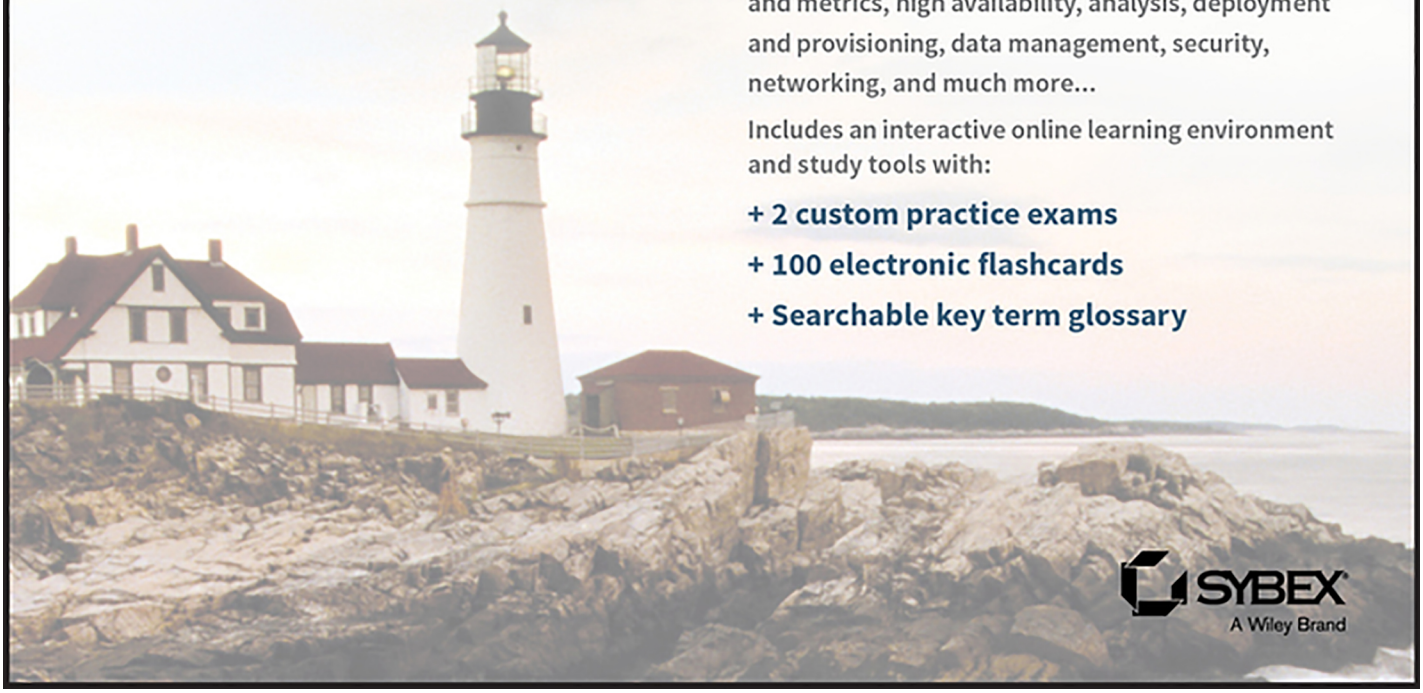
amazon web services

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

b. For Description, type a brief description of the option group. The description is used for display purposes.

c. For Engine, choose the DB engine that you want.

d. For Major Engine Version, choose the major version of the DB engine that you want.

5. To continue, choose Yes, Create. To cancel the operation instead, choose Cancel.

EXERCISE 7.2

Create an Amazon DynamoDB Table from the AWS CLI.

This exercise assumes that the AWS CLI has been installed. For readability, long commands in this section are broken into separate lines. The backslash character lets you copy and paste (or type) multiple lines into a Linux terminal. For Windows PowerShell, the backtick (`) does the same thing.

This exercise will create an Amazon DynamoDB table called `MusicCollection` that has the attributes Artist and Title.

