish the difference between the **AutoScalingReplacingUpdate** and **AutoScalingRollingUpdate** UpdatePolicy attribute, which define how the instances on your group will be updated when you deploy a new revision of your application.

AutoScalingReplacingUpdate - will create a new auto scaling group with new launch configuration. This is more like an immutable type of deployment.

AutoScalingRollingUpdate - will replace the instances on the current auto-scaling group. You can control if instances will be replaced “all-at-once” or use a rolling update by batches. The default behavior is to delete instances first, before creating the new instances.

The **AutoScalingReplacingUpdate** policy specifies how AWS CloudFormation handles replacement updates for an Auto Scaling group. This policy enables you to specify whether AWS CloudFormation replaces an Auto Scaling group with a new one or replaces only the instances in the Auto Scaling group.

```
"UpdatePolicy" : {
  "AutoScalingReplacingUpdate" : {
    "WillReplace" : Boolean
  }
}
```

For example, you can set **AutoScalingReplacingUpdate WillReplace** property to TRUE to have CloudFormation retain the old ASG and the instances it contains. CloudFormation will wait for the successful creation of the new ASG and its instances before it deletes the old ASG. This is helpful when the update fails; CloudFormation can quickly rollback as it will only delete the new ASG. The current ASG and its instances are not affected during the deployment and rollback process.

The **AutoScalingRollingUpdate** policy specifies how AWS CloudFormation handles rolling updates for an Auto Scaling group. Rolling updates enable you to specify whether AWS CloudFormation updates instances that are in an Auto Scaling group in batches or all at once.

```
"UpdatePolicy" : {
  "AutoScalingRollingUpdate" : {
      "MaxBatchSize" : Integer,
      "MinInstancesInService" : Integer,
      "MinSuccessfulInstancesPercent" : Integer,
      "PauseTime" : String,
      "SuspendProcesses" : [ List of processes ],
      "WaitOnResourceSignals" : Boolean
  }
}
```

For example, **AutoScalingRollingUpdate** allows you to specify the **MaxBatchSize** property to set the maximum number of instances that AWS CloudFormation updates at any given time. Or use the **MinInstancesInService** property to ensure that there is a minimum number of instances that are in service while CloudFormation updates the old instances.

Sources:
https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-attribute-updatepolicy.html
https://aws.amazon.com/premiumsupport/knowledge-center/auto-scaling-group-rolling-updates/

Time-Based vs Load-Based Instance

With OpsWorks stacks, you can create a layer of auto scaling EC2 instances and a load balancer to host your application and service traffic from the public Internet. You can set up AWS OpsWorks Stacks to start or stop EC2 instances based on the configuration of your instances. You can use this to automatically increase/decrease the size of your EC2 cluster depending on when you need computing power. This automatic scaling is based on two instance types:

1. **Time-based instances -** allow a stack to handle loads that follow a predictable pattern by including instances that run only at certain times or on certain days. This is helpful in situations when you want to start some instances after 6PM to perform nightly backup tasks or stop some instances on weekends when traffic is lower.

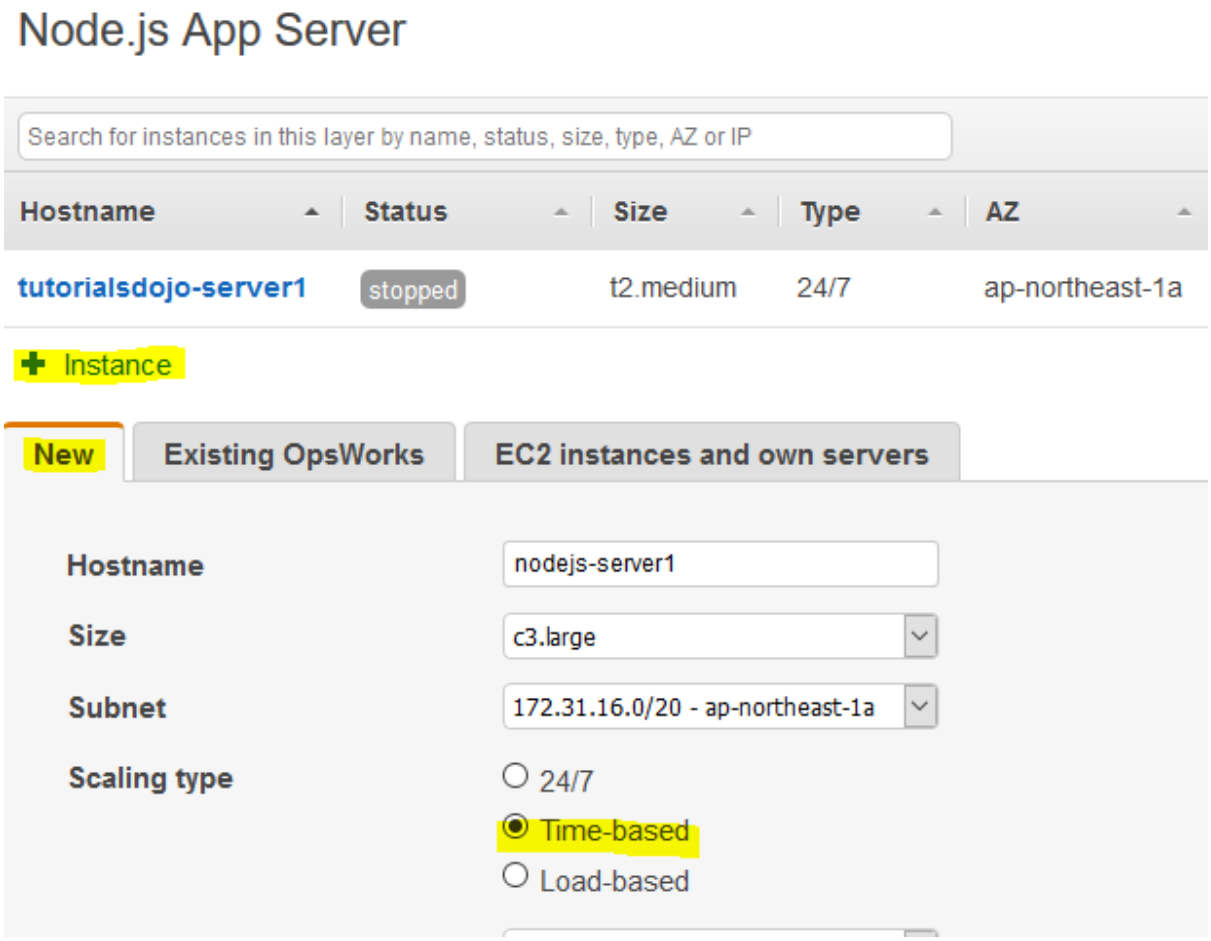
2. **Load-based instances -** allow a stack to handle variable loads by starting additional instances when traffic is high and stopping instances when traffic is low, based on any of several load metrics. This is helpful in situations where you want to start instances when the average CPU utilization exceeds 80% and stop instances when the average CPU load falls below 60%.

