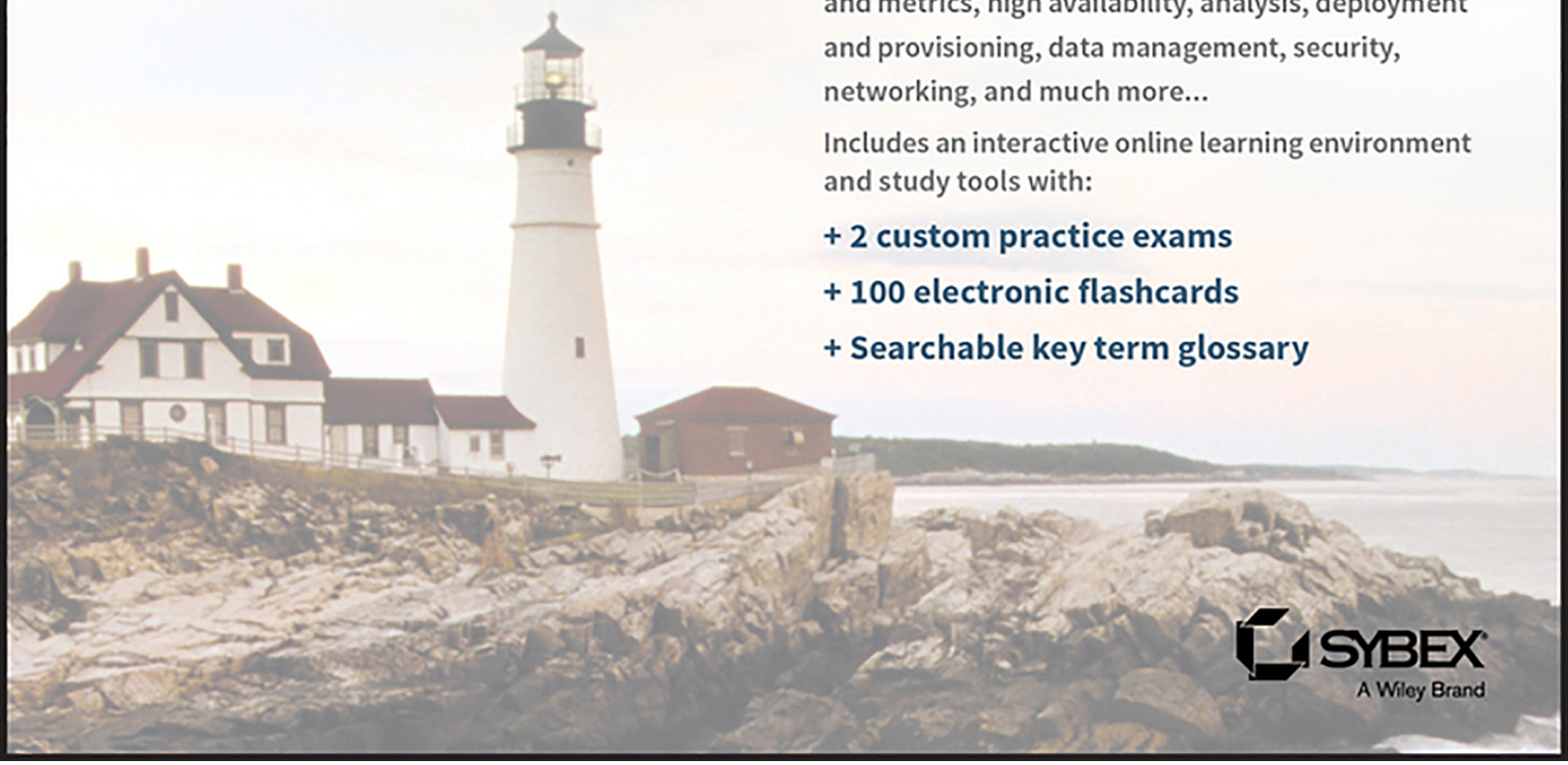
amazon web services

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

Languages Supported in AWS Elastic Beanstalk

AWS Elastic Beanstalk supports applications developed in the following languages:

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

Services that AWS Elastic Beanstalk Deploys

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

Implementation

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

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

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

Management

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

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

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

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

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

Making Changes in Your AWS Elastic Beanstalk Environment

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

Monitoring Your AWS Elastic Beanstalk Environment

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

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

Security

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

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

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

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

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

AWS Lambda

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

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

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

Implementation

Implementation for AWS Lambda involves the following steps:

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

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

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

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

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

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

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

In addition to the dashboard, it is also possible to subscribe to the RSS feed for any service.