Type the following command into the terminal window with the AWS CLI installed:

For Linux, use the following:

```
aws dynamodb create-table \
  --table-name MusicCollection \
  --attribute-definitions \
    AttributeName=Artist,AttributeType=S AttributeName=Title,AttributeType=S \
  --key-schema AttributeName=Artist,KeyType=HASH \
    AttributeName=Title,KeyType=RANGE \
  --provisioned-throughput ReadCapacityUnits=1,WriteCapacityUnits=1
```

For Microsoft Windows, use the following:

```
aws dynamodb create-table `
  --table-name MusicCollection `
  --attribute-definitions `
    AttributeName=Artist,AttributeType=S AttributeName=Title,AttributeType=S `
  --key-schema AttributeName=Artist,KeyType=HASH `
    AttributeName=Title,KeyType=RANGE `
  --provisioned-throughput ReadCapacityUnits=1,WriteCapacityUnits=1
```

EXERCISE 7.3

Add Items to the Amazon DynamoDB Table MusicCollection Using the AWS CLI.

Now that the table has been created, add data to it using the AWS CLI.

For Linux, use the following:

```
aws dynamodb put-item \
  --table-name MusicCollection \
  --item '{
    "Artist": {"S": "Mystical Father"},
    "Title": {"S": "Song for Elijah"} ,
    "AlbumTitle": {"S": "Lovely Laura Likes Llamas"} }' \
  --return-consumed-capacity TOTAL

aws dynamodb put-item \
  --table-name MusicCollection \
  --item '{
    "Artist": {"S": "Steps in Springtime"},
    "Title": {"S": "Allemande Left but will Return"} ,
    "AlbumTitle": {"S": "Square Dances for Round Rooms"} }' \
  --return-consumed-capacity TOTAL
```

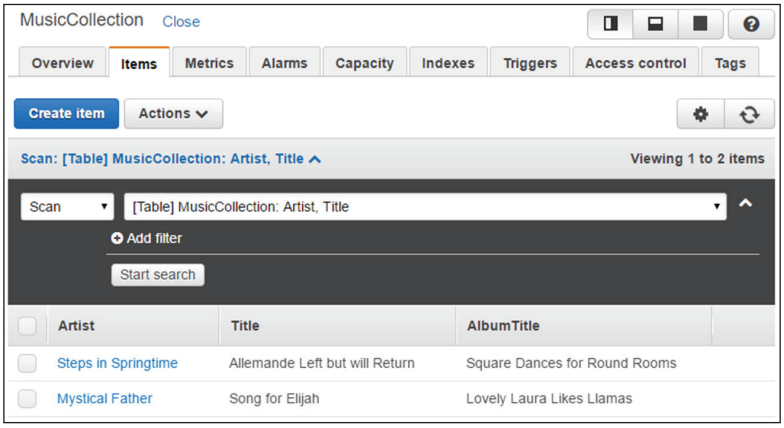
For Microsoft Windows, use the following:

```
aws dynamodb put-item `
  --table-name MusicCollection `
  --item '{
    \"Artist\": {\"S\": \"Mystical Father\"},
    \"Title\": {\"S\": \"Song for Elijah\"} ,
    \"AlbumTitle\": {\"S\": \"Lovely Laura Likes Llamas\"} }' `
  --return-consumed-capacity TOTAL

aws dynamodb put-item `
  --table-name MusicCollection `
  --item '{
    \"Artist\": {\"S\": \"Steps in Springtime\"},
    \"Title\": {\"S\": \"Allemande Left but will Return\"} ,
    \"AlbumTitle\": {\"S\": \"Square Dances for Round Rooms\"} }' `
  --return-consumed-capacity TOTAL
```

Microsoft Windows eats double quotes (") for breakfast. To hide them so that Amazon DynamoDB (or other services) can use them, use the backslash (\) as an escape character.

In the Amazon DynamoDB console, the table with items will look like the following graphic.



EXERCISE 7.4

Create a MySQL Amazon RDS DB Instance.

The basic building block of Amazon RDS is the DB instance. This is the environment in which you will run your MySQL databases.

In this exercise, create a DB instance running the MySQL database engine called **`west2-mysql-instance1`**, with a `db.m1.small` DB instance class, 5 GB of storage, and automated backups enabled with a retention period of one day.

Create a MySQL DB Instance

1. Sign in to the AWS Management Console, and open the Amazon RDS console.
2. In the top-right corner of the Amazon RDS console, choose the region in which to create the DB instance.
3. In the navigation pane, choose Instances.
4. Choose Launch DB Instance. The Launch DB Instance wizard opens on the Select Engine page.
5. On the Select Engine page, choose the MySQL icon, and then choose Select for the MySQL DB engine.

