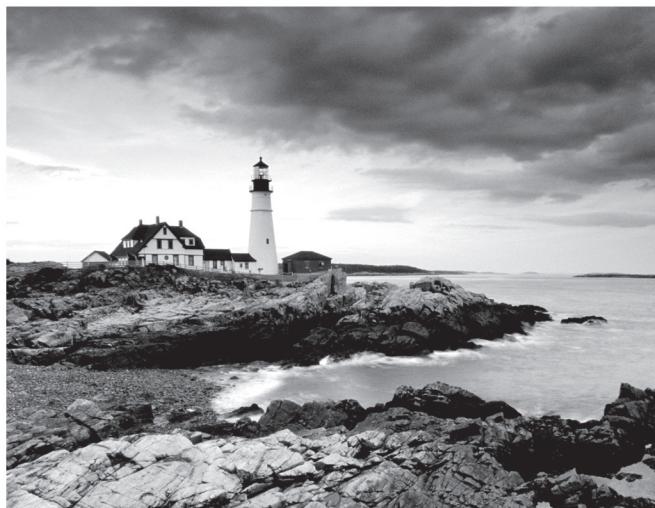


AWS[®]

Certified Developer

Official Study Guide

Associate (DVA-C01) Exam

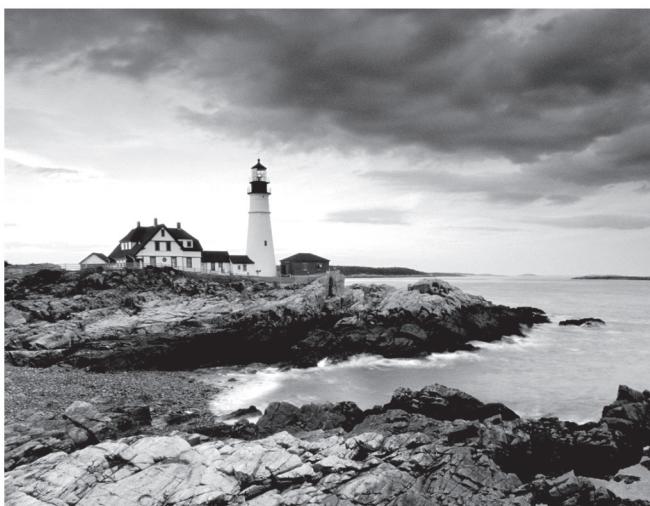


AWS®

Certified Developer

Official Study Guide

Associate (DVA-C01) Exam



Nick Alteen	Jennifer Fisher	Casey Gerena
Wes Gruver	Asim Jalil	Heiwad Osman
Marife Pagan	Santosh Patlolla	Michael Roth

Copyright © 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Published by John Wiley & Sons, Inc., Indianapolis, Indiana.

Published simultaneously in Canada

ISBN: 978-1-119-50819-9

ISBN: 978-1-119-50821-2 (ebk.)

ISBN: 978-1-119-50820-5 (ebk.)

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at <http://www.wiley.com/go/permissions>.

Limit of Liability/Disclaimer of Warranty: The publisher and the author make no representations or warranties with respect to the accuracy or completeness of the contents of this work and specifically disclaim all warranties, including without limitation warranties of fitness for a particular purpose. No warranty may be created or extended by sales or promotional materials. The advice and strategies contained herein may not be suitable for every situation. This work is sold with the understanding that the publisher is not engaged in rendering legal, accounting, or other professional services. If professional assistance is required, the services of a competent professional person should be sought. Neither the publisher nor the author shall be liable for damages arising herefrom. The fact that an organization or website is referred to in this work as a citation and/or a potential source of further information does not mean that the author or the publisher endorses the information the organization or website may provide or recommendations it may make. Further, readers should be aware that Internet website listed in this work may have changed or disappeared between when this work was written and when it is read.

For general information on our other products and services or to obtain technical support, please contact our Customer Care Department within the U.S. at (877) 762-2974, outside the U.S. at (317) 572-3993 or fax (317) 572-4002.

Wiley publishes in a variety of print and electronic formats and by print-on-demand. Some material included with standard print versions of this book may not be included in e-books or in print-on-demand. If this book refers to media such as a CD or DVD that is not included in the version you purchased, you may download this material at <http://booksupport.wiley.com>. For more information about Wiley products, visit www.wiley.com.

Library of Congress Control Number: 2019943088

TRADEMARKS: Wiley, the Wiley logo, and the Sybex logo are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates, in the United States and other countries, and may not be used without written permission. AWS is a registered trademark of Amazon Technologies, Inc. All other trademarks are the property of their respective owners. John Wiley & Sons, Inc. is not associated with any product or vendor mentioned in this book.

10 9 8 7 6 5 4 3 2 1

About the Authors



Nick Alteen, technical training architect, Amazon Web Services

Nick specializes in designing and building training labs that educate the U.S. intelligence community on AWS best practices and design patterns. Before this, Nick worked as a cloud support engineer, assisting customers in resolving any number of issues related to AWS DevOps services, with a specific focus on configuration management and infrastructure as code. In his free time, he enjoys building LEGO models with his daughter and watching horror movies with his wife.



Jennifer Fisher, senior technical curriculum developer, Amazon Web Services

Jennifer started at AWS in 2014 as a technical trainer and was the lead instructor for Big Data on AWS. She holds multiple AWS certifications and currently leads a curriculum development team and develops technical curriculum and labs to support public sector customers. Before that, Jennifer spent 20 years as a software and data engineer in the financial services, defense, and healthcare industries. She holds a BS in programming and an MS in software engineering management.

Jennifer grew up on a farm in Northern Maine and bought her first computer, a Tandy TRS-80, with her potato-picking money at the age of 12. She began writing basic programs and role-playing games, not realizing at the time that her passion for coding would turn into a lifelong career. She now mentors female engineers and volunteers for K-12 students in STEM.

Jennifer is based in Herndon, Virginia, and lives with her husband Steve. She is a doting stepmother to Kate, Sophie, and Mason. In her free time, Jennifer enjoys hiking, geocaching, kayaking, mountain biking, weight lifting, and competing in obstacle course races.

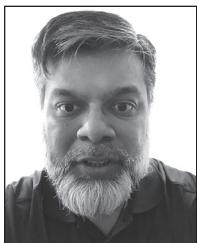


Casey Gerena, senior technical trainer, Amazon Web Services

Casey is passionate about helping others learn about the AWS Cloud. He enjoys teaching others new technical skills to help them solve problems using serverless technologies such as AWS Lambda. Casey holds a BS in management information systems from the University of Central Florida and an MS in logistics and global supply chain management from Embry-Riddle Aeronautical University. He is pursuing a second master's degree in computer science from the Georgia Institute of Technology. Casey holds several IT certifications, including the Certified Information Systems Security Professional (CISSP) and nine AWS certifications. Before joining Amazon, Casey was a software developer and cybersecurity consultant. In his free time, Casey enjoys spending time with his family, watching movies, playing video games, and running.

**Wes Gruver**, senior technical trainer, Amazon Web Services

Wes has been with AWS since 2015 and is a senior technical trainer with more than 20 years of experience and success in managing IT infrastructure and all aspects of application development and management. He is currently responsible for training AWS enterprise customers on how to use the AWS services best suited for their business and IT solutions. He teaches a broad range of classes, including basic to advanced architecture, DevOps on AWS, Big Data on AWS, and security operations. In his free time, Wes teaches scuba diving and loves to travel.

**Asim Jalis**, senior technical trainer, Amazon Web Services

Asim is a senior technical trainer at AWS. He has an MS in computer science from the University of Virginia and an MA in mathematics from the University of Wisconsin. When he is not working with AWS technologies, he likes to read and write fiction.

**Heiwad Osman**, senior manager, Solutions Builders, Amazon Web Services

Heiwad holds a BS in computer science and engineering from UCLA. In his role as an AWS trainer, he meets with AWS customers and teaches them to build resilient, scalable cloud applications. He has helped hundreds of software developers get started with AWS APIs through in-person training and online training videos. His current professional interests include user experience, web application development, and machine learning. In his free time, you can find him in New York City, trying new places to eat or relaxing in Central Park.

**Marife Pagan**, technical trainer, Amazon Web Services

Marife is a technical trainer for AWS, delivering training to AWS customers in North America. She has more than 15 years of experience in software and web development. Her experience brings a set of skills for multiple platforms, including .NET, Java, and Python. She holds a BS in information technology with a web design/development concentration from George Mason University, in addition to various leading industry certifications. She is currently working on her master's degree and pursuing higher studies in machine learning.

Before working at AWS, Marife worked for various government contracting firms, including Lockheed Martin. She also serves in the U.S. military as a signal officer working

on the setup and maintenance of LAN and WAN signal network footprints, supporting voice and data for various military operations. She currently lives in the Washington, DC, metro area, and in her spare time enjoys fitness, travel, and gardening.



Santosh Patlolla, technical curriculum architect, Amazon Web Services

Santosh is a technical curriculum architect for AWS. He has more than 18 years of experience in developing software applications, automated solutions, and migration projects with complex data conversions. Santosh has been instrumental in providing production-support solutions and managing application delivery programs for enterprises. He also designed cost-effective technical and business solutions for the banking and insurance industries. Santosh is passionate about applying this experience in using the broad range of AWS services for developing business automations. Outside of work, he coaches elementary school robotics, and enjoys watching basketball games and playing with his kids.



Michael Roth, technical trainer, Amazon Web Services

Michael is a technical trainer having joined Amazon in 2015. He is one of the authors of the SysOps Administrator Study Guide (also by Wiley). He is a Certified Cisco Network Academy Instructor, and he has taught Linux. Michael graduated from the University of Michigan with a BS in zoology and a BA in urban planning. He also has an MS in telecommunications management from Golden Gate University.

Michael would like to thank his coworkers in the AWS Training and Certification organization—he is very proud to be a part of this amazing group of people. Finally, he would like to thank his spouse, Betsy, and son, Robert. Without their support and love, this book would not have been possible.

Contents at a Glance

<i>Introduction</i>		<i>xxix</i>
<i>Assessment Test</i>		<i>xxxv</i>
Chapter 1	Introduction to AWS Cloud API	1
Chapter 2	Introduction to Compute and Networking	37
Chapter 3	Hello, Storage	85
Chapter 4	Hello, Databases	175
Chapter 5	Encryption on AWS	259
Chapter 6	Deployment Strategies	281
Chapter 7	Deployment as Code	317
Chapter 8	Infrastructure as Code	381
Chapter 9	Configuration as Code	445
Chapter 10	Authentication and Authorization	495
Chapter 11	Refactor to Microservices	519
Chapter 12	Serverless Compute	585
Chapter 13	Serverless Applications	621
Chapter 14	Stateless Application Patterns	663
Chapter 15	Monitoring and Troubleshooting	797
Chapter 16	Optimization	833
Appendix	Answers to Review Questions	885
<i>Index</i>		917

Contents

<i>Introduction</i>	<i>xxix</i>
<i>Assessment Test</i>	<i>xxxv</i>
Chapter 1 <i>Introduction to AWS Cloud API</i>	1
Introduction to AWS	2
Getting Started with an AWS Account	2
AWS Management Console	3
AWS Software Development Kits	4
AWS CLI Tools	4
Calling an AWS Cloud Service	5
API Example: Hello World	5
SDK Configuration	7
Working with Regions	9
Regions Are Highly Available	10
Working with Regional API Endpoints	10
API Credentials and AWS Identity and Access Management	14
Users	15
Groups	16
Roles	17
Choosing IAM Identities	19
Managing Authorization with Policies	20
Custom Policies	22
Summary	24
Exam Essentials	24
Resources to Review	25
Exercises	26
Review Questions	33
Chapter 2 <i>Introduction to Compute and Networking</i>	37
Amazon Elastic Compute Cloud	38
Instance Types	39
Storage	40
Software Images	41
Network Interfaces	42
Accessing Instances	43
Instance Lifecycle	43
Running Applications on Instances	44
Connecting to Amazon EC2 Instances	45
Customizing Software with User Data	46
Discovering Instance Metadata	47