A common practice is to use all three instance types together, as follows:

- A set 24/7 instances to handle the base load. You typically just start these instances and let them run continuously.
- A set of time-based instances, which AWS OpsWorks Stacks starts and stops to handle predictable traffic variations.
- A set of load-based instances, which AWS OpsWorks Stacks starts and stops to handle unpredictable traffic variations.

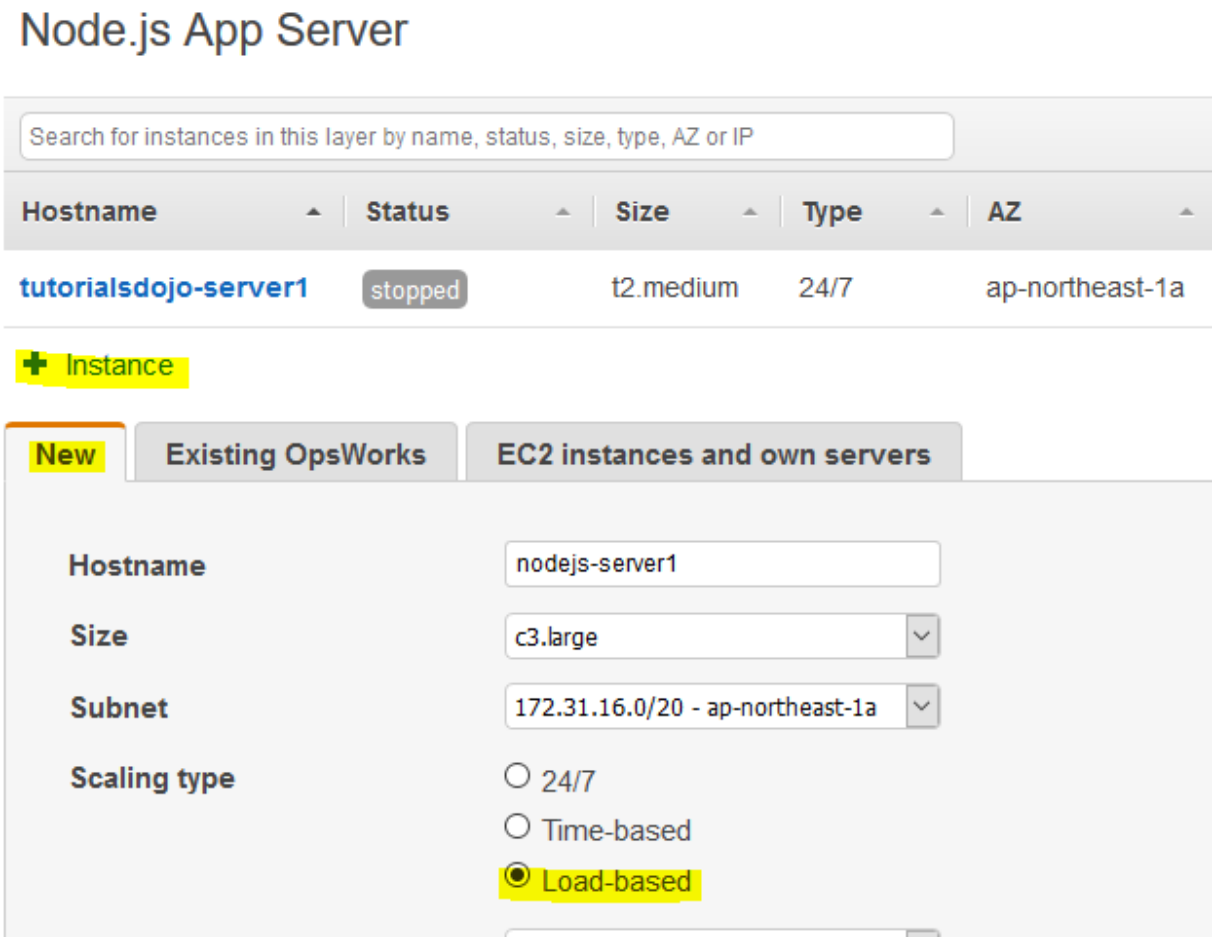
If you want to add a **time-based instance** to a layer follow these steps:

Click the Instances page, click + Instance to add an instance. On the New tab, click Advanced >> and then click time-based.



If you want to add a **load-based instance** to a layer:

On the Instances page, click +Instance to add an instance. Click Advanced >> and then click load-based.



Source:
https://docs.aws.amazon.com/opsworks/latest/userguide/workinginstances-autoscaling.html

CloudFormation Template for ECS, Auto Scaling and ALB

Amazon Elastic Container Service (ECS) allows you to manage and run Docker containers on clusters of EC2 instances. You can also configure your ECS to use Fargate launch type which eliminates the need to manage EC2 instances.

With CloudFormation, you can define your ECS clusters and tasks definitions to easily deploy your containers. For high availability of your Docker containers, ECS clusters are usually configured with an auto scaling group behind an application load balancer. These resources can also be declared on your CloudFormation template.