EXERCISE 7.4 *(continued)*

6. On the Specify DB Details page, specify the DB instance information as shown here:

DB Details

For this Parameter	Do This
License Model	Choose the default, general-public license to use the general license agreement for MySQL. MySQL has only one license model.
DB Engine Version	Choose the default version of MySQL. Note that Amazon RDS supports multiple versions of MySQL in some regions.
DB Instance Class	Choose `db.m1.small` for a configuration that equates to 1.7 GB memory, 1 ECU (1 virtual core with 1 ECU), 64-bit platform, and moderate I/O capacity.
Multi-AZ Deployment	Choose Yes to have a standby replica of the DB instance created in another Availability Zone for failover support. AWS recommends Multi-AZ for production workloads to maintain high availability.
Allocated Storage	Type **5** to allocate 5 GB of storage for the database. In some cases, allocating a higher amount of storage for your DB instance than the size of your database can improve I/O performance.
Storage Type	Choose the Magnetic storage type.
DB Instance Identifier	Type a name for the DB instance that is unique to the account in the region chosen. Feel free to add some intelligence to the name, such as including the region and DB engine chosen; for example, **`west2-mysql-instance1`**.
Master Username	Type a name using alphanumeric characters to use as the master user name to log on to the DB instance. This will be the user name used to log on to the database on the DB instance for the first time.
Master Password and Confirm Password	Type a password that contains from 8 to 41 printable ASCII characters (excluding /,", and @) for the master user password. This will be the password used for the user name to log on to the database. Type the password again in the Confirm Password box.

7. When the settings have been configured, choose Next.
8. On the Configure Advanced Settings page, provide additional information that Amazon RDS needs to launch the MySQL DB instance. The following table shows settings for the Amazon DB instance in this exercise:

Configure Advanced Settings

For This Parameter	Do This
VPC	Choose the name of the Amazon VPC that will host your MySQL DB instance.
Availability Zone	On the previous page, choosing Yes for the Multi-AZ Deployment parameter removes this option. Otherwise, select an AZ in which to deploy the database.
DB Security Groups	Choose the security group to use with this DB instance.
Database Name	Type a name for the default database that is between 1 to 64 alphanumeric characters long. Without a name, Amazon RDS will not automatically create a database on the newly provisioned DB instance.
Database Port	Leave the default value of 3306.
DB Parameter Group	Leave the default value.
Option Group	Choose the default value. This option group is used with the MySQL version chosen on the previous page.
Copy Tags To Snapshots	Choose this option to have any DB instance tags copied to a DB snapshot when creating a snapshot.
Enable Encryption	Choose Yes to enable encryption at rest for this DB instance.
Backup Retention Period	Set this value to 1.
Backup Window	Use the default of No Preference.
Enable Enhanced Monitoring	Use the default of No.
Auto Minor Version Upgrade	Choose Yes to enable the DB instance to receive minor DB engine version upgrades automatically when they become available.
Maintenance Window	Choose No Preference.

EXERCISE 7.4 *(continued)*

9. Specify your DB instance information and then choose Launch DB Instance.

On the Amazon RDS console, the new DB instance appears in the list of DB instances. The DB instance will have a status of `creating` until the DB instance is created and ready for use. When the state changes to `available`, connect to the database created on the DB instance.

Depending on the DB instance class and store allocated, it could take several minutes for the new DB instance to become available.

Review Questions

1. How are charges calculated for data transferred between Amazon Relational Database Service (Amazon RDS) and Amazon Elastic Compute Cloud (Amazon EC2) instances in the same Availability Zone?
   - **A.** GB out from Amazon RDS
   - **B.** GB in from Amazon RDS
   - **C.** IOPS from Amazon EC2
   - **D.** There is no charge.
2. On an Amazon Elastic Compute Cloud (Amazon EC2) instance using an Amazon Elastic Block Store (Amazon EBS) volume for persistence, there is a large inconsistency in the response times of the database queries. While investigating the issue, you find that there is a large amount of wait time on the database's disk volume. How can you improve the performance of the database's storage while maintaining the current persistence of the data? (Choose two.)
   - **A.** Move to a Solid-State Drive (SSD)-backed instance.
   - **B.** Use an Amazon EC2 instance with an Instance Store volume.
   - **C.** Use an Amazon EC2 instance type that is Amazon EBS-Optimized.
   - **D.** Use a Provisioned IOPs Amazon EBS volume.
3. Amazon Relational Database Service (Amazon RDS) automated backups and DB snapshots are currently supported for which storage engine?
   - **A.** JDBC
   - **B.** ODBC
   - **C.** InnoDB
   - **D.** MyISAM
4. In Amazon CloudWatch, which metric monitors the amount of free space on a DB instance?
   - **A.** `FreeStorageTotal`
   - **B.** `FreeStorageVolume`
   - **C.** `FreeStorageSpace`
   - **D.** `FreeRDSStorage`
5. In an Amazon DynamoDB table, what happens if the application performs more reads or writes than the provisioned capacity?
   - **A.** Requests above the provisioned capacity will be performed, but it will return HTTP 400 error codes.
   - **B.** Requests above the provisioned capacity will be throttled, and it will return HTTP 400 error codes.
   - **C.** Requests above the provisioned capacity will be performed, but it will return HTTP 500 error codes.
   - **D.** Requests above the provisioned capacity will be throttled, and it will return HTTP 500 error codes.