Assigning AWS API Credentials	48
Serving a Custom Webpage	49
Monitoring Instances	50
Customizing the Network	51
Amazon Virtual Private Cloud	51
Connecting to Other Networks	51
IP Addresses	52
Subnets	54
Route Tables	55
Security Groups	56
Network Access Control Lists	58
Network Address Translation	61
DHCP Option Sets	63
Monitoring Amazon VPC Network Traffic	64
Managing Your Resources	64
Shared Responsibility Security Model	64
Comparing Managed and Unmanaged Services	65
Developer Tools	66
Summary	66
Exam Essentials	67
Resources to Review	68
Exercises	69
Review Questions	80
Chapter 3 Hello, Storage	85
Introduction to AWS Storage	86
Storage Fundamentals	87
Data Dimensions	87
One Tool Does Not Fit All	90
Block, Object, and File Storage	90
AWS Shared Responsibility Model and Storage	91
Confidentiality, Integrity, Availability Model	91
AWS Block Storage Services	92
Amazon Elastic Block Store	93
Instance Store	97
AWS Object Storage Services	99
Amazon Simple Storage Service	99
Object Lifecycle Management	134
AWS File Storage Services	136
Amazon Elastic File System	136
Storage Comparisons	142
Use Case Comparison	142
Storage Temperature Comparison	143
Comparison of Amazon EBS and Instance Store	143
Comparison of Amazon S3, Amazon EBS, and	
Amazon EFS	144

Cloud Data Migration	145	
AWS Storage Gateway	145	
AWS Import/Export	146	
AWS Snowball	147	
AWS Snowball Edge	148	
AWS Snowmobile	150	
Amazon Kinesis Data Firehose	151	
AWS Direct Connect	152	
VPN Connection	153	
Summary	154	
Exam Essentials	154	
Resources to Review	159	
Exercises	162	
Review Questions	170	
Chapter 4	Hello, Databases	175
Introduction to Databases	176	
Relational Databases	178	
Characteristics of Relational Databases	179	
Managed vs. Unmanaged Databases	180	
Nonrelational Databases	195	
NoSQL Database	195	
Amazon DynamoDB	196	
Data Warehouse	217	
Data Warehouse Architecture	217	
Amazon Redshift	220	
In-Memory Data Stores	226	
Caching	226	
In-Memory Key-Value Store	228	
Amazon ElastiCache	229	
Amazon DynamoDB Accelerator	230	
Graph Databases	230	
Amazon Neptune	231	
Cloud Database Migration	232	
AWS Database Migration Service	233	
AWS Schema Conversion Tool	234	
Running Your Own Database on Amazon Elastic Compute Cloud	235	
Compliance and Security	236	
AWS Identity and Access Management	236	
Summary	237	
Exam Essentials	237	
Resources to Review	239	
Exercises	242	
Review Questions	256	

Chapter 5	Encryption on AWS	259
Introduction to Encryption		260
AWS Key Management Service		260
Centralized Key Management		261
Integration with Other AWS Services		261
Auditing Capabilities and High Availability		262
Custom Key Store		262
Compliance		262
AWS CloudHSM		262
Controlling the Access Keys		263
Option 1: You Control the Encryption Method and the Entire KMI		264
Option 2: You Control the Encryption Method, AWS Provides the KMI Storage Component, and You Provide the KMI Management Layer		268
Option 3: AWS Controls the Encryption Method and the Entire KMI		269
Summary		273
Exam Essentials		273
Resources to Review		274
Exercises		275
Review Questions		279
Chapter 6	Deployment Strategies	281
Deployments on the AWS Cloud		282
Phases of the Release Lifecycle		282
Environment Variables		284
Software Development Lifecycle with AWS Cloud		284
Continuous Integration/Continuous Deployment		285
Deploying Highly Available and Scalable Applications		287
Deploying and Maintaining Applications		288
AWS Elastic Beanstalk		290
Implementation Responsibilities		291
Working with Your Source Repository		292
Concepts		293
AWS Elastic Beanstalk Command Line Interface		296
Customizing Environment Configurations		296
Integrating with Other AWS Services		297
AWS Identity and Access Management Roles		299
Deployment Strategies		299
All-at-Once and In-Place Deployments		300
Rolling Deployments		300
Container Deployments		302
Monitoring and Troubleshooting		303
Summary		307

Exam Essentials	307
Resources to Review	308
Exercises	309
Review Questions	313
Chapter 7 Deployment as Code	317
Introduction to AWS Code Services	318
Continuous Delivery with AWS CodePipeline	318
Benefits of Continuous Delivery	319
Using AWS CodePipeline to Automate Deployments	320
What Is AWS CodePipeline?	320
AWS CodePipeline Concepts	321
AWS CodePipeline Service Limits	328
AWS CodePipeline Tasks	329
Using AWS CodeCommit as a Source Repository	332
What Is AWS CodeCommit?	332
AWS CodeCommit Concepts	333
AWS CodeCommit Service Limits	343
Using AWS CodeCommit with AWS CodePipeline	344
Using AWS CodeBuild to Create Build Artifacts	344
What Is AWS CodeBuild?	345
AWS CodeBuild Concepts	345
AWS CodeBuild Service Limits	351
Using AWS CodeBuild with AWS CodePipeline	352
Using AWS CodeDeploy to Deploy Applications	352
What Is AWS CodeDeploy?	353
AWS CodeDeploy Concepts	353
AWS CodeDeploy Service Limits	370
Using AWS CodeDeploy with AWS CodePipeline	371
Summary	371
Exam Essentials	372
Resources to Review	373
Exercises	374
Review Questions	377
Chapter 8 Infrastructure as Code	381
Introduction to Infrastructure as Code	382
Infrastructure as Code	382
Using AWS CloudFormation to Deploy Infrastructure	383
What Is AWS CloudFormation?	383
AWS CloudFormation Concepts	384
AWS CloudFormation Service Limits	429
Using AWS CloudFormation with AWS CodePipeline	429
Summary	432
Exam Essentials	434

Resources to Review	436
Exercises	437
Review Questions	440
Chapter 9 Configuration as Code	445
Introduction to Configuration as Code	446
Using AWS OpsWorks Stacks to Deploy Applications	447
What Is AWS OpsWorks Stacks?	447
AWS OpsWorks Stack Concepts	448
AWS OpsWorks Stacks Service Limits	469
Using Amazon Elastic Container Service to Deploy Containers	471
What Is Amazon ECS?	472
Amazon ECS Concepts	472
Amazon ECS Service Limits	482
Using Amazon ECS with AWS CodePipeline	482
Summary	483
Exam Essentials	485
Resources to Review	487
Exercises	488
Review Questions	491
Chapter 10 Authentication and Authorization	495
Introduction to Authentication and Authorization	496
Different Planes of Control	497
Identity and Authorization	497
Microsoft Active Directory	500
AWS Security Token Service	502
Amazon Cognito	505
Summary	508
Exam Essentials	509
Resources to Review	509
Exercises	510
Review Questions	517
Chapter 11 Refactor to Microservices	519
Introduction to Refactor to Microservices	521
Amazon Simple Queue Service	523
Amazon SQS Parameters	525
Dead-Letter Queue	528
Monitoring Amazon SQS Queues Using Amazon CloudWatch	533
Amazon Simple Notification Service	534
Features and Functionality	536
Amazon SNS APIs	536

Transport Protocols	537		
Amazon SNS Mobile Push Notifications	537		
Billing, Limits, and Restrictions	539		
Amazon Kinesis Data Streams	540		
Multiple Applications	541		
High Throughput	541		
Real-Time Analytics	542		
Open Source Tools	542		
Producer Options	542		
Consumer Options	543		
Amazon Kinesis Data Firehose	543		
Amazon Kinesis Data Analytics	544		
Amazon Kinesis Video Streams	545		
Amazon DynamoDB Streams	546		
Amazon DynamoDB Streams Use Case	546		
Amazon DynamoDB Streams Consumers	546		
Amazon DynamoDB Streams Concurrency and Shards	547		
AWS IoT Device Management	547		
Rules Engine	548		
Message Broker	549		
Device Shadow	550		
Amazon MQ	550		
AWS Step Functions	551		
State Machine	551		
Task State	554		
Choice State	556		
Choice Rules	559		
Parallel State	561		
Parallel State Output	563		
End State	564		
Input and Output	564		
AWS Step Functions Use Case	568		
Summary	568		
Exam Essentials	569		
Resources to Review	570		
Exercises	573		
Review Questions	582		
Chapter 12		Serverless Compute	585
		Introduction to Serverless Compute	586
		AWS Lambda	586
		Where Did the Servers Go?	587
		Monolithic vs. Microservices Architecture	588

AWS Lambda Functions	588
Languages AWS Lambda Supports	589
Creating an AWS Lambda Function	589
Execution Methods/Invocation Models	590
Securing AWS Lambda Functions	592
Inside the AWS Lambda Function	593
Function Package	593
Function Handler	594
Event Object	595
Context Object	595
Configuring the AWS Lambda Function	596
Descriptions and Tags	596
Memory	596
Timeout	596
Network Configuration	596
Concurrency	597
Dead Letter Queues	599
Environment Variables	599
Versioning	599
Creating an Alias	600
Invoking AWS Lambda Functions	601
Monitoring AWS Lambda Functions	602
Using Amazon CloudWatch	602
Using AWS X-Ray	603
Summary	605
Exam Essentials	605
Resources to Review	606
Exercises	607
Review Questions	618
Chapter 13 Serverless Applications	621
Introduction to Serverless Applications	622
Web Server with Amazon Simple Storage Service (Presentation Tier)	622
Amazon S3 Static Website	623
Configuring Web Traffic Logs	624
Creating Custom Domain Name with Amazon Route 53	625
Speeding Up Content Delivery with Amazon CloudFront	626
Dynamic Data with Amazon API Gateway (Logic or App Tier)	627
Endpoints	628
Resources	629
HTTP Methods	630
Stages	630
Authorizers	630

API Keys	631
Cross-Origin Resource Sharing	631
Integrating with AWS Lambda	631
Monitoring Amazon API Gateway with Amazon CloudWatch	632
Other Notable Features	633
User Authentication with Amazon Cognito	634
Amazon Cognito User Pools	634
Password Policies	636
Multi-factor Authentication	636
Device Tracking and Remembering	636
User Interface Customization	637
Amazon Cognito Identity Pools	639
Amazon Cognito SDK	639
Standard Three-Tier vs. the Serverless Stack	640
Amazon Aurora Serverless	642
AWS Serverless Application Model	643
AWS SAM CLI	645
AWS Serverless Application Repository	647
Serverless Application Use Cases	647
Summary	647
Exam Essentials	649
Resources to Review	650
Exercises	651
Review Questions	660
Chapter 14	
Stateless Application Patterns	663
Introduction to the Stateless Application Pattern	664
Amazon DynamoDB	664
Using Amazon DynamoDB to Store State	665
Primary Key, Partition Key, and Sort Key	665
Using Write Shards to Distribute Workloads Evenly	668
Amazon DynamoDB Tables	672
Provisioned Throughput	672
Creating Tables to Store the State	678
Control Plane	678
Data Plane	679
Return Values	680
Requesting Throttle and Burst Capacity	682
Amazon DynamoDB Secondary Indexes: Global and Local	682
Amazon DynamoDB Streams	700
Amazon DynamoDB Auto Scaling	707
Managing Throughput Capacity Automatically with AWS Auto Scaling	708