DevOps Exam Notes:

Going on to the exam, be sure to remember the syntax needed to declare your ECS cluster, Auto Scaling group, and application load balancer. The **AWS::ECS::Service** resource creates an ECS cluster and the **AWS::ECS::TaskDefinition** resource creates a task definition for your container. The **AWS::ElasticLoadBalancingV2::LoadBalance**r resource creates an application load balancer and the **AWS::AutoScaling::AutoScalingGroup** resource creates an EC2 auto scaling group.

AWS provides an example template which you can use to deploy a web application in an Amazon ECS container with auto scaling and application load balancer. Here's a snippet of the template with the core resources:

```
{
  "AWSTemplateFormatVersion":"2010-09-09",
  "Resources":{
      "ECSCluster":{
      "Type":"AWS::ECS::Cluster"

      },

……

      "taskdefinition":{
      "Type":"AWS::ECS::TaskDefinition",

      "Properties":{

……

      "ECSALB":{
      "Type":"AWS::ElasticLoadBalancingV2::LoadBalancer",
```

```
        "Properties":{

……

        "ECSAutoScalingGroup":{
        "Type":"AWS::AutoScaling::AutoScalingGroup",
        "Properties":{
        "VPCZoneIdentifier":{
              "Ref":"SubnetId"

        },
```

Sources:
https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/quickref-ecs.html
https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-ecs-service.html
https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-ecs-service-loadbalancers.html

Monitoring Service Limits with Trusted Advisor

AWS Trusted Advisor provides you real-time guidance to help you provision your resources following AWS best practices. It provides recommendations on 5 categories – Cost Optimization, Performance, Security, Fault Tolerance and Service Limits.

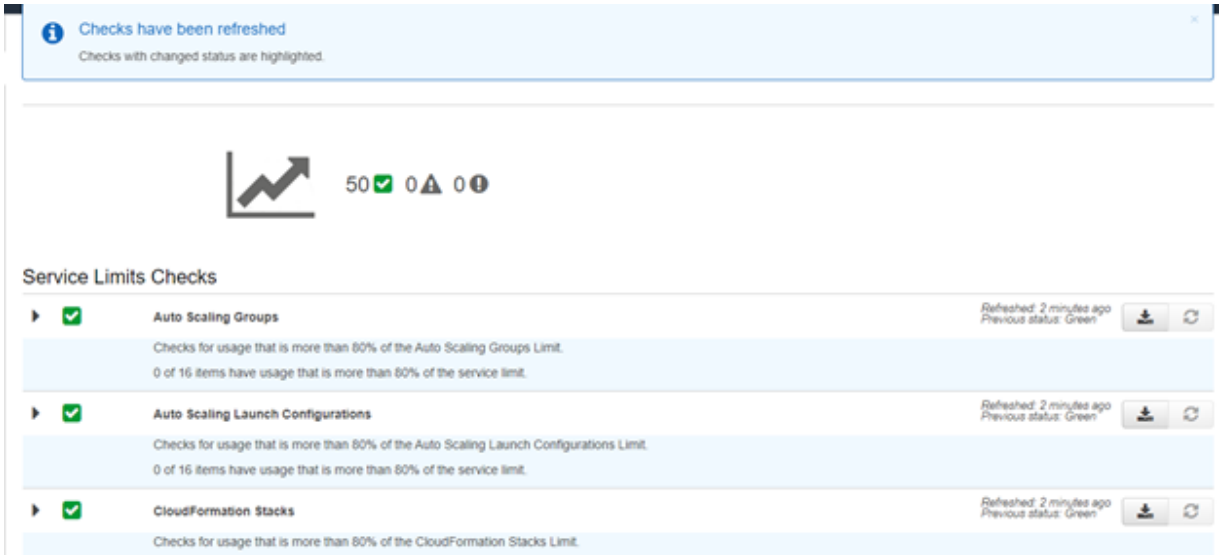
With Trusted Advisor's Service Limit Dashboard, you can view, refresh, and export utilization and limit data on a per-limit basis. These metrics are published on AWS CloudWatch in which you can create custom alarms based on percentage of service utilization against limits, understand the number of resources by each check, or view time-aggregate views of check results across service categories.

DevOps Exam Notes:

You need to understand that service limits are important when managing your AWS resources. In the exam you can be given a scenario in which you have several Auto Scaling groups in your AWS account and you need to make sure that you are not reaching the service limit when you perform your blue/green deployments for your application. You can track service limits with Trusted Advisor and CloudWatch Alarms. The ServiceLimitUsage metric on CloudWatch Alarms is only visible for Business and Enterprise support customers.

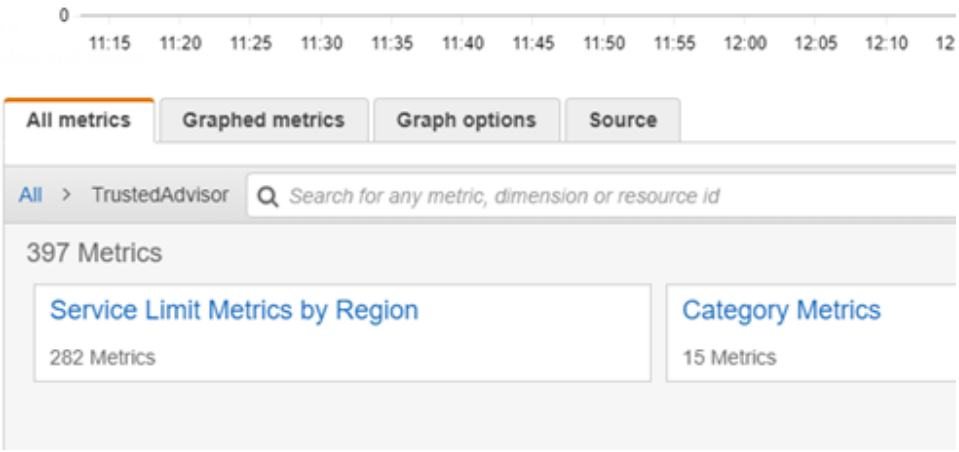
Here's how you can create a CloudWatch Alarm to detect if you are nearing your auto scaling service limit and send a notification so you can request for a service limit increase to AWS support.

