



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



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.

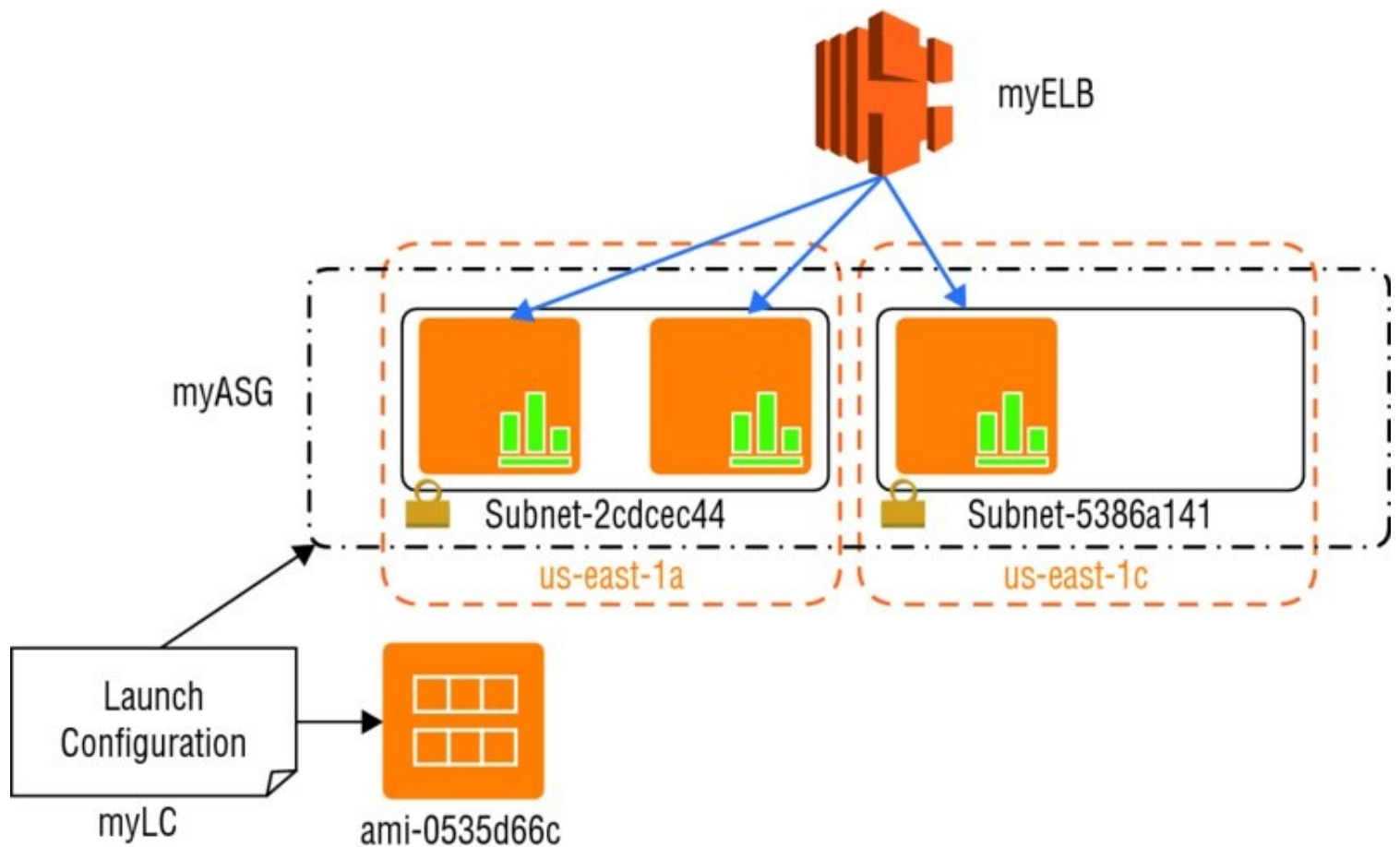


FIGURE 5.1 Auto Scaling group behind an Elastic Load Balancing load balancer

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 *CPU Load*, and the CloudWatch metric *CPU Utilization* 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).