Partitions and Data Distribution	711
Optimistic Locking with Version Number	713
Disabling Optimistic Locking	714
DynamoDB Tags	714
DynamoDB Items	715
Atomic Counters	715
Conditional Writes	716
Time to Live	719
Error Handling in Your Application	720
Capacity Units Consumed by Conditional Writes	721
Configuring Item Attributes	722
Working with Queries	729
DynamoDB Encryption at Rest	730
On-Demand Backup and Restore	737
Amazon ElastiCache	739
Considerations for Choosing a Distributed Cache	740
ElastiCache Terminology	741
Cache Scenarios	742
Scaling Your Environment	745
Backup and Recovery	746
Control Access	747
Amazon Simple Storage Service	747
Amazon S3 Core Concepts	747
Buckets	748
Bucket Policies	756
Amazon S3 Storage Classes	757
Amazon S3 Default Encryption for S3 Buckets	759
Working with Amazon S3 Objects	761
Performance Optimization	770
Storing Large Attribute Values in Amazon S3	772
Amazon Elastic File System	773
How Amazon EFS Works	773
Creating an IAM User	777
Creating Resources for Amazon EFS	777
Creating a File System	777
Using File Systems	778
Deleting an Amazon EFS File System	779
Managing Access to Encrypted File Systems	779
Amazon EFS Performance	779
Summary	781
Exam Essentials	782
Resources to Review	785
Exercises	786
Review Questions	793

Chapter 15	Monitoring and Troubleshooting	797
Introduction to Monitoring and Troubleshooting	798	
Monitoring Basics	799	
Amazon CloudWatch	800	
How Amazon CloudWatch Works	801	
Amazon CloudWatch Metrics	802	
Amazon CloudWatch Logs	811	
Amazon CloudWatch Alarms	814	
Amazon CloudWatch Dashboards	817	
AWS CloudTrail	818	
AWS X-Ray	820	
AWS X-Ray Use Cases	821	
Tracking Application Requests	821	
Summary	823	
Exam Essentials	823	
Resources to Review	825	
Exercises	826	
Review Questions	829	
Chapter 16	Optimization	833
Introduction to Optimization	834	
Cost Optimization: Everyone's Responsibility	834	
Tagging	835	
Reduce AWS Usage	836	
Right Sizing	838	
Select the Right Use Case	838	
Select the Right Instance Family	838	
Select the Right Instance Compatibility	840	
Using Instance Reservations	840	
AWS Pricing for Reserved Instances	840	
Amazon EC2 Reservations	841	
Amazon Relational Database Service Reservations	842	
Using Spot Instances	843	
Spot Fleets	843	
Amazon EC2 Fleets	844	
Design for Continuity	844	
Using AWS Auto Scaling	845	
Amazon EC2 Auto Scaling	846	
AWS Auto Scaling	847	
DynamoDB Auto Scaling	848	
Amazon Aurora Auto Scaling	848	
Accessing AWS Auto Scaling	848	

Using Containers	849	
Containerize Everything	849	
Containers without Servers	849	
Using Serverless Approaches	850	
Optimize Lambda Usage	851	
Optimizing Storage	851	
Object Storage	852	
Block Storage	852	
File Storage	853	
Optimize Amazon S3	853	
Optimize Amazon EBS	855	
Optimizing Data Transfer	858	
Caching	858	
Relational Databases and Amazon DynamoDB	859	
Apply NoSQL Design	860	
Keep Related Data Together	860	
Keep Fewer Tables	860	
Distribute Workloads Evenly	861	
Use Sort Keys for Version Control	862	
Keep the Number of Indexes to a Minimum	862	
Choose Projections Carefully	863	
Optimize Frequent Queries to Avoid Fetches	863	
Use Sparse Indexes	863	
Avoid Scans as Much as Possible	863	
Monitoring Costs	864	
Cost Management Tools	864	
Monitoring Performance	868	
Amazon CloudWatch	868	
AWS Trusted Advisor	869	
Summary	869	
Exam Essentials	871	
Resources to Review	874	
Exercises	876	
Review Questions	881	
Appendix	Answers to Review Questions	885
<i>Index</i>		917

Table of Exercises

Exercise	1.1	Sign Up for an Account.....	26
Exercise	1.2	Create an IAM Administrators Group and User.....	26
Exercise	1.3	Install and Configure the AWS CLI	28
Exercise	1.4	Download the Code Samples.....	28
Exercise	1.5	Run a Python Script that Makes AWS API Calls.....	29
Exercise	1.6	Working with Multiple Regions	29
Exercise	1.7	Working with Additional Profiles.....	30
Exercise	2.1	Create an Amazon EC2 Key Pair.....	69
Exercise	2.2	Create an Amazon VPC with Public and Private Subnets.....	70
Exercise	2.3	Use an IAM Role for API Calls from Amazon EC2 Instances	71
Exercise	2.4	Launch an Amazon EC2 Instance as a Web Server	71
Exercise	2.5	Connect to the Amazon EC2 Instance.....	73
Exercise	2.6	Configure NAT for Instances in the Private Subnet.....	74
Exercise	2.7	Launch an Amazon EC2 Instance into the Private Subnet	75
Exercise	2.8	Make Requests to Private Instance	76
Exercise	2.9	Launch an AWS Cloud9 Instance.....	77
Exercise	2.10	Perform Partial Cleanup	78
Exercise	2.11	(Optional) Complete Cleanup.....	79
Exercise	3.1	Create an Amazon Simple Storage Service (Amazon S3) Bucket	163
Exercise	3.2	Upload an Object to a Bucket.....	164
Exercise	3.3	Emptying and Deleting a Bucket.....	167
Exercise	4.1	Create a Security Group for the Database Tier on Amazon RDS.....	242
Exercise	4.2	Spin Up the MariaDB Database Instance	243
Exercise	4.3	Obtain the Endpoint Value for the Amazon RDS Instance	245
Exercise	4.4	Create a SQL Table and Add Records to It	246
Exercise	4.5	Query the Items in the SQL Table	248
Exercise	4.6	Remove Amazon RDS Database and Security Group.....	249
Exercise	4.7	Create an Amazon DynamoDB Table	250
Exercise	4.8	Add Users to the Amazon DynamoDB Table	252
Exercise	4.9	Look Up a User in the Amazon DynamoDB Table	253
Exercise	4.10	Write Data to the Table as a Batch Process	253
Exercise	4.11	Scan the Amazon DynamoDB Table	254
Exercise	4.12	Remove the Amazon DynamoDB Table	255

Exercise	5.1	Configure an Amazon S3 Bucket to Deny Unencrypted Uploads	275
Exercise	5.2	Create and Disable an AWS Key Management Service (AWS KMS) Key	276
Exercise	5.3	Create an AWS KMS Customer Master Key with the Python SDK	277
Exercise	6.1	Deploy Your Application	309
Exercise	6.2	Deploy a Blue/Green Solution	310
Exercise	6.3	Change Your Environment Configuration on AWS Elastic Beanstalk	310
Exercise	6.4	Update an Application Version on AWS Elastic Beanstalk	311
Exercise	7.1	Create an AWS CodeCommit Repository and Submit a Pull Request	374
Exercise	7.2	Create an Application in AWS CodeDeploy	375
Exercise	7.3	Create an AWS CodeBuild Project	375
Exercise	8.1	Write Your Own AWS CloudFormation Template	437
Exercise	8.2	Troubleshoot a Failed Stack Deletion	438
Exercise	8.3	Monitor Stack Update Activity	438
Exercise	9.1	Launch a Sample AWS OpsWorks Stacks Environment	488
Exercise	9.2	Launch an Amazon ECS Cluster and Containers	488
Exercise	9.3	Migrate an Amazon RDS Database	489
Exercise	9.4	Configure Auto Healing Event Notifications in AWS OpsWorks Stacks	490
Exercise	10.1	Setting Up a Simple Active Directory	510
Exercise	10.2	Setting Up an AWS Managed Microsoft AD	512
Exercise	10.3	Setting Up an Amazon Cloud Directory	514
Exercise	10.4	Setting Up Amazon Cognito	516
Exercise	11.1	Create an Amazon SQS Queue, Add Messages, and Receive Messages	573
Exercise	11.2	Send an SMS Text Message to Your Mobile Phone with Amazon SNS	575
Exercise	11.3	Create an Amazon Kinesis Data Stream and Write/Read Data	575
Exercise	11.4	Create an AWS Step Functions State Machine 1	578
Exercise	11.5	Create an AWS Step Functions State Machine 2	579
Exercise	12.1	Create an Amazon S3 Bucket for CSV Ingestion	608
Exercise	12.2	Create an Amazon S3 Bucket for Final Output JSON	608
Exercise	12.3	Verify List Buckets	609
Exercise	12.4	Prepare the AWS Lambda Function	610

Exercise	12.5	Create AWS IAM Roles	612
Exercise	12.6	Create the AWS Lambda Function.....	614
Exercise	12.7	Give Amazon S3 Permission to Invoke an AWS Lambda Function	615
Exercise	12.8	Add the Amazon S3 Event Trigger.....	616
Exercise	12.9	Test the AWS Lambda Function.....	617
Exercise	13.1	Create an Amazon S3 Bucket for the Swagger Template	652
Exercise	13.2	Edit the HTML Files	653
Exercise	13.3	Define an AWS SAM Template	655
Exercise	13.4	Define an AWS Lambda Function Locally	656
Exercise	13.5	Generate an Event Source	657
Exercise	13.6	Run the AWS Lambda Function.....	657
Exercise	13.7	Modify the AWS SAM template to Include an API Locally.....	658
Exercise	13.8	Modify Your AWS Lambda Function for the API	658
Exercise	13.9	Run Amazon API Gateway Locally.....	659
Exercise	14.1	Create an Amazon ElastiCache Cluster Running Memcached.....	786
Exercise	14.2	Expand the Size of a Memcached Cluster	787
Exercise	14.3	Create and Attach an Amazon EFS Volume	787
Exercise	14.4	Create and Upload to an Amazon S3 Bucket	788
Exercise	14.5	Create an Amazon DynamoDB Table	789
Exercise	14.6	Enable Amazon S3 Versioning.....	789
Exercise	14.7	Create an Amazon DynamoDB Global Table	790
Exercise	14.8	Enable Cross-Region Replication.....	791
Exercise	14.9	Create an Amazon DynamoDB Backup Table.....	791
Exercise	14.10	Restoring an Amazon DynamoDB Table from a Backup.....	792
Exercise	15.1	Create an Amazon CloudWatch Alarm on an Amazon S3 Bucket	826
Exercise	15.2	Enable an AWS CloudTrail Trail on an Amazon S3 Bucket	827
Exercise	15.3	Create an Amazon CloudWatch Dashboard.....	828
Exercise	16.1	Set Up a CPU Usage Alarm Using AWS CLI.....	876
Exercise	16.2	Modify Amazon EBS Optimization for a Running Instance	877
Exercise	16.3	Create an AWS Config Rule	878
Exercise	16.4	Create a Launch Configuration and an AWS Auto Scaling Group, and Schedule a Scaling Action	879

Foreword

Software development is changing. In today's competitive market, customers demand low-latency, highly scalable, responsive applications that work—all the time. Customers expect to receive the same level of performance and consistency of applications regardless of their device. Whether they are on a mobile device, desktop, laptop, or Amazon Fire tablet, they expect that applications will behave similarly across platforms.

The goal of building working applications that respond to increasing expectations means that building applications on highly available architecture is now more important than ever. As developers, you can use AWS Cloud computing to build highly available architectures and services on which to deploy and run your applications.

AWS provides you with a broad set of tools to build and develop your applications. We empower you by providing the best tools to achieve your goals. To that end, you'll learn about compute services, such as Amazon Elastic Compute Cloud (Amazon EC2), and file object storage services, such as Amazon Simple Storage Service (Amazon S3). You'll also learn about the many types of applications that you can build on top of these services.

Historically, developers have been responsible for designing, creating, and running their applications. In the AWS Cloud, you can create your compute resources with one click using AWS CloudFormation, or you can fully automate the running of your containers using AWS Fargate.

AWS continually listens to customer feedback to understand your workloads and changing needs better. AWS also monitors market trends, understanding that you want to build and run applications on the cloud, but you don't want to worry about managing the underlying infrastructure. You want infrastructure to scale automatically, you want services with a built-in high availability infrastructure, and you want to pay only for what you consume.

In response to these demands, AWS pioneered services such as AWS Lambda, which is based on serverless technology. It enables you to run compute programming logic in applications without having to worry about maintaining anything other than their code and core logic.