1. First, head over to AWS Trusted Advisor > Service Limits and click the refresh button. This will refresh the service limit status for your account.

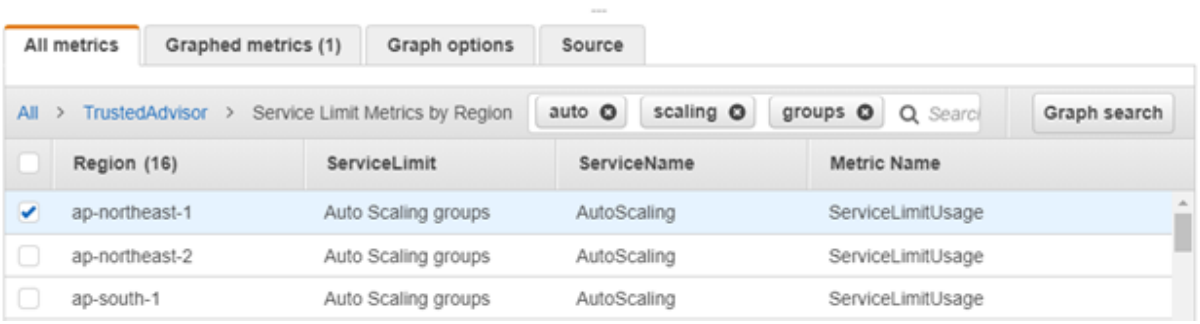


2. Go to CloudWatch > Alarms. Make sure that you are in the North Virginia region. Click the “Create Alarm” button and click “Select Metric”.

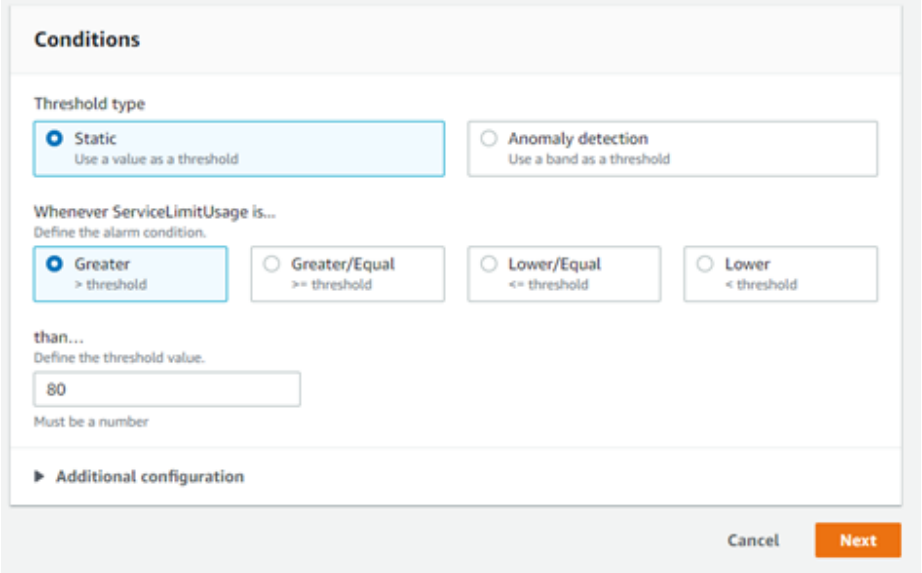
3. In the “All metrics” tab, click “Trusted Advisor” category and you will see “Service Limits by Region”.



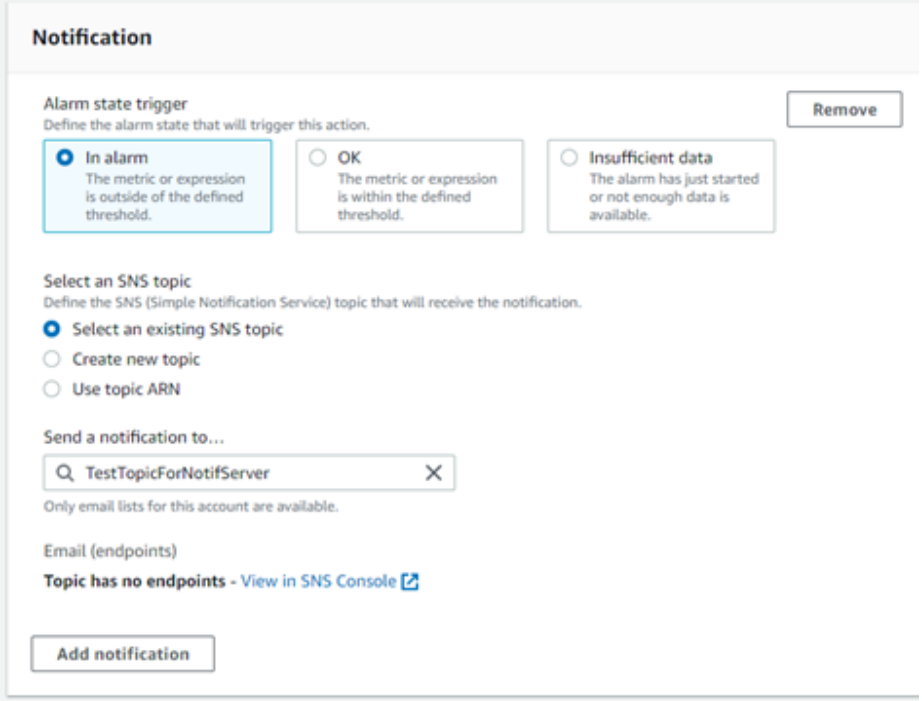
4. Search for Auto Scaling groups on your desired region and click Select Metric.



5. We can set the condition for this Alarm so that when your Auto Scaling group reaches 80 for that particular region, the alarm is triggered.