For anyone experiencing a real-time operational issue with one of the AWS Cloud services currently reporting as being healthy, there is a Contact Us link at the top of the page to report an issue.

The AWS Service Health Dashboard is available at `http://status.aws.amazon.com/`.

All dates and times in the AWS Service Health Dashboard are in Pacific Time (PST/PDT).

AWS Personal Health Dashboard

The *AWS Personal Health Dashboard* provides alerts and remediation guidance when AWS is experiencing events that impact customers. While the AWS Service Health Dashboard displays the general status of AWS Cloud services, the AWS Personal Health Dashboard provides a personalized view into the performance and availability of the AWS Cloud services underlying provisioned AWS resources.

The dashboard displays relevant and timely information to help manage events in progress and provides proactive notification to help plan scheduled activities. Alerts are automatically triggered by changes in the health of AWS resources and provide event visibility and also guidance to help diagnose and resolve issues quickly.

Now let's take a deep dive into each of these services.

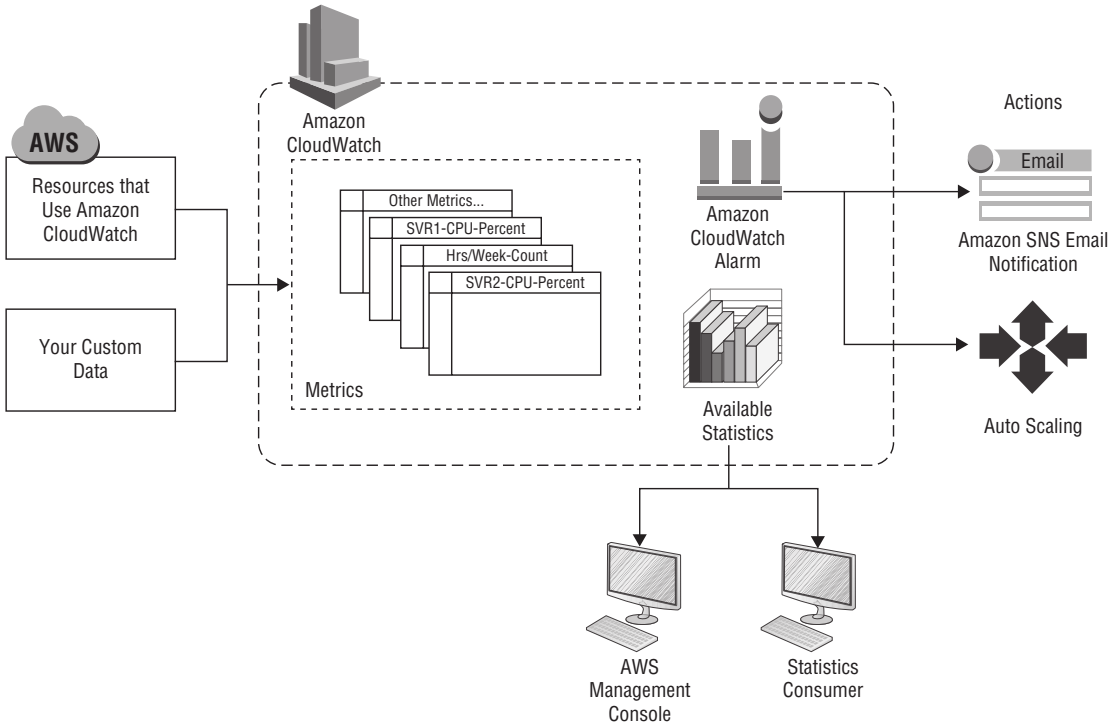
Amazon CloudWatch

Amazon CloudWatch monitors, in real time, AWS resources and applications running on AWS. Amazon CloudWatch is used to collect and track metrics, which are variables used to measure resources and applications. Amazon CloudWatch Alarms send notifications and can automatically make changes to the resources being monitored based on user-defined rules. Amazon CloudWatch is basically a metrics repository. An AWS product such as Amazon EC2 puts metrics into the repository and customers retrieve statistics based on those metrics. Additionally, custom metrics can be placed into Amazon CloudWatch for reporting and statistical analysis.

For example, it is possible to monitor the CPU usage and disk reads and writes of Amazon EC2 instances. With this information, it is possible to determine when additional instances should be launched to handle the increased load. Additionally, these new instances can be launched automatically before there is a problem, eliminating the need for human intervention. Conversely, monitoring data can be used to stop underutilized instances automatically in order to save money.