Today is the most exciting time to be a developer. With AWS services, you can focus on the core functionality of your application and allow the AWS Cloud to perform all of the administration of the resources, including server and operating system maintenance. This flexibility provides you with the unique ability to focus on what matters to you most—building, maintaining, and, most importantly, innovating your applications.

In this study guide, AWS experts coach you on how to develop and build applications that can run on and integrate with AWS services. This knowledge allows you, as a developer, to build your services and features quickly and get them running in the AWS Cloud for your customers to use. When you complete this guide and the test bank in the accompanying interactive online learning environment, you have gained the fundamental knowledge to succeed on the AWS Certified Developer – Associate certification exam.

So imagine, dream, and build, because on the AWS Cloud, the only limit is your imagination.

Werner Vogels
Vice President and Corporate Technology Officer
Amazon

Introduction

Developers are builders. They are responsible for imagining, designing, and building applications. This study guide is designed to help you develop, build, and create solutions by using AWS services and to provide you with the knowledge required to obtain the AWS Certified Developer – Associate certification.

The study guide covers relevant topics on the exam, with additional context to increase your understanding of how to build applications on AWS. This study guide references the exam blueprint throughout all of its chapters and content to provide a comprehensive view of the required knowledge to pass the exam. Furthermore, this study guide was designed to help you understand the key concepts required to earn the certification and for you to use as a reference for building highly available applications that run on the AWS Cloud. However, the study guide does not cover any prerequisite knowledge concerning software development; that is, the study guide does not cover how to program in Java, Python, .NET, and other platform languages. Instead, you will use these languages to build, manage, and deploy your resources on AWS.

The study guide begins with an introduction to the AWS Cloud and how you can interact with the AWS Cloud by using API calls. API calls are the heart of the AWS Cloud, as every interaction with AWS is an API call to the service. As such, the initial chapter provides you with the core knowledge on which the rest of the chapters are built. Because security is a top priority for all applications, the first chapter also describes how to create your API keys by using AWS Identity and Access Management (IAM). The rest of the chapters cover topics ranging from compute services, storage services, databases, encryption, and serverless-based applications.

The chapters were designed with the understanding that developers build. To enhance learning through hands-on experience, at the end of each chapter is an “Exercises” section with activities that help reinforce the main topic of the chapter. Each chapter also contains a “Review Questions” section to assess your understanding of the main concepts required to work with AWS. However, understand that the actual exam will test you on your ability to combine multiple concepts. The review questions at the end of each chapter focus only on the topics discussed in that chapter.

To help you determine the level of your AWS Cloud knowledge and aptitude before reading the guide, an assessment test with 50 questions is provided at the end of this introduction. Two practice exams with 75–100 questions each are also included to help you gauge your readiness to take the exam.

What Does This Book Cover?

This book covers topics that you need to know to prepare for the Amazon Web Services (AWS) Certified Developer – Associate Exam.

Chapter 1: Introduction to AWS Cloud API This chapter provides an overview of how to use AWS Cloud API calls. The chapter includes an introduction to AWS software development kits (AWS SDKs) and the AWS global infrastructure. A review of AWS API keys and how to manage them using AWS Identity and Access Management (IAM) is also included.

Chapter 2: Introduction to Compute and Networking This chapter reviews compute and networking environments in AWS. It provides an overview of resources, such as Amazon Elastic Compute Cloud (Amazon EC2), and the network controls exposed through Amazon Virtual Private Cloud (Amazon VPC).

Chapter 3: Hello, Storage In this chapter, you will learn about cloud storage with AWS. It provides a review of storage fundamentals and the AWS storage portfolio of services, such as Amazon Simple Storage Service (Amazon S3) and Amazon S3 Glacier. The chapter also covers how to choose the right type of storage for a workload.

Chapter 4: Hello, Databases This chapter provides an overview of the AWS database services. The chapter provides a baseline understanding of SQL versus NoSQL. It also introduces concepts such as caching with Amazon ElastiCache and business intelligence with Amazon Redshift. The chapter also covers Amazon Relational Database Service (Amazon RDS) and Amazon DynamoDB.

Chapter 5: Encryption on AWS In this chapter, you will explore AWS services that enable you to perform encryption of data at rest using both customer and AWS managed solutions. An overview of each approach and the use case for each is provided. Example architectures are included that show the differences between a customer and an AWS managed infrastructure.

Chapter 6: Deployment Strategies In this chapter, you will learn about automated application deployment, management, and maintenance by using AWS Elastic Beanstalk. You will also learn about the various deployment methodologies and options to determine the best approach for individual workloads.

Chapter 7: Deployment as Code This chapter describes the AWS code services used to automate infrastructure and application deployments across AWS and on-premises resources. You will learn about the differences among continuous integration, continuous delivery, and continuous deployment, in addition to how AWS enables you to achieve each.

Chapter 8: Infrastructure as Code This chapter focuses on AWS CloudFormation and how you can use the service to create flexible, repeatable templates for a cloud infrastructure. You will learn about the different AWS CloudFormation template components, supported resources, and how to integrate non-AWS resources into your templates using custom resources.

Chapter 9: Configuration as Code In this chapter, you will learn about AWS OpsWorks Stacks and Amazon Elastic Container Service (Amazon ECS). OpsWorks Stacks enables

you to perform automated configuration management on resources in your AWS account and on-premises instances using Chef cookbooks. You will learn how to add a Chef cookbook to your stack, associate it with an instance, and perform configuration changes. Using Amazon ECS, you will learn how to create clusters and services and how to deploy tasks to your cluster in response to changes in customer demand.

Chapter 10: Authentication and Authorization This chapter explains the differences between authentication and authorization and how these differences apply to infrastructure and applications running on AWS. You will also learn about integrating third-party identity services, in addition to the differences between the control pane and data pane.

Chapter 11: Refactor to Microservices In this chapter, you will learn about microservices and how to refactor large application stacks into small, portable containers. You will also learn how to implement messaging infrastructure to enable communication between microservices running in your environment.

Chapter 12: Serverless Compute This chapter reviews AWS Lambda as a compute service that you can use to run code without provisioning or managing servers. In this chapter, you will learn about creating, triggering, and securing Lambda functions. You will also learn other features of Lambda, such as versioning and aliases.

Chapter 13: Serverless Applications This chapter expands on the serverless concepts you learned in Chapter 12, “Serverless Compute,” and shows you how to architect a full-stack serverless web application. You will learn how to map server-based application architectures to serverless application architectures.

Chapter 14: Stateless Application Patterns This chapter expands on the concepts you learned in Chapter 13, “Serverless Applications,” by explaining how to design stateless applications. You will learn how to develop applications that do not depend on state information stored on individual resources, allowing for additional portability and availability.

Chapter 15: Monitoring and Troubleshooting This chapter discusses AWS services that you can use to monitor the health of your applications, in addition to changes to AWS resources over time. You will learn how to use Amazon CloudWatch to perform log analysis and create custom metrics for ingestion by other tools and for creating visualizations in the dashboard. You will also learn how to use AWS CloudTrail to monitor API activity for your AWS account to ensure that changes are appropriately audited over time. You will also learn how to use AWS X-Ray to create visual maps of application components for step-by-step analysis.

Chapter 16: Optimization This chapter covers some of the best practices and considerations for designing systems to achieve business outcomes at a minimal cost and to maintain optimal performance efficiency. This chapter covers scenarios for compute and storage, how to use a serverless platform, and what to consider for efficient data transfer to optimize your solutions. The chapter describes key AWS tools for managing and monitoring the cost and performance of your infrastructure. It includes code snippets, samples, and exercises to develop monitoring solutions and designs that integrate other AWS services.

Interactive Online Learning Environment and Test Bank

The authors have worked hard to provide you with some great tools to help you with your certification process. The interactive online learning environment that accompanies the *AWS Certified Developer – Associate Official Study Guide* provides a test bank with study tools to help you prepare for the certification exam. This helps you increase your chances of passing it the first time! The test bank includes the following:

Sample Tests All of the questions in this book, including the 50-question assessment test at the end of this introduction and the review questions that are provided at the end of each chapter are available online. In addition, there are two practice exams available online with 75–100 questions each. Use these questions to test your knowledge of the study guide material. The online test bank runs on multiple devices.

Flashcards The online test banks include more than 200 flashcards specifically written to quiz your knowledge of AWS operations. After completing all the exercises, review questions, practice exams, and flashcards, you should be more than ready to take the exam. The flashcard questions are provided in a digital flashcard format (a question followed by a single correct answer). You can use the flashcards to reinforce your learning and provide last-minute test prep before the exam.

Glossary A glossary of key terms from this book is available as a fully searchable PDF.



Go to www.wiley.com/go/sybextestprep to register and gain access to this interactive online learning environment and test bank with study tools.

Exam Objectives

The AWS Certified Developer – Associate Exam is intended for individuals who perform in a developer role. Exam concepts that you should understand for this exam include the following:

- Core AWS services, uses, and basic AWS architecture best practices
- Developing, deploying, and debugging cloud-based applications using AWS
In general, certification candidates should understand the following:
 - AWS APIs, AWS CLI, and AWS SDKs to write applications
 - Key features of AWS services
 - AWS shared responsibility model

- Application lifecycle management
- CI/CD pipeline to deploy applications on AWS
- Using or interacting with AWS services
- Using cloud-native applications to write code
- Writing code using AWS security best practices (for example, not using secret and access keys in the code, and instead using AWS Identity and Access Management (IAM) roles)
- Authoring, maintaining, and debugging code modules on AWS
- Writing code for serverless applications
- Using containers in the development process

The exam covers five different domains, with each domain broken down into objectives and subobjectives.

Objective Map

The following table lists each domain and its weighting in the exam, along with the chapters in this book where that domain's objectives and subobjectives are covered.

Domain	Percentage of Exam	Chapter
Domain 1: Deployment	22%	6, 7, 8, 9, 12, 13, 14
1.1 Deploy written code in AWS using existing CI/CD pipelines, processes, and patterns.		6, 7, 8, 9
1.2 Deploy applications using Elastic Beanstalk.		6, 8, 9
1.3 Prepare the application deployment package to be deployed to AWS.		7, 9, 12
1.4 Deploy serverless applications.		7, 12, 13, 14
Domain 2: Security	26%	1, 3, 4, 5, 6, 10, 12, 14
2.1 Make authenticated calls to AWS services.		1, 4, 10, 12, 13, 14
2.2 Implement encryption using AWS services.		3, 4, 5, 14
2.3 Implement application authentication and authorization.		3, 10, 13, 14

Domain 3: Development with AWS Services	30%	1, 2, 3, 4, 5, 7, 9, 12, 13, 14, 16
3.1 Write code for serverless applications.		9, 12, 13
3.2 Translate functional requirements into application design.		2, 3, 4, 13, 14
3.3 Implement application design into application code.		3, 4, 13, 14
3.4 Write code that interacts with AWS services by using APIs, SDKs, and AWS CLI.		1, 2, 3, 5, 7, 9, 12, 13, 14, 16
Domain 4: Refactoring	10%	2, 3, 4, 11,16
4.1 Optimize application to best use AWS services and features.		3, 4, 11, 16
4.2 Migrate existing application code to run on AWS.		2, 3, 11
Domain 5: Monitoring and Troubleshooting	12%	2, 4, 6, 8, 11, 12, 13, 15, 16
5.1 Write code that can be monitored.		8, 12, 13, 15, 16
5.2 Perform root cause analysis on faults found in testing or production.		2, 4, 12, 15

Assessment Test

1. You have an application running on Amazon Elastic Compute Cloud (Amazon EC2) that needs read-only access to several AWS services. What is the best way to grant that application permissions only to a specific set of resources within your account?
 - A. Use API credentials derived based on the AWS account.
 - B. Launch the EC2 instance into an AWS Identity and Access Management (IAM) role and attach the ReadOnlyAccess IAM-managed policy.
 - C. Declare the necessary permissions as statements in the AWS SDK configuration file on the EC2 instance.
 - D. Launch the EC2 instance into an IAM role with custom IAM policies for the permissions.
2. You have deployed a new application in the US West (Oregon) Region. However, you have accidentally deployed an Amazon Polly lexicon needed for your application in EU (London). How can you use your lexicon to synthesize speech while minimizing the changes to your application code and reducing cost?
 - A. Point your SDK client to the EU (London) for all requests to Amazon Polly, but to US West (Oregon) for all other API calls.
 - B. No action needed; the data is automatically available from all Regions.
 - C. Upload a copy of the lexicon to US West (Oregon).
 - D. Move the rest of the application resources to EU (London).
3. When you're placing subnets for a specific Amazon Virtual Private Cloud (Amazon VPC), you can place the subnets in which of the following?
 - A. In any Availability Zone within the Region for the Amazon VPC
 - B. In any Availability Zone in any Region
 - C. In any AWS edge location
 - D. In any specific AWS data center
4. You have identified two Amazon Elastic Compute Cloud (Amazon EC2) instances in your account that appear to have the same private IP address. What could be the cause?
 - A. These instances are in different Amazon Virtual Private Cloud (Amazon VPCs).
 - B. The instances are in different subnets.
 - C. The instances have different network ACLs.
 - D. The instances have different security groups.
5. You have a workload that requires 15,000 consistent IOPS for data that must be durable. What combination of the following do you need? (Select TWO.)
 - A. Use an Amazon Elastic Block Store (Amazon EBS) optimized instance.
 - B. Use an instance store.
 - C. Use a Provisioned IOPS SSD volume.
 - D. Use a previous-generation EBS volume.