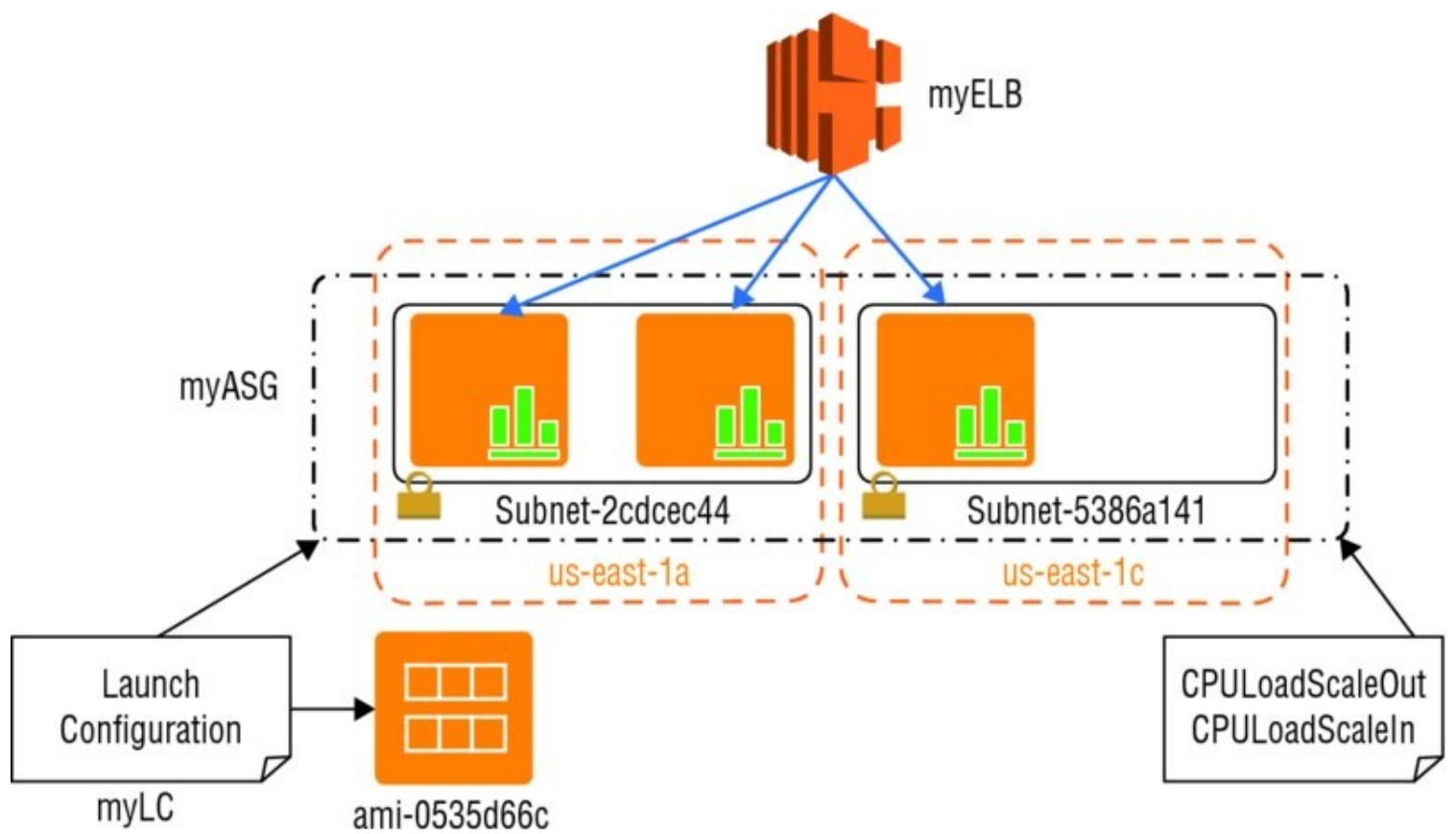


FIGURE 5.2 Auto Scaling group with policy

```
> 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**.

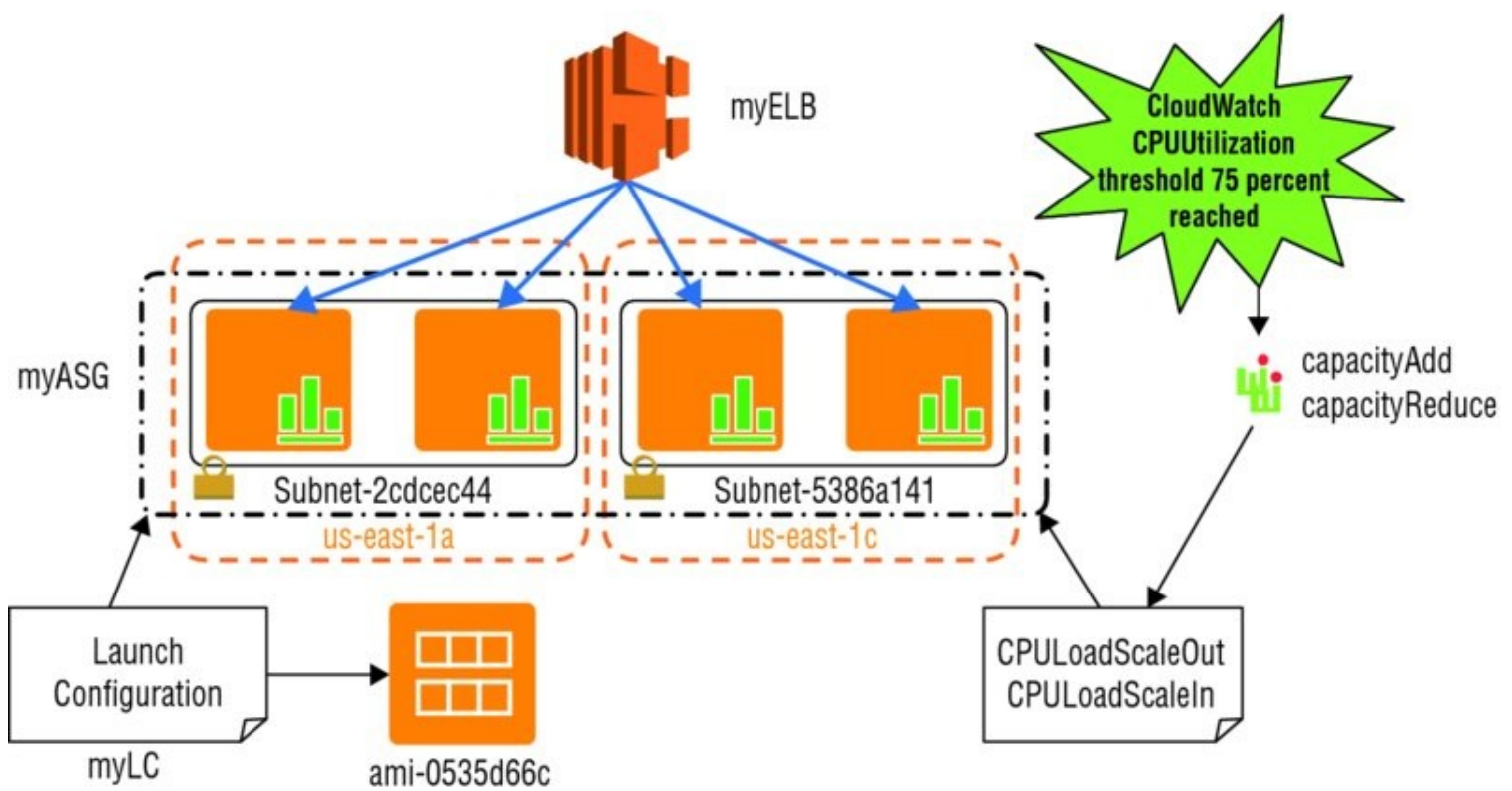


FIGURE 5.3 Amazon CloudWatch alarm triggering scaling out