In addition to monitoring the built-in metrics that come with AWS, it is possible to create, monitor, and trigger actions using custom metrics. Amazon CloudWatch provides system-wide visibility into resource utilization, application performance, and operational health. Figure 9.1 illustrates how Amazon CloudWatch connects to both AWS and on-premises environments.

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

To have Amazon SNS deliver notifications to an Amazon SQS queue, a developer should subscribe to a topic specifying "Amazon SQS" as the transport and a valid Amazon SQS queue as the endpoint. In order to allow the Amazon SQS queue to receive notifications from Amazon SNS, the Amazon SQS queue owner must subscribe the Amazon SQS queue to the topic for Amazon SNS. If the user owns both the Amazon SNS topic being subscribed to and the Amazon SQS queue receiving the notifications, nothing further is required. Any message published to the topic will automatically be delivered to the specified Amazon SQS queue. If the owner of the Amazon SQS queue is not the owner of the topic, Amazon SNS will require an explicit confirmation to the subscription request.

Characteristics of Amazon SNS

- Each notification message contains a single published message.
- Message order is not guaranteed.
- A message cannot be deleted after it has been published.
- Amazon SNS delivery policy can be used to control retries in case of message delivery failure.
- Messages can contain up to 256 KB of text data, including XML, JSON, and unformatted text.

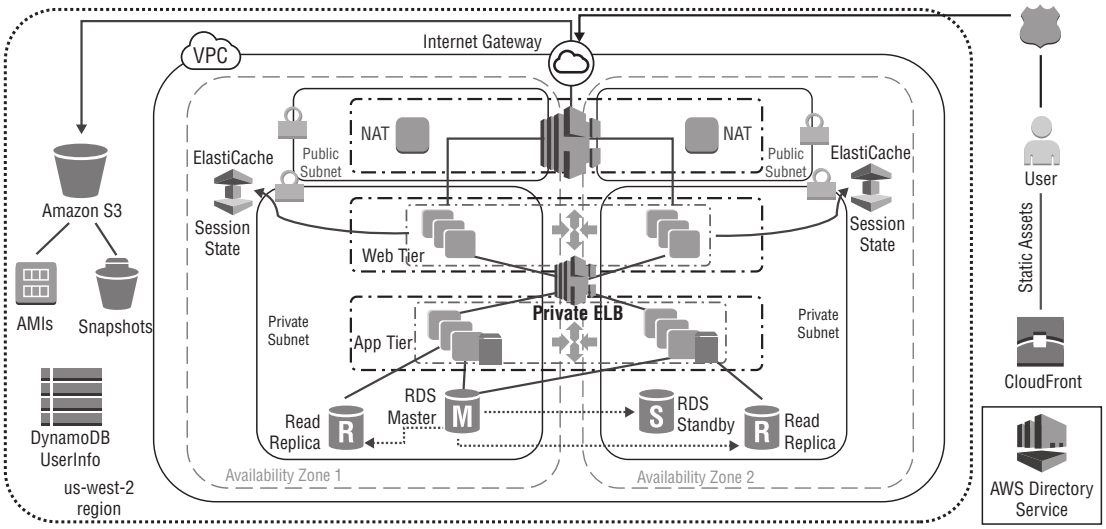
Highly Available Architectures

In Chapter 5, "Networking," you learned that the boundary of an Amazon Virtual Private Cloud (Amazon VPC) is at the region, and that regions comprise Availability Zones. In Chapter 4, "Compute," you learned how Amazon EC2 instances are bound to the Availability Zone. What happens if a natural disaster takes out your Amazon EC2 instance running in an affected Availability Zone? What if your application goes viral and starts taking on more traffic than expected? How do you continue to serve your users?

So far in this chapter, you have learned that services like Amazon SNS and Amazon SQS are highly available, fault-tolerant systems, which can be used to decouple your application. Chapter 7, "Databases," introduced Amazon DynamoDB and Amazon ElastiCache as NoSQL databases. Now we are going to put all these parts together to deliver a highly available architecture.

Figure 10.5 illustrates a highly available three-tier architecture (the concept of which was discussed in Chapter 2, "Working with AWS Cloud Services").

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.

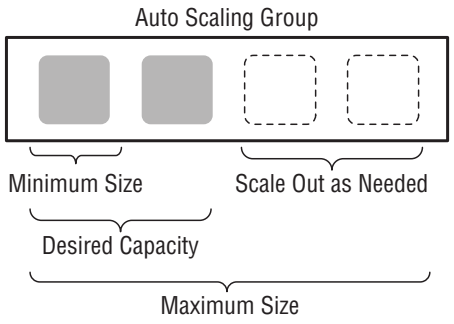
Auto Scaling

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.