6. Your company stores critical documents in Amazon Simple Storage Service (Amazon S3), but it wants to minimize cost. Most documents are used actively for only about one month and then used much less frequently after that. However, all data needs to be available within minutes when requested. How can you meet these requirements?

 - A. Migrate the data to Amazon S3 Reduced Redundancy Storage (RRS) after 30 days.
 - B. Migrate the data to Amazon S3 Glacier after 30 days.
 - C. Migrate the data to Amazon S3 Standard – Infrequent Access (IA) after 30 days.
 - D. Turn on versioning and then migrate the older version to Amazon S3 Glacier.
7. You are migrating your company’s applications and data from on-premises to the AWS Cloud. You have performed a data inventory and discovered that you will need to transfer about 2 PB of data to AWS. Which migration option will be the best choice for your company with minimal cost and shortest time?

 - A. AWS Snowball
 - B. AWS Snowmobile
 - C. Upload files directly to AWS over the internet using Amazon Simple Storage Service (Amazon S3) Transfer Acceleration.
 - D. Amazon Kinesis Data Firehose
8. You are changing your application to take advantage of the elasticity and cost benefits provided by AWS Auto Scaling. To do this, you must move session state information from the individual Amazon Elastic Compute Cloud (Amazon EC2) instances. Which of the following AWS Cloud services is best suited as an alternative for storing session state information?

 - A. Amazon DynamoDB
 - B. Amazon Redshift
 - C. AWS Storage Gateway
 - D. Amazon Kinesis
9. Your company’s senior management wants to query several data stores to obtain a “big picture” view of the business. The amount of data contained within the data stores is at least 2 TB in size. Which of the following is the best AWS service to deliver results to senior management?

 - A. Amazon Elastic Block Store (Amazon EBS)
 - B. Amazon Simple Storage Service (Amazon S3)
 - C. Amazon Relational Database Service (Amazon RDS)
 - D. Amazon Redshift
10. Your ecommerce application provides daily and ad hoc reporting to various business units on customer purchases. These operations result in a high level of read traffic to your MySQL Amazon Relational Database Service (Amazon RDS) instance. What can you do to scale up read traffic without impacting your database’s performance?

 - A. Increase the allocated storage for the Amazon RDS instance.
 - B. Modify the Amazon RDS instance to be a Multi-AZ deployment.

- C. Create a read replica for an Amazon RDS instance.
 - D. Change the Amazon RDS instance DB engine version.
11. Your company has refactored their application to use NoSQL instead of SQL. They would like to use a managed service for running the new NoSQL database. Which AWS service should you recommend?
- A. Amazon Relational Database Service (Amazon RDS)
 - B. Amazon Elastic Compute Cloud (Amazon EC2)
 - C. Amazon DynamoDB
 - D. Amazon Redshift
12. A company is currently using Amazon Relational Database Service (Amazon RDS); however, they are retiring a database that is currently running. They have automatic backups enabled on the database. They want to make sure that they retain the last backup before deleting the Amazon RDS database. As the lead developer on the project, what should you do?
- A. Delete the database. Amazon RDS automatic backups are already enabled.
 - B. Create a manual snapshot before deleting the database.
 - C. Use the AWS Database Migration Service (AWS DMS) to back up the database.
 - D. SSH into the Amazon RDS database and perform a SQL dump.
13. When using Amazon Redshift, which node do you use to run your SQL queries?
- A. Compute node
 - B. Cluster node
 - C. Master node
 - D. Leader node
14. Your company is building a recommendation feature for their application. They would like to use an AWS managed graph database. Which service should you recommend?
- A. Amazon Relational Database Service (Amazon RDS)
 - B. Amazon Neptune
 - C. Amazon ElastiCache
 - D. Amazon Redshift
15. You have an Amazon DynamoDB table that has a partition key and a sort key. However, a business analyst on your team wants to be able to query the DynamoDB table with a different partition key. What should you do?
- A. Create a local secondary index.
 - B. Create a global secondary index.
 - C. Create a new DynamoDB table.
 - D. Advise the business analyst that this is not possible.

- 16.** An application is using Amazon DynamoDB. Recently, a developer on your team has noticed that occasionally the application does not return the most up-to-date data after a read from the database. How can you solve this issue?
- A.** Increase the number of read capacity units (RCUs) for the table.
 - B.** Increase the number of write capacity units (WCUs) for the table.
 - C.** Refactor the application to use a SQL database.
 - D.** Configure the application to perform a strongly consistent read.
- 17.** A developer on your team would like to test a new idea and requires a NoSQL database. Your current applications are using Amazon DynamoDB. What should you recommend?
- A.** Create a new table inside DynamoDB.
 - B.** Use DynamoDB Local.
 - C.** Use another NoSQL database on-premises.
 - D.** Create an Amazon Elastic Compute Cloud (Amazon EC2) instance, and install a NoSQL database.
- 18.** The AWS Encryption SDK provides an encryption library that integrates with AWS Key Management Service (AWS KMS) as a master key provider. Which of the following operations does the AWS Encryption SDK perform to build on the AWS SDKs?
- A.** Generates, encrypts, and decrypts data keys
 - B.** Uses the data keys to encrypt and decrypt your raw data
 - C.** Stores the encrypted data keys with the corresponding encrypted data in a single object
 - D.** All of the above
- 19.** Of all the cryptographic algorithms that the AWS Encryption SDK supports, which one is the default algorithm?
- A.** AES-256
 - B.** AES-192
 - C.** AES-128
 - D.** SSH-256
- 20.** Amazon Elastic Block Store (Amazon EBS) volumes are encrypted by default.
- A.** True
 - B.** False
- 21.** Which of the following cannot be retained when deleting an AWS Elastic Beanstalk environment?
- A.** Source code from the Git repository
 - B.** Data from the automatic backups of an Amazon Relational Database Service (Amazon RDS) instance
 - C.** Packaged code from the source bundle stored in an Amazon Simple Storage Service (Amazon S3) bucket
 - D.** Data from the snapshot of an Amazon RDS instance

- 22.** Which of the following is not part of the AWS Elastic Beanstalk functionality?
- A.** Notify the account user of language runtime platform changes
 - B.** Display events per environment
 - C.** Show instance statuses per environment
 - D.** Perform automatic changes to AWS Identity and Access Management (IAM) policies
- 23.** What happens to AWS CodePipeline revisions that, upon reaching a manual approval gate, are rejected?
- A.** The pipeline continues.
 - B.** A notification is sent to the account administrator.
 - C.** The revision is treated as failed.
 - D.** The pipeline creates a revision clone and continues.
- 24.** Which of the following is an invalid strategy for migrating data to AWS CodeCommit?
- A.** Incrementally committing files from a large repository
 - B.** Syncing the files from Amazon Simple Storage Service (Amazon S3) using the sync AWS CLI command
 - C.** Cloning an existing repository, updating the remote, and pushing
 - D.** Manually creating files in the AWS Management Console
- 25.** You have an AWS CodeBuild task in your pipeline that requires large binary files that do not frequently change. What would be the best way to include these files in your build?
- A.** Store the files in your source code repository. They will be passed in as part of the revision.
 - B.** Store the files in an Amazon Simple Storage Service (Amazon S3) bucket and copy them during the build.
 - C.** Create a custom build container that includes the files.
 - D.** It is not possible to include files above a certain size.
- 26.** When you update an AWS::S3::Bucket resource, what is the expected behavior if the Name property is updated?
- A.** The resource is updated with no interruption.
 - B.** The resource is updated with some interruption.
 - C.** The resource is replaced.
 - D.** The resource is deleted.
- 27.** What is the preferred method for updating resources created by AWS CloudFormation?
- A.** Updating the resource directly in the AWS Management Console
 - B.** Submitting an updated template to AWS CloudFormation to modify the stack
 - C.** Updating the resource using the AWS Command Line Interface (AWS CLI)
 - D.** Updating the resource using an AWS Software Development Kit (AWS SDK)

- 28.** When does the AWS OpsWorks Stacks configure lifecycle event run?
- A.** On individual instances immediately when they are first created
 - B.** On individual instances after a deploy lifecycle event
 - C.** On all instances in a stack when a single instance comes online or goes offline
 - D.** On all instances in a stack after a deploy lifecycle event
- 29.** Which non-Amazon Elastic Compute Cloud (Amazon EC2) AWS resources can AWS OpsWorks Stacks manage? (Select THREE.)
- A.** Elastic IP addresses
 - B.** Amazon Elastic Block Store (Amazon EBS) volumes
 - C.** Amazon Relational Database Service (Amazon RDS) database instances
 - D.** Amazon ElastiCache clusters
 - E.** Amazon Redshift data warehouses
- 30.** Which AWS Cloud service can Simple Active Directory (Simple AD) use to authenticate users?
- A.** Amazon WorkDocs
 - B.** Amazon Cognito
 - C.** Amazon Elastic Compute Cloud (Amazon EC2)
 - D.** Amazon Simple Storage Service (Amazon S3)
- 31.** What is the best application of Amazon Cognito?
- A.** Use instead of Active Directory for AWS Identity and Access Management (IAM) users.
 - B.** Provide authentication to third-party web applications.
 - C.** Use as an Amazon Aurora database.
 - D.** Use to access objects in an Amazon Simple Storage Service (Amazon S3) bucket.
- 32.** You manage a sales tracking system in which point-of-sale devices send transactions of this form:
- ```
{"date": "2017-01-30", "amount": 100.20, "product_id": "1012", "region": "WA", "customer_id": "3382"}
```
- You need to generate two real-time reports. The first reports on the total sales per day for each customer. The second reports on the total sales per day for each product. Which AWS offerings and services can you use to generate these real-time reports?
- A.** Ingest the data through Amazon Kinesis Data Streams. Use Amazon Kinesis Data Analytics to query for sales per day for each product and sales per day for each customer using SQL queries. Feed the result into two new streams in Amazon Kinesis Data Firehose.
  - B.** Ingest the data through Kinesis Data Streams. Use Kinesis Data Firehose to query for sales per day for each product and sales per day for each customer with SQL queries. Feed the result into two new streams in Kinesis Data Firehose.