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 Instance 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.

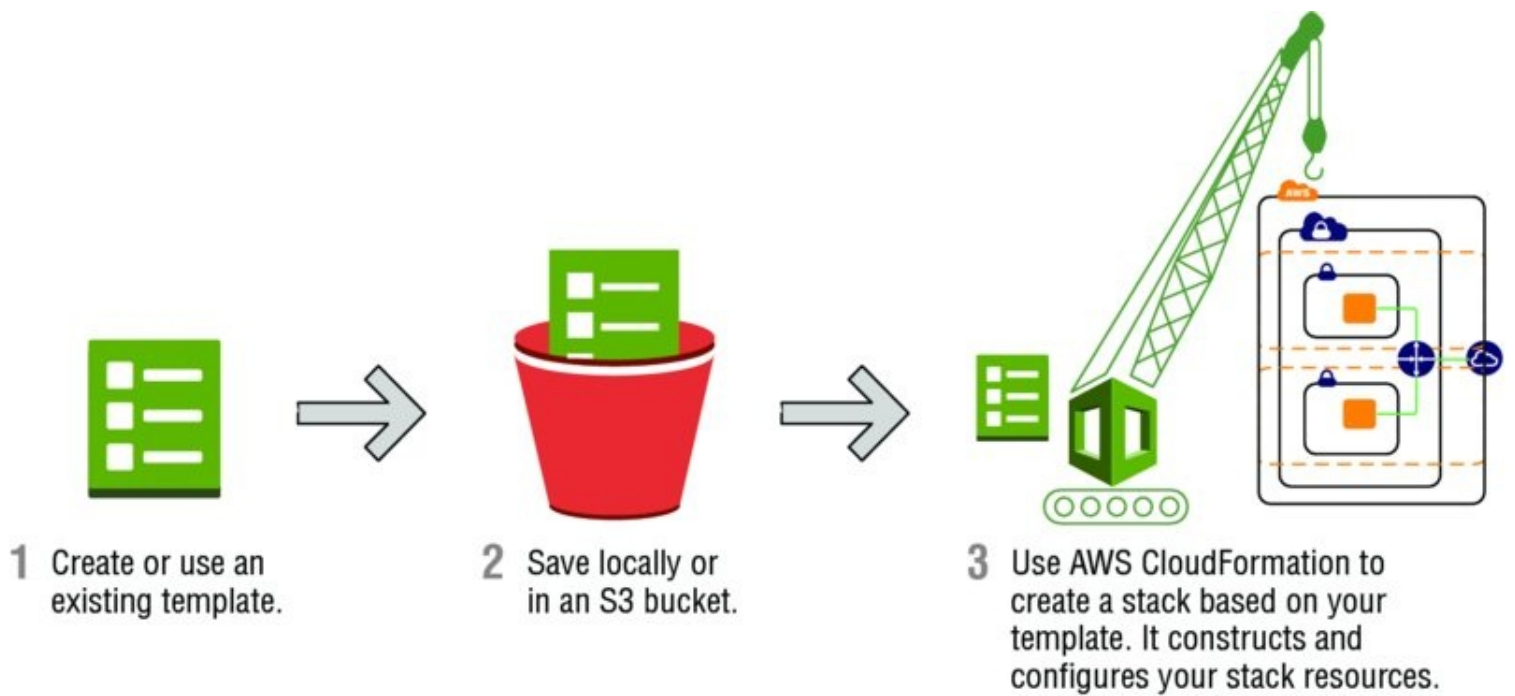


FIGURE 11.8 Creating a stack workflow