6. Which metric should be monitored carefully in order to know when a Read Replica should be recreated if it becomes out of sync due to replication errors?
   - **A.** `ReadLag`
   - **B.** `ReadReplica`
   - **C.** `ReplicaLag`
   - **D.** `ReplicaThreshold`

7. Can connections between my application and a DB instance be encrypted using Secure Sockets Layer (SSL)?
   - **A.** Yes
   - **B.** No
   - **C.** Yes, but only inside an Amazon Virtual Private Cloud (Amazon VPC)
   - **D.** Yes, but only inside certain regions

8. After inadvertently deleting a database table, which Amazon Relational Database Service (Amazon RDS) feature will allow you to restore a database reliably to within five minutes of the deletion?
   - **A.** Amazon RDS Automated Backup
   - **B.** Multi-AZ Amazon RDS
   - **C.** Amazon RDS Read Replicas
   - **D.** Amazon RDS Snapshots

9. What is the range, in days, to which the backup retention period can be set?
   - **A.** From 1 day to 7 days
   - **B.** From 1 day to 28 days
   - **C.** From 1 day to 35 days
   - **D.** From 1 day to 90 days

10. Regarding Amazon DynamoDB, which of the following statements is correct?
    - **A.** Each item should have at least two value sets: a primary key and an attribute.
    - **B.** An item can have multiple attributes.
    - **C.** Primary keys must store a single value.
    - **D.** Individual attributes can have one or more sub-attributes.

11. The maximum Amazon DynamoDB item size limit is which of the following?
    - **A.** 256 KB
    - **B.** 400 KB
    - **C.** 512 KB
    - **D.** 1,024 KB

12. Will a standby Amazon Relational Database Service (Amazon RDS) instance be in the same Availability Zone as the primary?
    - **A.** Yes
    - **B.** No
    - **C.** Only for Oracle and Microsoft SQL Server-based Amazon RDS instances
    - **D.** Only if configured at launch

13. For both Single and Multi-AZ deployments, defining a subnet for all Availability Zones in a region allows Amazon RDS to create a new standby in another Availability Zone should the need arise. Is creating a Read Replica of another Read Replica supported inside Amazon Relational Database Service (Amazon RDS)?
    - **A.** No
    - **B.** Only in certain regions
    - **C.** Only with MySQL-based Amazon RDS instances
    - **D.** Only for Oracle Amazon RDS DB instances

14. When running a DB instance as a Multi-AZ deployment, can the standby DB instance be used for read or write operations?
    - **A.** Yes
    - **B.** No
    - **C.** Only with Microsoft SQL Server-based Amazon Relational Database Service (Amazon RDS) DB instances
    - **D.** Only for Oracle-based Amazon RDS DB instances

15. If there are multiple Read Replicas for a primary DB instance and one of them is promoted, what happens to the rest of the Read Replicas?
    - **A.** The remaining Read Replicas will still replicate from the original primary DB instance.
    - **B.** The other Read Replicas will be deleted.
    - **C.** The remaining Read Replicas suspend operations.
    - **D.** Read Replicas cannot be promoted.