- C. Ingest the data through Kinesis Data Analytics. Use Kinesis Data Streams to query for sales per day for each product and sales per day for each customer with SQL queries. Feed the result into two new streams in Kinesis Data Firehose.
  - D. Ingest the data in Amazon Simple Queue Service (Amazon SQS). Use Kinesis Data Firehose to query for sales per day for each product and sales per day for each customer with SQL queries. Feed the result into two new streams in Kinesis Data Firehose.
- 33.** You design an application for selling toys online. Every time a customer orders a toy, you want to add an item into the `orders` table in Amazon DynamoDB and send an email to the customer acknowledging their order. The solution should be performant and cost-effective. How can you trigger this email?
- A. Use an Amazon Simple Queue Service (Amazon SQS) queue.
  - B. Schedule an AWS Lambda function to check for changes to the `orders` table every minute.
  - C. Schedule an Lambda function to check for changes to the `orders` table every second.
  - D. Use Amazon DynamoDB Streams.
- 34.** A company would like to use Amazon DynamoDB. They want to set up a NoSQL-style trigger. Is this something that can be accomplished? If so, how?
- A. No. This cannot be done with DynamoDB and NoSQL.
  - B. Yes, but not with AWS Lambda.
  - C. No. DynamoDB is not a supported event source for Lambda.
  - D. Yes. You can use Amazon DynamoDB Streams and poll them with Lambda.
- 35.** A company wants to access the infrastructure on which AWS Lambda runs. Is this possible?
- A. No. Lambda is a managed service and runs the necessary infrastructure on your behalf.
  - B. Yes. They can access the infrastructure and make changes to the underlying OS.
  - C. Yes. They need to open a support ticket.
  - D. Yes, but they need to contact their Solutions Architect to provide access to the environment.
- 36.** Using the smallest amount of memory possible for an AWS Lambda function, currently 128 MB, will result in the lowest bill.
- A. True. Lambda bills based on the total memory allocated.
  - B. False. Lambda has a flat rate—memory allocation is not important for billing, only performance.
  - C. False. Lambda bills based on memory plus the number of times that you trigger the function.
  - D. False. Lambda bills based on memory, the amount of compute time spent on a function in 100-ms increments, and the number of times that you execute or trigger a function.

- 37.** Which Amazon services can you use for caching? (Select TWO.)
- A.** AWS CloudFormation
  - B.** Amazon Simple Storage Service (Amazon S3)
  - C.** Amazon CloudFront
  - D.** Amazon ElastiCache
- 38.** Which Amazon API Gateway feature enables you to create a separate path that can be helpful in creating a development endpoint and a production endpoint?
- A.** Authorizers
  - B.** API keys
  - C.** Stages
  - D.** Cross-origin resource sharing (CORS)
- 39.** Which of the following methods does Amazon API Gateway support?
- A.** GET
  - B.** POST
  - C.** OPTIONS
  - D.** All of the above
- 40.** Which authorization mechanisms does Amazon API Gateway support?
- A.** AWS Identity and Access Management (IAM) policies
  - B.** AWS Lambda custom authorizers
  - C.** Amazon Cognito user pools
  - D.** All of the above
- 41.** Which tool can you use to develop and test AWS Lambda functions locally?
- A.** AWS Serverless Application Model (AWS SAM)
  - B.** AWS SAM CLI
  - C.** AWS CloudFormation
  - D.** None of the above
- 42.** Which serverless AWS service can you use to store user session state?
- A.** Amazon Elastic Compute Cloud (Amazon EC2)
  - B.** Amazon ElastiCache
  - C.** AWS Elastic Beanstalk
  - D.** Amazon DynamoDB
- 43.** Which AWS service can you use to store user profile information?
- A.** Amazon CloudFront
  - B.** Amazon Cognito
  - C.** Amazon Kinesis
  - D.** AWS Lambda

- 44.** Which of the following objects are good candidates to store in a cache? (Select THREE.)
- A.** Session state
  - B.** Shopping cart
  - C.** Product catalog
  - D.** Bank account balance
- 45.** Which of the following cache engines does Amazon ElastiCache support? (Select TWO.)
- A.** Redis
  - B.** MySQL
  - C.** Couchbase
  - D.** Memcached
- 46.** How can you aggregate Amazon CloudWatch metrics across Regions?
- A.** CloudWatch does not aggregate data across Regions.
  - B.** This is enabled by default.
  - C.** Send the metric data from other Regions to Amazon Simple Storage Service (Amazon S3) for retrieval by CloudWatch.
  - D.** Stream the metric data to Amazon Kinesis, and retrieve it using an AWS Lambda function.
- 47.** Why would an Amazon CloudWatch alarm report as INSUFFICIENT\_DATA instead of OK or ALARM? (Select THREE.)
- A.** The alarm was just created.
  - B.** The metric is not available.
  - C.** There is an AWS Identity and Access Management (IAM) permission preventing the metric from receiving data.
  - D.** Not enough data is available for the metric to determine the alarm state.
  - E.** The alarm period is missing.
- 48.** You were asked to develop an administrative web application that consumes low throughput and rarely receives high traffic. Which of the following instance type families will be the most optimized choice?
- A.** Memory optimized
  - B.** Compute optimized
  - C.** General purpose
  - D.** Accelerated computing
- 49.** Which of the following AWS Cost Management Tools can you use to view your costs and find ways to take advantage of elasticity?
- A.** AWS Cost Explorer
  - B.** AWS Trusted Advisor
  - C.** Amazon CloudWatch
  - D.** Amazon EC2 Auto Scaling

- 50.** Because cloud resources are easier to deploy and they incur usage-based costs, your organization is setting up good governance rules to manage costs. They are currently focusing on controlling and restricting Amazon Elastic Compute Cloud (Amazon EC2) instance deployments. Which of the following is an effective recommendation?
- A.** Seek approval from Cost Engineering teams before deploying any EC2 instances.
  - B.** Use AWS Identity and Access Management (IAM) policies to enable engineers to deploy EC2 instances only when specific mandatory tags are used.
  - C.** Review Amazon CloudWatch metrics to optimize the resource utilization.
  - D.** Use AWS Cost Explorer usage and forecasting reports.
- 51.** Because your applications are showing a consistent steady-state compute usage, you have decided to purchase Amazon Elastic Compute Cloud (Amazon EC2) Reserved Instances to gain significant pricing discounts. Which of the following is *not* the best purchase option?
- A.** All Upfront
  - B.** Partial Upfront
  - C.** No Upfront
  - D.** Pay-as-you-go
- 52.** Your application processes transaction-heavy and IOPS-intensive database workloads. You need to choose the right Amazon Elastic Block Store (Amazon EBS) volume so that application performance is not affected. Which of the following options would you suggest?
- A.** HDD-backed storage (st1)
  - B.** SSD-backed storage (io1)
  - C.** Amazon Simple Storage Service (Amazon S3) Intelligent Tier class storage
  - D.** Cold HDD-backed storage (sc1)
- 53.** A legacy financial institution is planning for a huge technical upgrade and planning to go global. The architecture depends heavily on using caching solutions. Which one of the following services does *not* fit into the caching solutions?
- A.** Amazon ElastiCache for Redis
  - B.** Amazon ElastiCache for Memcached
  - C.** Amazon DynamoDB Accelerator
  - D.** Amazon Elastic Compute Cloud (Amazon EC2) memory-optimized
- 54.** Which of the following characteristics separates Amazon DynamoDB from the Amazon Relational Database Service (Amazon RDS) design?
- A.** Incurs the performance costs of an ACID-compliant transaction system
  - B.** Normalizes data and stores it on multiple tables
  - C.** Keeps related data together
  - D.** May require expensive joins

- 55.** Which of the following partition key choices is an inefficient design that leads to poor distribution of the data in an Amazon DynamoDB table?
- A.** User ID, where the application has many users
  - B.** Device ID, where each device accesses data at relatively similar intervals
  - C.** Status code, where there are only a few possible status codes
  - D.** Session ID, where the user session remains distinct
- 56.** You are planning to build serverless backends by using AWS Lambda to handle web, mobile, Internet of Things (IoT), and third-party API requests. Which of the following are the main benefits in opting for a serverless architecture in this scenario? (Select THREE.)
- A.** No need to manage servers
  - B.** No need to ensure application fault tolerance and fleet management
  - C.** No charge for idle capacity
  - D.** Flexible maintenance schedules
  - E.** Powered for high complex processing
- 57.** Your enterprise infrastructure has recently migrated to the AWS Cloud. You are now trying to optimize the storage solutions. Which of the following are the appropriate storage management tools that you can use to review and analyze the storage classes and access patterns usage to help reduce costs? (Select TWO.)
- A.** Amazon Simple Storage Service (Amazon S3) analytics
  - B.** Cost allocation Amazon S3 bucket tags
  - C.** Amazon S3 Transfer Acceleration
  - D.** Amazon Route 53
  - E.** AWS Budgets

# Answers to Assessment Test

1. D. Use the custom IAM policy to configure the permissions to a specific set of resources in your account. The ReadOnlyAccess IAM policy restricts write access but grants access to all resources within your account. AWS account credentials are unrestricted. Policies do not go in an SDK configuration file. They are enforced by AWS on the backend.
2. C. This is the simplest approach because only a single resource is in the wrong Region. Option A is a possible approach, but it is not the simplest approach because it introduces cross-region calls that may increase latency and cross-region data transfer pricing.
3. A. Each Amazon VPC is placed in a specific Region and can span all the Availability Zones within that Region. Option B is incorrect because a subnet must be placed within the Region for the selected VPC. Option C is incorrect because edge locations are not available for subnets, and option D is incorrect because you cannot choose specific data centers.
4. A. Even though each instance in an Amazon VPC has a unique private IP address, you could assign the same private IP address ranges to multiple Amazon VPCs. Therefore, two instances in two different Amazon VPCs in your account could end up with the same private IP address. Options B, C, and D are incorrect because within the same Amazon VPC, there is no duplication of private IP addresses.
5. A, C. Amazon EBS optimized instances reserve network bandwidth on the instance for I/O, and Provisioned IOPS SSD volumes provide the highest consistent IOPS. Option B is incorrect because instance store is not durable. Option D is incorrect because a previous-generation EBS volume offers an average of 100 IOPS.
6. C. Migrating the data to Amazon S3 Standard-IA after 30 days using a lifecycle policy is correct. The lifecycle policy will automatically change the storage class for objects aged over 30 days. The Standard-IA storage class is for data that is accessed less frequently, but still requires rapid access when needed. It offers the same high durability, high throughput, and low latency of Standard, with a lower per gigabyte storage price and per gigabyte retrieval fee. Option A is incorrect because RRS provides a lower level of redundancy. The question did not state that the customer is willing to reduce the redundancy level of the data, and RRS does not replicate objects as many times as standard Amazon S3 storage. This storage option enables customers to store noncritical, reproducible data. Option B is incorrect because the fastest retrieval option for Amazon S3 Glacier is typically 3–5 hours. The customer requires retrieval in minutes. Option D is incorrect. Versioning will increase the number of files if new versions of files are being uploaded, which will increase cost. The question did not mention a need for multiple versions of files.
7. A. Option B is incorrect. You could use Snowmobile, but that would not be as cost effective because it is meant to be used for datasets of 10 PB or more. Option C is incorrect because uploading files directly over the internet to Amazon S3, even using Amazon S3 Transfer Accelerator, would take many months and would be using your on-premises bandwidth. Option D is incorrect because Amazon Kinesis Data Firehose would still be transferring over the internet and take months to complete while using your on-premises bandwidth.