6. You can then configure an SNS topic to receive a notification for this alarm.



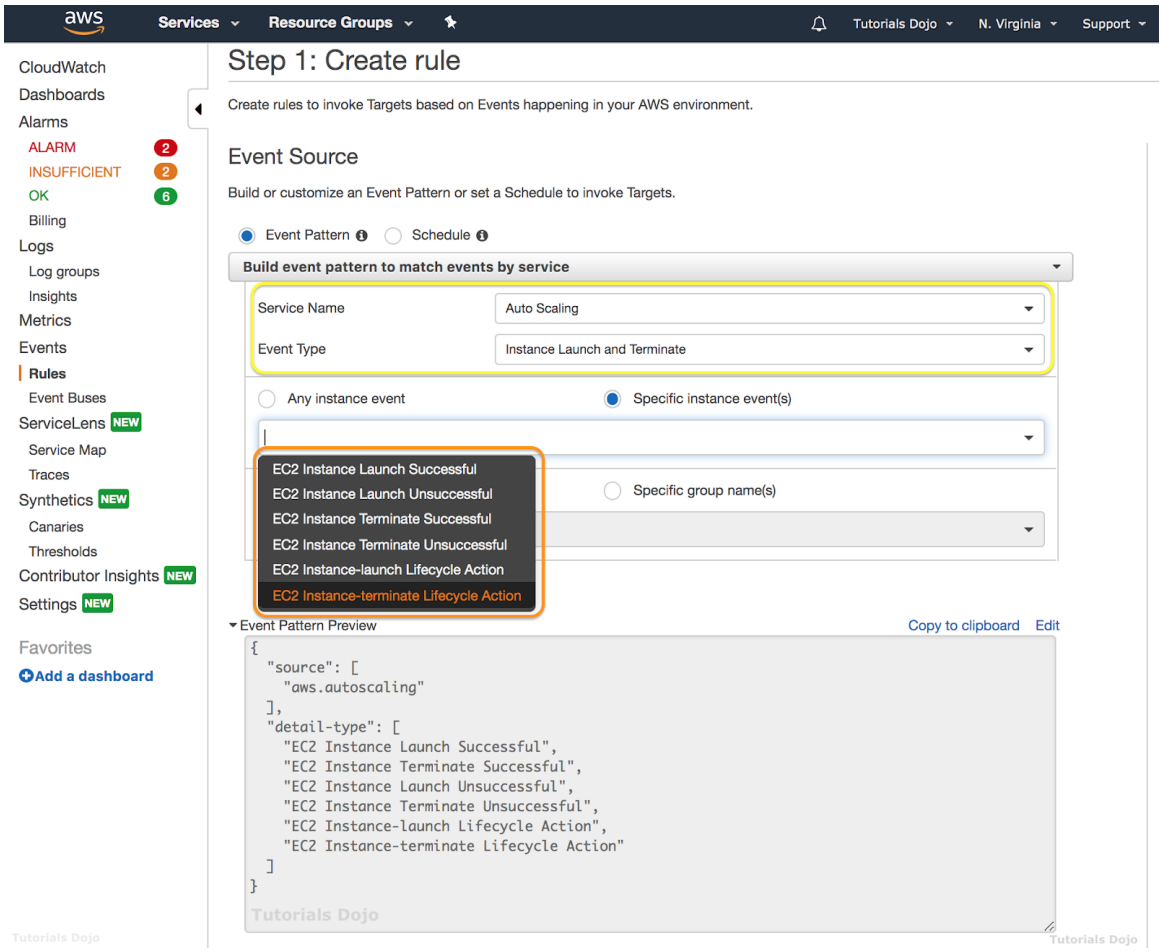
7. Click Next to Preview the alarm and click “Create Alarm” to create the alarm.

Sources:
https://aws.amazon.com/about-aws/whats-new/2017/11/aws-trusted-advisor-adds-service-limit-dashboard-and-cloudwatch-metrics/
https://aws.amazon.com/blogs/mt/monitoring-service-limits-with-trusted-advisor-and-amazon-cloudwatch/

Monitoring Amazon EC2 Auto Scaling Events

Auto Scaling provides fault tolerance, high availability, and cost management to your computing resources. It automatically scales in (terminates existing EC2 instances) if the demand is low and scales out (launch new EC2 instances) if the incoming traffic exceeds the specified threshold. AWS offers various ways of monitoring your fleet of Amazon EC2 instances as well responding to Auto Scaling events.

You can either use Amazon EventBridge or Amazon CloudWatch Events to track the Auto Scaling Events and run a corresponding custom action using AWS Lambda. Amazon EventBridge extends the capabilities of Amazon CloudWatch Events to integrate internal and external services. It allows you to track the changes of your Auto Scaling group in near real-time, including your custom applications, Software-as-a-Service (SaaS) partner apps, and other AWS services.



Amazon EventBridge or Amazon CloudWatch Events can be used to send events to the specified target when the following events occur:

- EC2 Instance-launch Lifecycle Action
- EC2 Instance Launch Successful
- EC2 Instance Launch Unsuccessful
- EC2 Instance-terminate Lifecycle Action
- EC2 Instance Terminate Successful
- EC2 Instance Terminate Unsuccessful

The **`EC2 Instance-launch Lifecycle Action`** is a scale-out event in which the Amazon EC2 Auto Scaling moved an EC2 instance to a **`Pending:Wait`** state due to a lifecycle hook. Conversely, the **`EC2 Instance-terminate Lifecycle Action`** is a scale-in event in which EC2 Auto Scaling updates an instance to a **`Terminating:Wait`** state.

Source:
https://docs.aws.amazon.com/autoscaling/ec2/userguide/cloud-watch-events.html

Auto Scaling Group with MinSize = 1 and MaxSize = 1

With Auto Scaling, you can set the number of EC2 instances that you need depending on the traffic of your application. However, there will be scenarios on the exam where you will need to set a fixed number of instances.

For example, you have a legacy application hosted on an EC2 instance. The application does not support working on a cluster of EC2 instances so it needs to run on a single EC2 instance and you need to make sure that the application is available and will heal itself even if that EC2 instance crashes.