16. When using a Multi-AZ deployment, in the event of a planned or unplanned outage of the primary DB instance, Amazon RDS automatically switches to the standby replica. Which DNS record is updated by the automatic failover mechanism and points to the standby DB instance?
    - **A.** The A Record
    - **B.** CNAME
    - **C.** MX
    - **D.** SOA

17. When automatic failover occurs, Amazon Relational Database Service (Amazon RDS) will create a DB instance event. Using the AWS CLI, which of the following will return information about the events related to the DB instance?
    - **A.** `ReturnEventFailure`
    - **B.** `DescribeFailureEvent`
    - **C.** `DescribeEvents`
    - **D.** `ReturnEvents`

18. Is it possible to force a failover for a MySQL Multi-AZ DB instance deployment?
    - **A.** Yes
    - **B.** No
    - **C.** Only in certain regions
    - **D.** Only in Amazon Virtual Private Cloud (Amazon VPC)

19. How does the Amazon Relational Database Service (Amazon RDS) Multi-AZ model work?
    - **A.** A second, standby database is deployed and maintained in a different Availability Zone from the primary database, using synchronous replication.
    - **B.** A second, standby database is deployed and maintained in a different region from the primary database, using synchronous replication.
    - **C.** A second, standby database is deployed and maintained in a different Availability Zone from the primary database, using asynchronous replication.
    - **D.** A second, standby database is deployed and maintained in a different region from the primary database, using asynchronous replication.

20. What does Amazon ElastiCache provide?
    - **A.** An Amazon Elastic Compute Cloud (Amazon EC2) instance with a large amount of memory and CPU
    - **B.** A managed in-memory cache service
    - **C.** An Amazon EC2 instance with Redis and Memcached preinstalled
    - **D.** A highly available and fast indexing service for searching

21. When developing a highly available web application using stateless web servers, which services are suitable for storing session-state data? (Choose three.)
    - **A.** Amazon Simple Queue Service (Amazon SQS)
    - **B.** Amazon Relational Database Service (Amazon RDS)
    - **C.** Amazon CloudWatch
    - **D.** Amazon ElastiCache
    - **E.** Amazon DynamoDB
    - **F.** Amazon CloudFront

22. Which statement best describes Amazon ElastiCache?
    - **A.** It reduces the latency of a DB instance by splitting the workload across multiple Availability Zones.
    - **B.** It provides a simple web interface to create and store multiple datasets, query your data easily, and return the results.
    - **C.** It is a managed service from AWS that makes it easy to set up, operate, and scale a relational database in the cloud.
    - **D.** It offloads the read traffic from a database in order to reduce latency caused by read-heavy workload.

23. Which two AWS Cloud services provide out-of-the-box, user-configurable, automatic backup-as-a-service and backup rotation options? (Choose two.)
    - **A.** AWS CloudFormation
    - **B.** Amazon Simple Storage Service (Amazon S3)
    - **C.** Amazon Relational Database Service (Amazon RDS)
    - **D.** Amazon Elastic Block Store (Amazon EBS)
    - **E.** Amazon DynamoDB
    - **F.** Amazon Redshift

Chapter 8

Application Deployment and Management

THE AWS CERTIFIED SYSOPS ADMINISTRATOR - ASSOCIATE EXAM TOPICS COVERED IN THIS CHAPTER MAY INCLUDE, BUT ARE NOT LIMITED TO, THE FOLLOWING:

Domain 2.0 High Availability

✓ **2.1 Implement scalability and elasticity based on scenarios**

Content may include the following:

- Which AWS compute service to use for deploying scalable and elastic environments
- Including scalability of specific AWS compute service in the deployment and management of applications
- Methods for implementing upgrades while maintaining high availability

✓ **2.2 Ensure level of fault tolerance based on business needs**

Content may include the following:

- What AWS Cloud deployment services can be used for deploying fault-tolerant applications

Domain 4.0 Deployment and Provisioning

✓ **4.1 Demonstrate the ability to build the environment to conform to architectural design**

Content may include the following:

- Choosing the appropriate AWS Cloud service to meet requirements for deploying applications

✓ **4.2 Demonstrate the ability to provision cloud resources and manage implementation automation**

Content may include the following:

- Automating the deployment and provisioning of AWS Cloud services

Introduction to Application Deployment and Management

As a candidate for the AWS Certified SysOps Administrator – Associate certification, you will need to be familiar with application deployment strategies and services used for the deployment and management of applications. AWS offers many capabilities for provisioning your infrastructure and deploying your applications. The deployment model varies from customer to customer depending on the capabilities required to support operations. Understanding these capabilities and techniques will help you pick the best strategy and toolset for deploying the infrastructure that can handle your workload.