8. A. DynamoDB is a NoSQL database store that is a good alternative because of its scalability, high availability, and durability characteristics. Many platforms provide open source, drop-in replacement libraries that enable you to store native sessions in DynamoDB. DynamoDB is a suitable candidate for a session storage solution in a share-nothing, distributed architecture.
9. D. Amazon Redshift is the best choice for data warehouse workloads that typically span multiple data repositories and are at least 2 TB in size.
10. C. Amazon RDS read replicas provide enhanced performance and durability for Amazon RDS instances. This replication feature makes it easy to scale out elastically beyond the capacity constraints of a single Amazon RDS instance for read-heavy database workloads. You can create one or more replicas of a given source Amazon RDS instance and serve high-volume application read traffic from multiple copies of your data, increasing aggregate read throughput.
11. C. DynamoDB is the best option. The question states a *managed service*, so this eliminates the Amazon EC2 service. Additionally, Amazon RDS and Amazon Redshift are SQL database products. The company is looking for a NoSQL product. DynamoDB is a managed NoSQL service.
12. B. Automatic backups do not retain the backup after the database is deleted. Therefore, option A is incorrect. Option C is incorrect. The AWS Database Migration Service is used to migrate databases from one source to another, which isn't what you are trying to accomplish here. Option D is incorrect because you cannot SSH into the Amazon RDS database, which is an AWS managed service.
13. D. The leader node acts as the SQL endpoint and receives queries from client applications, parses the queries, and develops query execution plans. Option A is incorrect because the compute nodes execute the query execution plan. However, the leader node is where you will submit the actual query. Options B and C are incorrect because there is no such thing as a cluster or master node in Amazon Redshift.
14. B. Amazon Neptune is a managed graph database service, which can be used to build recommendation applications. Option A is incorrect, because Amazon RDS is a managed database service and you are looking for a graph database. Option C is incorrect. Amazon ElastiCache is a caching managed database service. Option D is incorrect. Amazon Redshift is a data warehouse service.
15. B. A global secondary index enables you to use a different partition key or primary key in addition to a different sort key. Option A is incorrect because a local secondary index can only have a different sort key. Option C is incorrect. A new DynamoDB table would not solve the issue. Option D is incorrect because it is possible to accomplish this.
16. D. The application is configured to perform an eventually consistent read, which may not return the most up-to-date data. Option A is incorrect—increasing RCUs does not solve the underlying issue. Option B is incorrect because this is a read issue, not a write issue. Option C is incorrect. There is no need to refactor the entire application, because the issue is solvable.

- 17.** B. DynamoDB Local is the downloadable version of DynamoDB that enables you to write and test applications without accessing the web service. Option A is incorrect. Although you can create a new table, there is a cost associated with this option, so it is not the best option. Option C is incorrect. Even though you can use another NoSQL database, your team is already using DynamoDB. This strategy would require them to learn a new database platform. Additionally, you would have to migrate the database to DynamoDB after development is done. Option D is incorrect for the same reasons as option C.
- 18.** D. The AWS Encryption SDK is a client-side library designed to streamline data security operations so that customers can follow encryption best practices. It supports the management of data keys, encryption and decryption activities, and the storage of encrypted data. Thus, option D is correct.
- 19.** A. Options B, C, and D refer to more outdated encryption algorithms. By default, the AWS Encryption SDK uses the industry-recommended AES-256 algorithm.
- 20.** B. Encryption of Amazon EBS volumes is optional.
- 21.** B. Elastic Beanstalk automatically deletes your Amazon RDS instance when your environment is deleted and does not automatically retain the data. You must create a snapshot of the Amazon RDS instance to retain the data.
- 22.** D. Elastic Beanstalk cannot make automated changes to the policies attached to the service roles and instance roles.
- 23.** C. Option C is correct because if a revision does not pass a manual approval transition (either by expiring or by being rejected), it is treated as a failed revision. Successive revisions can then progress past this approval gate (if they are approved). Pipeline actions for a specific revision will not continue past a rejected approval gate, so option A is incorrect. A notification can be sent to an Amazon Simple Notification Service (Amazon SNS) topic that you specify when a revision reaches a manual approval gate, but no additional notification is sent if a change is rejected; therefore, option B is incorrect. Option D is incorrect, as AWS CodePipeline does not have a concept of “cloning” revisions.
- 24.** B. Though option D would be time-consuming, it is still possible to create files in the AWS CodeCommit console. Option A is a recommended strategy for migrating a repository containing a large number of files. Option C is also a valid strategy for smaller repositories. However, there is no way to sync files directly from an Amazon S3 bucket to an AWS CodeCommit repository. Thus, option B is correct.
- 25.** C. Option A is not recommended, because storing binary files in a Git-based repository incurs significant storage costs. Option B can work. However, you would have to pay additional data transfer costs any time a build is started. Option C is the most appropriate choice, because you can update the build container any time you need to change the files. Option D is incorrect, as AWS CodeBuild does not limit the size of files that can be used.
- 26.** C. Amazon Simple Storage Service (Amazon S3) bucket names are globally unique and cannot be changed after a bucket is created. Thus, options A and B are incorrect. Option D is incorrect because the resource is not being deleted, only updated. Option C is correct because you must create a replacement bucket when changing this property in AWS CloudFormation.

- 27.** B. Option B is correct because you can manage resources declared in a stack entirely within AWS CloudFormation by performing stack updates. Manually updating the resource outside of AWS CloudFormation (using the AWS Management Console, AWS CLI, or AWS SDK) will result in inconsistencies between the state expected by AWS CloudFormation and the actual resource state. This can cause future stack operations to fail. Thus, options A, C, and D are incorrect.
- 28.** C. Option A is incorrect because this is not the only time configure events run on instances in a stack. Options B and D are incorrect because the configure event does not run after a deploy event. AWS OpsWorks Stacks issues a configure lifecycle event on all instances in a stack any time a single instance goes offline or comes online. This is so that all instances in a stack can be made “aware” of the instance’s status. Thus, option C is correct.
- 29.** A, B, C. AWS OpsWorks Stacks includes the ability to manage AWS resources such as Elastic IP addresses, EBS volumes, and Amazon RDS instances. Thus, options A, B, and C are correct. Options D and E are incorrect because OpsWorks Stacks does not include any automatic integrations with Amazon ElastiCache or Amazon Redshift.
- 30.** A. Option A is correct because Simple Active Directory (Simple AD) can be used to authenticate users of Amazon WorkDocs. Options B, C, and D are incorrect because Amazon Cognito is an identity provider (IdP), and you cannot use Simple AD to authenticate users of Amazon EC2 or Amazon S3.
- 31.** B. Amazon Cognito acts as an identity provider (IdP) to mobile applications, eliminating the need to embed credentials into the web application itself. Option A is incorrect because if a customer is currently using Active Directory as their IdP, it is not good practice to create another IdP to operate and manage. Option C is incorrect because an Amazon Aurora database that is used to track data does not assign policies. Option D is incorrect because you can use Amazon Cognito to control an application’s access to either an S3 bucket or an Amazon S3 object. You don’t use it to directly control access to that bucket or object.
- 32.** A. Option A is correct because you want to ingest into Amazon Kinesis Data Streams, pass that into Amazon Kinesis Data Analytics, and finally feed that data into Amazon Kinesis Data Firehose. Option B is incorrect because Kinesis Data Firehose cannot run SQL queries. Option C is incorrect because Kinesis Data Streams cannot run SQL queries. Option D is incorrect because Kinesis Data Analytics cannot run SQL queries against data in Amazon SQS.
- 33.** D. Option D is correct because Amazon DynamoDB Streams allows Amazon DynamoDB to publish a message every time there is a change in a table. This solution is performant and cost-effective. Option A is incorrect because if you add an item to the orders table in DynamoDB, it does not automatically produce messages in Amazon Simple Queue Service (Amazon SQS). Options B and C are incorrect because if you check the orders table every minute or every second, it will degrade performance and increase costs.
- 34.** D. AWS Lambda supports Amazon DynamoDB event streams as an event source, which can be polled. You can configure Lambda to poll this stream, look for changes, and create a trigger. Option A is incorrect because this can be accomplished with DynamoDB event streams. Option B is incorrect because this can be accomplished with Lambda. Option C DynamoDB is a supported event source for Lambda.

- 35.** A. AWS Lambda uses containers to operate and is a managed service—you cannot access the underlying infrastructure. This is a benefit because your organization does not need to worry about security patching and other system maintenance. Option B is incorrect—you cannot access the infrastructure. Recall that Lambda is serverless. Option C is incorrect. AWS Support cannot provide access to the direct environment. Option D is incorrect—the Solutions Architect cannot provide direct access to the environment.
- 36.** D. AWS Lambda uses three factors when determining cost: the amount of memory allocated, the amount of compute time spent on a function (in 100-ms increments), and the number of times you execute or trigger a function. Options A, B, and C are all incorrect because Lambda is billed based on memory allocated, compute time spent on a function in 100-ms increments, and the number of times that you execute or trigger a function.
- 37.** C, D. Option A is incorrect because AWS CloudFormation is a service that helps you model and set up your AWS resources. Option B is incorrect because you use Amazon S3 as a storage tool for the internet. Options C and D are correct because they are both caching tools.
- 38.** C. Option A is incorrect, as authorizers enable you to control access to your APIs by using Amazon Cognito or an AWS Lambda function. Option B is incorrect because API keys are used to provide customers to your API, which is useful for selling your API. Option C is the correct answer. You can use stages to create a separate path with multiple endpoints, such as development and production. Option D is incorrect, as CORS is used to allow one service to call another service.
- 39.** D. API Gateway supports all of the methods listed. GET, POST, PUT, PATCH, DELETE, HEAD, and OPTIONS are all supported methods.
- 40.** D. With Amazon API Gateway, you can enable authorization for a particular method with IAM policies, AWS Lambda custom authorizers, and Amazon Cognito user pools. Options A, B, and C are all correct, but option D is the best option because it combines all of them.
- 41.** B. Option A is incorrect. Though AWS SAM is needed for the YAML/JSON template defining the function, it does not allow for testing the AWS Lambda function locally. Option B is the correct answer. AWS SAM CLI allows you to test the Lambda function locally. Option C is incorrect. AWS CloudFormation is used to deploy resources to the AWS Cloud. Option D is incorrect because AWS SAM CLI is the tool to test Lambda functions locally.
- 42.** D. Option A is incorrect. Amazon EC2 is a virtual machine service. Option B is incorrect because Amazon ElastiCache deploys clusters of machines, which you are then responsible for scaling. Option C is incorrect because Elastic Beanstalk deploys full stack applications by using Amazon EC2. Option D is correct because ElastiCache can store session state in a NoSQL database. This option is also serverless.
- 43.** B. With Amazon Cognito, you can create user pools to store user profile information and store attributes such as user name, phone number, address, and so on. Option A is incorrect. Amazon CloudFront is a content delivery network (CDN). Option C is incorrect. Amazon Kinesis is a service that you can implement to collect, process, and analyze streaming data in real time. Option D is incorrect. By using AWS Lambda, you can create custom programming functions for compute processing.

- 44.** A, B, C. Option D is incorrect because when compared to the other options, a bank balance is not likely to be stored in a cache; it is probably not data that is retrieved as frequently as the others. Options A, B, and C are all better data candidates to cache because multiple users are more likely to access them repeatedly. However, you could also cache the bank account balance for shorter periods if the database query is not performing well.
- 45.** A, D. Options A and D are correct because Amazon ElastiCache supports both the Redis and Memcached open source caching engines. Option B is incorrect because MySQL is not a caching engine—it is a relational database engine. Option C is incorrect because Couchbase is a NoSQL database and not one of the caching engines that ElastiCache supports.
- 46.** A. Amazon CloudWatch does not aggregate data across Regions; therefore, option A is correct.
- 47.** A, B, D. Amazon CloudWatch alarms changes to a state other than INSUFFICIENT\_DATA only when the alarm resource has had sufficient time to initialize and there is sufficient data available for the specified metric and period. Option C is incorrect because permissions for sending metrics to CloudWatch are the responsibility of the resource sending the data. Option D is incorrect because the alarm does not create successfully unless it has a valid period.
- 48.** C. General-purpose instances provide a balance of compute, memory, and networking resources. T2 instances are a low-cost option that provides a small amount of CPU resources that can be increased in short bursts when additional cycles are available. They are well suited for lower-throughput applications, such as administrative applications or low-traffic websites. For more details on the instance types, see <https://aws.amazon.com/ec2/instance-types/>.
- 49.** A. AWS Cost Explorer reflects the cost and usage of Amazon Elastic Compute Cloud (Amazon EC2) instances over the most recent 13 months and forecasts potential spending for the next 3 months. By using Cost Explorer, you can examine patterns on how much you spend on AWS resources over time, identify areas that need further inquiry, and view trends that help you understand your costs. In addition, you can specify time ranges for the data and view time data by day or by month. Option D is incorrect because Amazon EC2 Auto Scaling helps you to maintain application availability and enables you to add or remove EC2 instances automatically according to conditions that you define. It does not give you insights into costs incurred.
- 50.** B. You can use tags to control permissions. Using IAM policies, you can enforce the tag to gain precise control over access to resources, ownership, and accurate cost allocation. Option A is incorrect because eventually deployments become unmanageable, given the scale and rate at which resources get deployed in a successful organization. Options C and D are incorrect because Amazon CloudWatch and AWS Cost Explorer are unrelated to access controls and measures, and these tools monitor resources after they are created.
- 51.** D. You can choose among the three payment options when you purchase a Standard or Convertible Reserved Instance. With the All Upfront option, you pay for the entire Reserved Instance term with one upfront payment. This option provides you with the largest discount compared to On-Demand Instance pricing. With the Partial Upfront option, you make a low upfront payment and then are charged a discounted hourly rate for the instance for the duration of the Reserved Instance term. The No Upfront option requires no upfront payment and provides a discounted hourly rate for the duration of the term.