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.

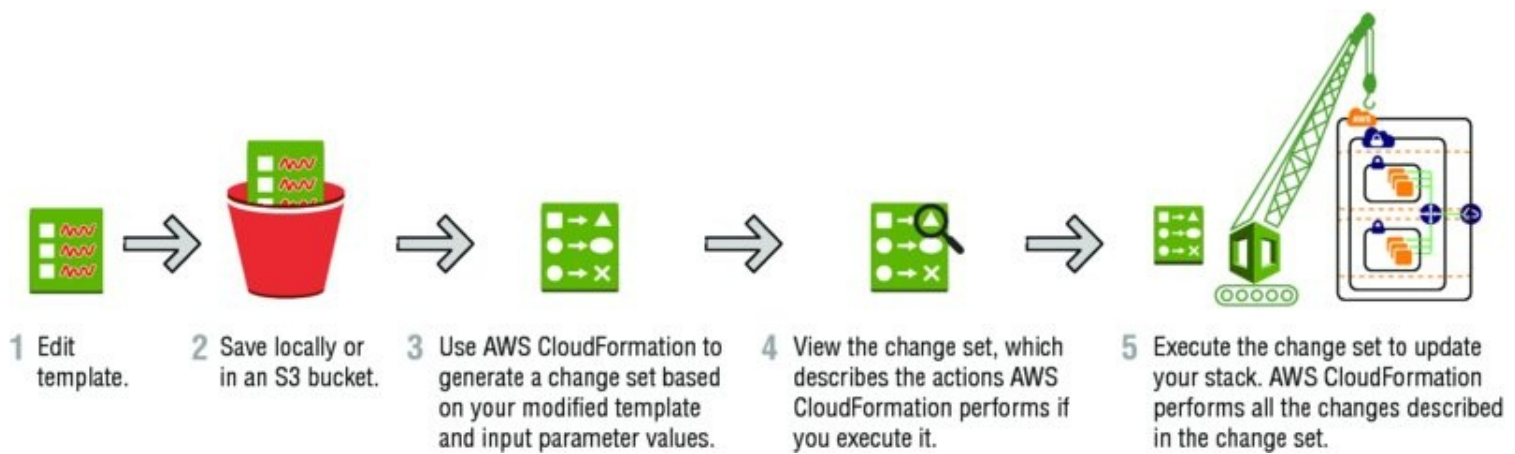



FIGURE 11.9 Updating a stack workflow

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



NOTE 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](#).