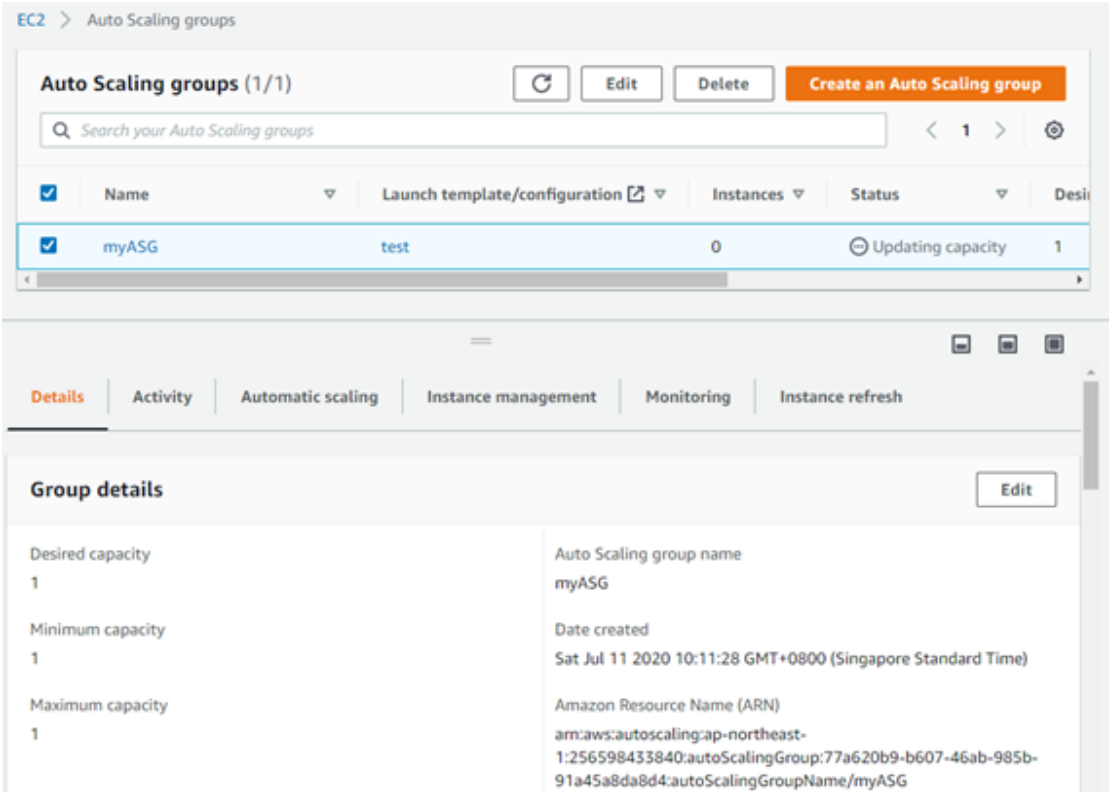
In this scenario, you will create an Auto Scaling Group using the EC2 AMI in the Launch configuration and set the size of the group to Min 1 and Max 1. This ensures that only instances of the application are running. Auto Scaling will perform health checks for the EC2 instances periodically. If the EC2 instance fails the health check, Auto Scaling will replace the instance.

Hence, it will always be available and self-healing. This makes your application fault-tolerant.

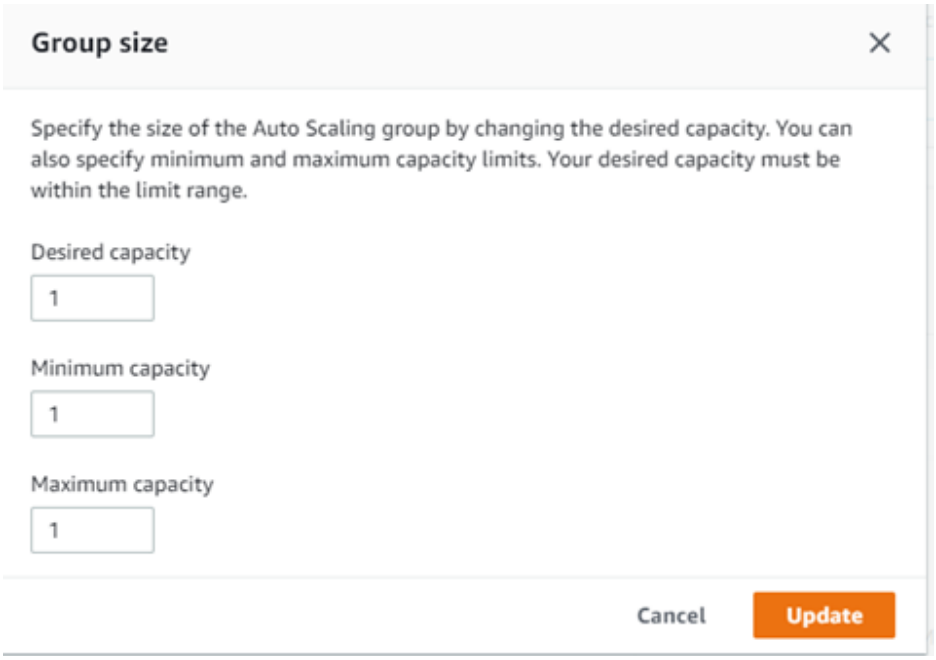
To set the MinSize and MaxSize of the Auto Scaling group:

1. Go to the EC2 Console page, on the left pane, choose Auto Scaling Groups.

2. Select the check box next to your Auto Scaling group. The bottom pane will show information on the selected Auto Scaling group.



3. On the Details tab, view or change the current settings for minimum, maximum, and desired capacity. Set the Desired capacity to 1, Minimum capacity to 1 and Maximum capacity to 1. Make sure that you don't have any Automatic Scaling policies configured for this group.