An experienced systems operator is aware of the "one size doesn't fit all" philosophy. For enterprise computing or to create the next big social media or gaming company, AWS provides multiple customization options to serve a broad range of use cases. The AWS platform is designed to address scalability, performance, security, and ease of deployment, as well as to provide tools to help migrate applications and an ecosystem of developers and architects that are deeply involved in the growth of its products and services.

This chapter details different deployment strategies and services. It reviews common features available on these deployment services, articulates strategies for updating application stacks, and presents examples of common usage patterns for various workloads.

Deployment Strategies

AWS offers several key features that are unique to each deployment service that will be discussed later in this chapter. There are some characteristics that are common to these services, however. This section discusses various common features and capabilities that a systems operator will need to understand to choose deployment strategies. Each feature can influence service adoption in its own way.

Provisioning Infrastructure

You can work with building-block services individually, such as provisioning an Amazon Elastic Compute Cloud (Amazon EC2) instance, Amazon Elastic Block Store (Amazon EBS)

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

the most up-to-date server configurations, which helps you avoid the types of problems that sometimes occur on long-running instances.

For a stateless web application, the update process is pretty straightforward. Simply upload the new version of your application and let your deployment service deploy a new version of your stack (green). To cut over to the new version, you simply replace the Elastic Load Balancing URLs in your Domain Name Server (DNS) records. AWS Elastic Beanstalk has a Swap Environment URLs feature to facilitate a simpler cutover process. If you use Amazon Route 53 to manage your DNS records, you need to swap Elastic Load Balancing endpoints for AWS CloudFormation or AWS OpsWorks deployment services.

For applications with session states, the cutover process can be complex. When you perform an update, you don't want your end users to experience downtime or lose data. You should consider storing the sessions outside of your deployment service because creating a new stack will re-create the session database with a certain deployment service. In particular, consider storing the sessions separately from your deployment service if you are using an Amazon RDS database.

If you use Amazon Route 53 to host your DNS records, you can consider using the Weighted Round Robin (WRR) feature for migrating from blue to green deployments. The feature helps to drive the traffic gradually rather than instantly. If your application has a bug, this method helps ensure that the blast radius is minimal, as it only affects a small number of users. This method also simplifies rollbacks if they become necessary by redirecting traffic back to the blue stack. In addition, you only use the required number of instances while you scale up in the green deployment and scale down in the blue deployment. For example, you can set WRR to allow 10 percent of the traffic to go to green deployment while keeping 90 percent of traffic on blue. You gradually increase the percentage of green instances until you achieve a full cutover. Keeping the DNS cache to a shorter Time To Live (TTL) on the client side also ensures that the client will connect to the green deployment with a rapid release cycle, thus minimizing bad DNS caching behavior. For more information on Amazon Route 53, see Chapter 5, "Networking."

Hybrid Deployments

You can also use the deployment services in a hybrid fashion for managing your application fleet. For example, you can combine the simplicity of managing AWS infrastructure provided by AWS Elastic Beanstalk and the automation of custom network segmentation provided by AWS CloudFormation. Leveraging a hybrid deployment model also simplifies your architecture because it decouples your deployment method so that you can choose different strategies for updating your application stack.

Deployment Services

AWS deployment services provide easier integration with other AWS Cloud services. Whether you need to load-balance across multiple Availability Zones by using Elastic Load Balancing or by using Amazon RDS as a back end, the deployment services like AWS Elastic

Beanstalk, AWS CloudFormation, and AWS OpsWorks make it simpler to use these services as part of your deployment.

If you need to use other AWS Cloud services, you can leverage tool-specific integration methods to interact with the resource. For example, if you are using AWS Elastic Beanstalk for deployment and want to use Amazon DynamoDB for your back end, you can customize your environment resources by including a configuration file within your application source bundle. With AWS OpsWorks, you can create custom recipes to configure the application so that it can access other AWS Cloud services. Similarly, several template snippets with a number of example scenarios are available for you to use within your AWS CloudFormation templates.

AWS offers multiple strategies for provisioning infrastructure. You could use the building blocks (for example Amazon EC2, Amazon EBS, Amazon S3, and Amazon RDS) and leverage the integration provided by third-party tools to deploy your application. But for even greater flexibility, you can consider the automation provided by the AWS deployment services.