Sources:
https://docs.aws.amazon.com/autoscaling/ec2/userguide/as-maintain-instance-levels.html
https://docs.aws.amazon.com/autoscaling/ec2/userguide/asg-capacity-limits.html

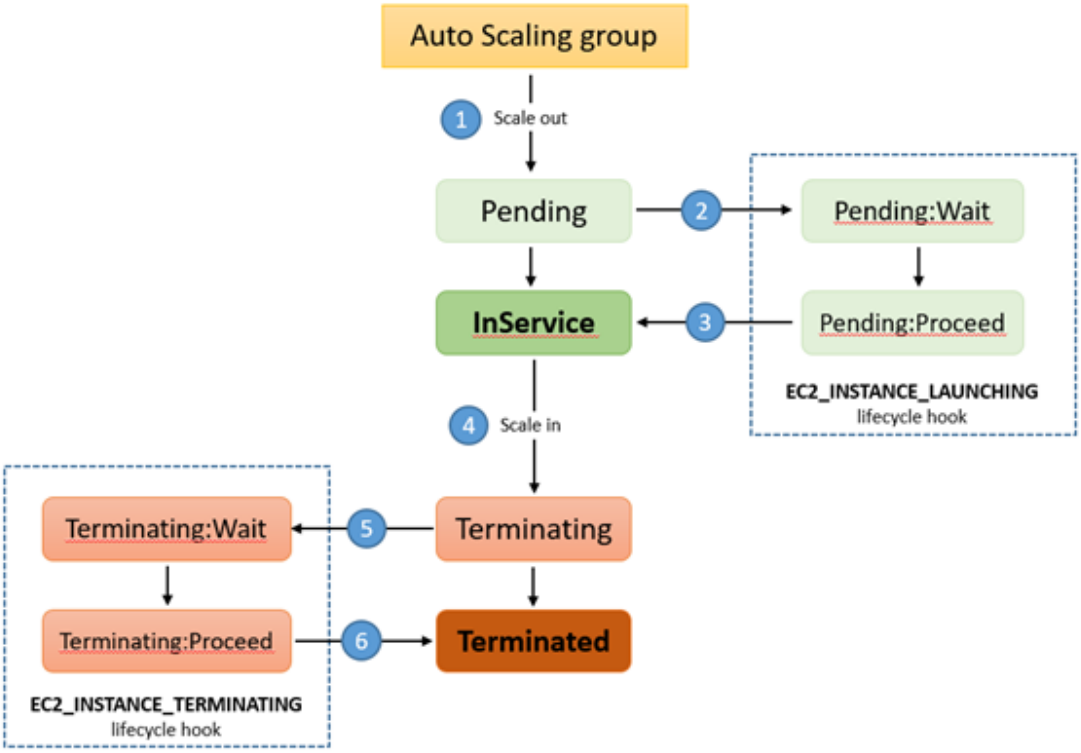
Auto Scaling Lifecycle Hooks

As your Auto Scaling group scales out or scales in your EC2 instances, you may want to perform custom actions before they start accepting traffic or before they get terminated. Auto Scaling Lifecycle Hooks allows you to perform custom actions during these stages. A lifecycle hook puts your EC2 instance into a wait state (`Pending:Wait` or `Terminating:Wait`) until your custom action has been performed or when the timeout period ends. The EC2 instance stays in a wait state for one hour by default, and then the Auto Scaling group resumes the launch or terminate process (`Pending:Proceed` or `Terminating:Proceed`).

For example, during the scale-out event of your ASG, you want to make sure that new EC2 instances download the latest code base from the repository and that your EC2 user data has completed before it starts accepting traffic. You can use the Pending:Wait hook. This way, the new instances will be fully ready and will quickly pass the load balancer health check when they are added as targets.

Another example: during the scale-in event of your ASG, suppose your instances upload data logs to S3 every minute. You may want to pause the instance termination for a certain amount of time to allow the EC2 to upload all data logs before it gets completely terminated.

The following diagram shows the transitions between the EC2 instance states with lifecycle hooks.



DevOps Exam Notes:

During the paused state (either launch or terminate), you can do more than just run custom scripts or wait for timeouts. CloudWatch Events receives the scaling action and you can define a CloudWatch Events Target to invoke a Lambda function that can perform custom actions, have it send a notification to your email via SNS, or trigger an SSM Run Command or SSM Automation to perform specific EC2 related tasks.

Sources:
https://docs.aws.amazon.com/autoscaling/ec2/userguide/lifecycle-hooks.html
https://docs.aws.amazon.com/cli/latest/reference/autoscaling/put-lifecycle-hook.html

AWS Auto Scaling

- Configure automatic scaling for the AWS resources quickly through a scaling plan that uses dynamic scaling and predictive scaling.
- Optimize for availability, for cost, or a balance of both.
- Scaling in means decreasing the size of a group while scaling out means increasing the size of a group.
- Useful for:
    - Cyclical traffic such as high use of resources during regular business hours and low use of resources overnight
    - On and off traffic patterns, such as batch processing, testing, or periodic analysis
    - Variable traffic patterns, such as software for marketing campaigns with periods of spiky growth
- **Features**
    - Launch or terminate EC2 instances in an Auto Scaling group.
    - Launch or terminate instances from an EC2 Spot Fleet request, or automatically replace instances that get interrupted for price or capacity reasons.
    - Adjust the ECS service desired count up or down in response to load variations.
    - Use Dynamic Scaling to add and remove capacity for resources to maintain resource utilization at the specified target value.
    - Use Predictive Scaling to forecast your future load demands by analyzing your historical records for a metric. It also allows you to schedule scaling actions that proactively add and remove resource capacity to reflect the load forecast, and control maximum capacity behavior. Only available for EC2 Auto Scaling groups.
    - You can suspend and resume any of your AWS Application Auto Scaling actions.
- **Amazon EC2 Auto Scaling**
    - Ensuring you have the correct number of EC2 instances available to handle your application load using Auto Scaling Groups.
    - An Auto Scaling group contains a collection of EC2 instances that share similar characteristics and are treated as a logical grouping for the purposes of instance scaling and management.
    - You specify the minimum, maximum and desired number of instances in each Auto Scaling group.
    - Key Components

Groups	Your EC2 instances are organized into groups so that they are treated as a logical unit for scaling and management. When you create a group, you can specify its minimum, maximum, and desired number of EC2 instances.

Launch configurations	Your group uses a launch configuration as a template for its EC2 instances. When you create a launch configuration, you can specify information such as the AMI ID, instance type, key pair, security groups, and block device mapping for your instances.
Scaling options	How to scale your Auto Scaling groups.

    - Scaling Options
        - Scale to maintain current instance levels at all times
        - Manual Scaling
        - Scale based on a schedule
        - Scale based on a demand
    - The cooldown period is a configurable setting that helps ensure to not launch or terminate additional instances before previous scaling activities take effect.
        - EC2 Auto Scaling supports cooldown periods when using simple scaling policies, but not when using target tracking policies, step scaling policies, or scheduled scaling.
    - Amazon EC2 Auto Scaling marks an instance as unhealthy if the instance is in a state other than running, the system status is impaired, or Elastic Load Balancing reports that the instance failed the health checks.
    - Termination of Instances
        - When you configure automatic scale in, you must decide which instances should terminate first and set up a termination policy. You can also use instance protection to prevent specific instances from being terminated during automatic scale in.
        - Default Termination Policy
        - Custom Termination Policies
            - OldestInstance - Terminate the oldest instance in the group.
            - NewestInstance - Terminate the newest instance in the group.
            - OldestLaunchConfiguration - Terminate instances that have the oldest launch configuration.
            - ClosestToNextInstanceHour - Terminate instances that are closest to the next billing hour.

  You can create launch templates that specifies instance configuration information when you launch EC2 instances, and allows you to have multiple versions of a template.
  A launch configuration is an instance configuration template that an Auto Scaling group uses to launch EC2 instances, and you specify information for the instances.
    - You can specify your launch configuration with multiple Auto Scaling groups.
    - You can only specify one launch configuration for an Auto Scaling group at a time, and you can't modify a launch configuration after you've created it.
- **Monitoring**
    - Health checks - identifies any instances that are unhealthy

        - Amazon EC2 status checks (default)
        - Elastic Load Balancing health checks
        - Custom health checks.
    - Auto scaling does not perform health checks on instances in the standby state. Standby state can be used for performing updates/changes/troubleshooting without health checks being performed or replacement instances being launched.
    - CloudWatch metrics - enables you to retrieve statistics about Auto Scaling-published data points as an ordered set of time-series data, known as metrics. You can use these metrics to verify that your system is performing as expected.
    - CloudWatch Events - Auto Scaling can submit events to CloudWatch Events when your Auto Scaling groups launch or terminate instances, or when a lifecycle action occurs.
    - SNS notifications - Auto Scaling can send Amazon SNS notifications when your Auto Scaling groups launch or terminate instances.
    - CloudTrail logs - enables you to keep track of the calls made to the Auto Scaling API by or on behalf of your AWS account, and stores the information in log files in an S3 bucket that you specify.

EC2 Instance Health Check vs ELB Health Check vs Auto Scaling and Custom Health Check

EC2 instance health check

- Amazon EC2 performs automated checks on every running EC2 instance to identify hardware and software issues.
- Status checks are performed every minute and each returns a pass or a fail status.
  - If all checks pass, the overall status of the instance is OK.
  - If one or more checks fail, the overall status is impaired.
- Status checks are built into EC2, so they cannot be disabled or deleted.
- You can create or delete alarms that are triggered based on the result of the status checks.
- There are two types of status checks
  - **System Status Checks**
    - These checks detect underlying problems with your instance that require AWS involvement to repair. When a system status check fails, you can choose to wait for AWS to fix the issue, or you can resolve it yourself.
  - **Instance Status Checks**
    - Monitor the software and network configuration of your individual instance. Amazon EC2 checks the health of an instance by sending an address resolution protocol (ARP) request to the ENI. These checks detect problems that require your involvement to repair.

Tutorials Dojo

Elastic Load Balancer (ELB) health check

- To discover the availability of your registered EC2 instances, a load balancer periodically sends pings, attempts connections, or sends requests to test the EC2 instances.
- 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.
- When configuring a health check, you would need to provide the following:
  - a specific port
  - protocol to use
    - HTTP/HTTPS health check succeeds if the instance returns a 200 response code within the health check interval.
    - A TCP health check succeeds if the TCP connection succeeds.
    - An SSL health check succeeds if the SSL handshake succeeds.
  - ping path
- ELB health checks do not support WebSockets.
- The load balancer routes requests only to the healthy instances. When an instance becomes impaired, the load balancer resumes routing requests to the instance only when it has been restored to a healthy state.
- The load balancer checks the health of the registered instances using either
  - the default health check configuration provided by Elastic Load Balancing or
  - a health check configuration that you configure (auto scaling or custom health checks for example).
- Network Load Balancers use active and passive health checks to determine whether a target is available to handle requests.
  - With **active health checks**, the load balancer periodically sends a request to each registered target to check its status. After each health check is completed, the load balancer node closes the connection that was established.
  - With **passive health checks**, the load balancer observes how targets respond to connections, which enables it to detect an unhealthy target before it is reported as unhealthy by active health checks. You cannot disable, configure, or monitor passive health checks.

Auto Scaling and Custom health checks

- All instances in your Auto Scaling group start in the healthy state. Instances are assumed to be healthy unless EC2 Auto Scaling receives notification that they are unhealthy. This notification can come from one or more of the following sources:
  - Amazon EC2 (default)
  - Elastic Load Balancing
  - A custom health check.
- After Amazon EC2 Auto Scaling marks an instance as unhealthy, it is scheduled for replacement. If you do not want instances to be replaced, you can suspend the health check process for any individual Auto Scaling group.
- If an instance is in any state other than running or if the system status is impaired, Amazon EC2 Auto Scaling considers the instance to be unhealthy and launches a replacement instance.
- If you attached a load balancer or target group to your Auto Scaling group, Amazon EC2 Auto Scaling determines the health status of the instances by checking both the EC2 status checks and the Elastic Load Balancing health checks.
- Amazon EC2 Auto Scaling waits until the health check grace period ends before checking the health status of the instance. Ensure that the health check grace period covers the expected startup time for your application.
- Health check grace period does not start until lifecycle hook actions are completed and the instance enters the InService state.
- With custom health checks, you can send an instance's health information directly from your system to Amazon EC2 Auto Scaling.