AWS Elastic Beanstalk

AWS Elastic Beanstalk is the fastest and simplest way to get an application up and running on AWS. It is ideal for developers who want to deploy code and not worry about managing the underlying infrastructure. AWS Elastic Beanstalk reduces management complexity without restricting choice or control. You simply upload your application, and AWS Elastic Beanstalk automatically handles the details of capacity provisioning, load balancing, scaling, and application health monitoring.

AWS Elastic Beanstalk provides platforms for programming languages (such as Java, PHP, Python, Ruby, or Go), web containers (for example Tomcat, Passenger, and Puma), and Docker containers, with multiple configurations of each. AWS Elastic Beanstalk provisions the resources needed to run your application, including one or more Amazon EC2 instances. The software stack running on the Amazon EC2 instances depends on the configuration. In a configuration name, the version number refers to the version of the platform configuration. You can also perform most deployment tasks, such as changing the size of your fleet of Amazon EC2 instances or monitoring your application, directly from the AWS Elastic Beanstalk Console.

Applications

The first step in using AWS Elastic Beanstalk is to create an application that represents your web application in AWS. In AWS Elastic Beanstalk, an application serves as a container for the environments that run your web application and versions of your web application's source code, saved configurations, logs, and other artifacts that you create while using AWS Elastic Beanstalk.

Environment Tiers

When you launch an AWS Elastic Beanstalk environment, you choose an environment tier, platform, and environment type. The environment tier that you choose determines whether

that provisions the Amazon VPC in a standard way for the organization. By parameterizing the template, they can reuse it for different deployments.

AWS Command Line Interface (AWS CLI)

The AWS CLI can be used to automate systems operations. You can use it to manage an operational environment that is currently in production, automating the provisioning and updating of application deployments.

Generating Skeletons

Most AWS CLI commands support `--generate-cli-skeleton` and `--cli-input-json` parameters that you can use to store parameters in JSON and read them from a file instead of typing them at the command line. You can generate AWS CLI skeleton outputs in JSON that outline all of the parameters that can be specified for the operation. This is particularly useful for generating templates that are used in scripting the deployment of applications. For example, you can use it to generate a template for Amazon ECS task definitions or an AWS CloudFormation resource's `Properties` section.

To generate an Amazon ECS task definition template, run the following AWS CLI command.

```
aws ecs register-task-definition --generate-cli-skeleton
```

You can use this template to create your task definition, which can then be pasted into the console JSON input area or saved to a file and used with the AWS CLI `--cli-input-json` option.

This process is also useful in generating the properties attributes for an AWS CloudFormation resource. You could use the output generated in the previous command.

```
{
  "Type" : "AWS::ECS::TaskDefinition",
  "Properties" : {
       <paste-generate-cli-skeleton>
  }
}
```

This can be done for other resources in an AWS CloudFormation template, such as an Amazon EC2 instance. Run the following command and paste the output into the `Properties` section of the resource.

```
aws ec2 run-instances --generate-cli-skeleton
```

```
{
  "Type" : "AWS::EC2::Instance",
  "Properties" : {
       <paste-generate-cli-skeleton>
  }
}
```

Summary

In this chapter, we discussed the following deployment strategies:

- Provisioning Infrastructure
  - Automating the provisioning of building-block services with AWS Cloud deployment services
- Deploying Applications
  - Methods of deploying applications onto your infrastructure
- Configuration Management
  - Using tagging to track resources and manage infrastructure
  - Using custom variables to make deployments flexible and reusable
  - Strategies for creating AMIs
  - Configuring infrastructure for security and troubleshooting
- Scalability Capabilities
  - Including scaling capabilities for the infrastructure of an application
  - Scalability considerations during the upgrade of an application
- Continuous Deployment
  - In-place vs. replacement upgrades
  - Methods for deploying application and infrastructure upgrades

In this chapter, we discussed the following deployment services:

- AWS Elastic Beanstalk
  - Creating and managing AWS Elastic Beanstalk environments
  - Deploying and managing an application in an AWS Elastic Beanstalk environment
- Amazon ECS
  - Building an Amazon ECS cluster
  - Deploying instances used for an Amazon ECS cluster
  - Managing containers with Amazon ECS tasks and services
  - Using a repository for container images
- AWS OpsWorks Stacks
  - Creating an AWS OpsWorks stack
  - Using layers for managing Amazon EC2 instances
  - Deploying applications to a layer of your stack
- AWS CloudFormation
  - Creating, updating, and deleting an AWS CloudFormation Stack
  - Methods and considerations for updating an AWS CloudFormation Stack