- 52.** B. The performance of the transaction-heavy workloads depends primarily on IOPS; SSD-backed volumes are designed for transactional, IOPS-intensive database workloads, boot volumes, and workloads that require high IOPS. For more information, see <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AmazonEBS.html>.
- 53.** D. Options A, B, and C help in building a high-speed data storage layer that stores a subset of data. This data is typically transient in nature so that future requests for that data are served up faster than is possible by accessing the data's primary storage location. Option D only supplements the setup of your own caching mechanism, and that is not the preferred solution for this scenario. For more information, see <https://aws.amazon.com/caching/aws-caching/>.
- 54.** C. Keeping data together is a basic characteristic of a NoSQL database such as Amazon DynamoDB. Keeping related data in proximity has a major impact on cost and performance. Instead of distributing related data items across multiple tables, keep related items in your NoSQL system as close together as possible. Options A, B, and D are typical characteristics of a relational database.
- 55.** C. The status code option suggests an inefficient partition key, because few possible status codes lead to uneven distribution of data and cause request throttling. Options A, B, and D suggest the efficient partition keys because of their distinct nature, which leads to an even distribution of the data. For more information, see:  
<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-partition-key-design.html>
- 56.** A, B, C. Using a serverless approach means not having to manage servers and not incurring compute costs when there is no user traffic. This is achieved while still offering instant scale to meet high demand, such as a flash sale on an ecommerce site or a social media mention that drives a sudden wave of traffic. Option D is incorrect because AWS Lambda runs your code on a high-availability compute infrastructure and performs all the administration of the compute resources, including server and operating system maintenance, capacity provisioning and automatic scaling, code and security patch deployment, and code monitoring and logging. Option E is incorrect because you can configure Lambda functions to run up to 15 minutes per execution. As a best practice, set the timeout value based on your expected execution time to prevent your function from running longer than intended.
- 57.** A, B. Use this feature to analyze storage access patterns to help you decide when to transition the right data to the right storage class. This feature observes data access patterns to help you determine when to transition less frequently accessed STANDARD storage to the STANDARD\_IA storage class. Option B is correct. A cost allocation tag is a key-value pair that you associate with an Amazon S3 bucket. To manage storage data most effectively, you can use these tags to categorize your Amazon S3 objects and filter on these tags in your data lifecycle policies. Options C and D are incorrect. These options focus on establishing a solution with an efficient data transfer. Option E is incorrect. With AWS Budgets, you can set custom budgets that alert you when your costs or usage exceed (or are forecasted to exceed) your budgeted amount.

# Chapter 1

AWS® Certified Developer Official Study Guide  
By Nick Alteen, Jennifer Fisher, Casey Gerena, Wes Gruver, Asim Jalis,  
Heiward Osman, Marife Pagan, Santosh Patlolla and Michael Roth  
Copyright © 2019 by Amazon Web Services, Inc.

# Introduction to AWS Cloud API

---

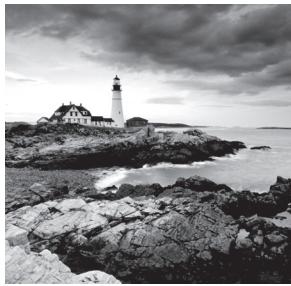
**THE AWS CERTIFIED DEVELOPER –  
ASSOCIATE EXAM TOPICS COVERED IN  
THIS CHAPTER MAY INCLUDE, BUT ARE  
NOT LIMITED TO, THE FOLLOWING:**

## Domain 2: Security

- ✓ 2.1 Make authenticated calls to AWS services.

## Domain 3: Development with AWS Services

- ✓ 3.4 Write code that interacts with AWS services by using APIs, SDKs, and AWS CLI.



# Introduction to AWS

The AWS Cloud provides infrastructure services, such as compute, storage, networking, and databases, and a broad set of platform capabilities such as mobile services, analytics, and machine learning (ML). These services are available on demand, through the internet, and with pay-as-you-go pricing.

Think of AWS as a programmable data center. Rather than making a phone call or sending email to provision servers or other resources, you can manage all of your resources programmatically, via *application programming interfaces* (APIs). For example, you can provision virtual servers on demand in minutes and pay only for the compute capacity you use. The same is true for de-provisioning those servers; make a single API call to stop paying for resources that you no longer need. AWS operates many data centers worldwide, so you are not limited to a single data center.

In this chapter, you are introduced to AWS and shown how to make your first API calls. The AWS infrastructure behind the API calls follows. Afterward, you will learn how to manage the API credentials and permissions that you need to make API calls.

## Getting Started with an AWS Account

The AWS Certified Developer – Associate is designed for developers who have hands-on experience working with AWS services. To help you prepare, this book has recommended exercises at the end of each chapter.

To work with AWS, you'll need an account. While you must provide contact and payment information to sign up for an account, you can test many of these services through the *AWS Free Tier*. The AWS Free Tier limits allow you to become familiar with the APIs for the included services without incurring charges.

The AWS Free Tier automatically provides usage alerts to help you stay in control of usage and identify possible charges. You can define additional alerts with AWS Budgets. To best take advantage of the AWS Free Tier and reduce costs, take some time to review the AWS Free Tier limits, and make sure to shut down or delete resources when you are done using them.

To create an account, sign up at <https://aws.amazon.com/free>.

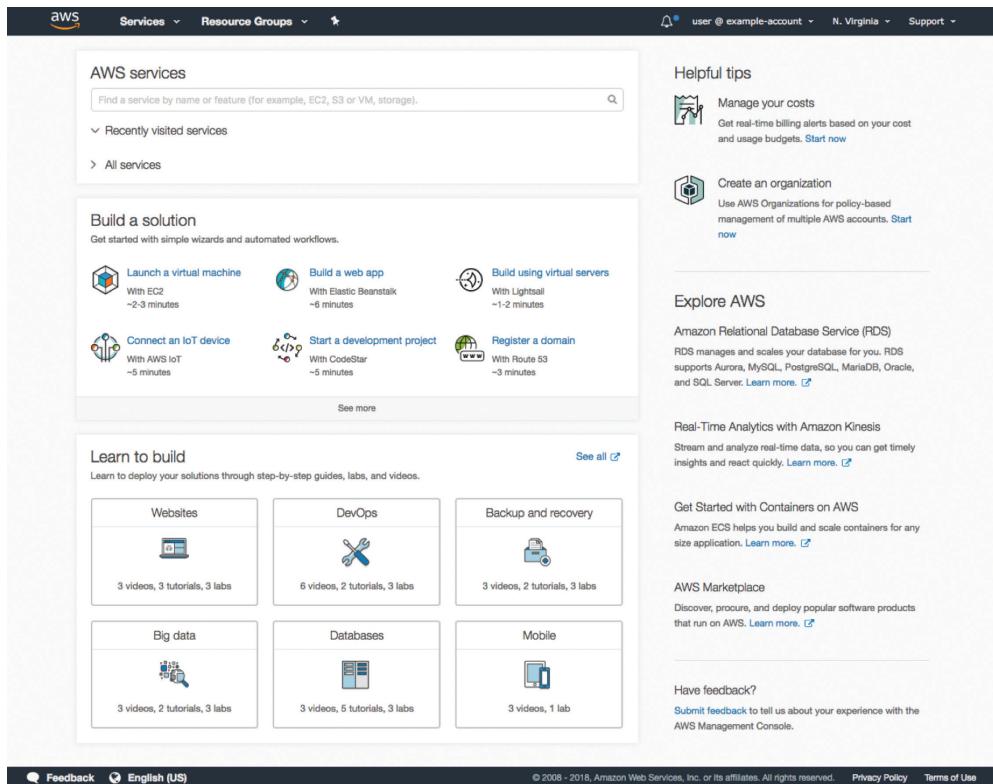
## AWS Management Console

After you have created an account, you will be prompted to sign in to the *AWS Management Console*. As part of the sign-up process, you define an email address and password to sign in to the console as the root user for the account.

The console is a web interface where you can create, configure, and monitor AWS resources in your account. You can quickly identify the AWS services that are available to you and explore the functionality of those services. Links are also provided to learning materials to help you get started.

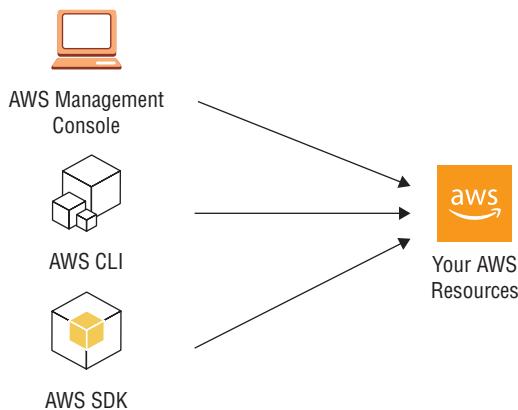
Sign in to the console, as shown in Figure 1.1, at <https://signin.aws.amazon.com/console>.

**FIGURE 1.1** AWS Management Console



Because all the functionality of AWS is exposed through APIs, AWS provides more than only the web interface for managing resources. For example, the console is also available as a mobile app for iOS and for Android.

After you become familiar with a service, you can manage AWS resources programmatically through either the AWS Command Line Interface (AWS CLI) or the AWS software development kits (AWS SDKs), as shown in Figure 1.2.

**FIGURE 1.2** Options for managing AWS resources

## AWS Software Development Kits

AWS SDKs are available in many popular programming languages such as Java, .NET, JavaScript, PHP, Python, Ruby, Go, and C++. AWS also provides specialty SDKs such as the AWS Mobile SDK and AWS Internet of Things (IoT) Device SDK.

Although the instructions for installing and using an AWS SDK vary depending on the operating system and programming language, they share many similarities. In this chapter, the examples are provided in Python.

The Python SDK for AWS is called AWS SDK for Python (Boto). If Python 2 or Python 3 is already installed on your machine, install boto3 using pip, the Python package manager.

To install boto3, open a terminal and run the following command:

```
pip install boto3 --upgrade -user
```

For documentation on the Python SDK, see <http://boto3.readthedocs.io/>.

For more information on SDKs for other programming languages or platforms, see <https://aws.amazon.com/tools/#sdk>.

## AWS CLI Tools

In addition to the AWS Management Console and SDKs, AWS provides tools to manage AWS resources from the command line. One such tool is the *AWS CLI*, which is available on Windows, Linux/Unix, and macOS.

The AWS CLI allows you to perform actions similar to those from the SDKs but in an interactive scripting environment. Because the AWS CLI is interactive, it is a good environment for experimenting with AWS features. Also, the AWS CLI and the SDK on the same server can share configuration settings.

If you prefer to manage your resources using PowerShell, use the AWS Tools for PowerShell instead of the AWS CLI. Other specialty command line tools are also provided, such as the Elastic Beanstalk command line interface and AWS SAM Local. For more information about these tools and installation, see <https://aws.amazon.com/tools/#cli>.

# Calling an AWS Cloud Service

The functionality of AWS is powered by web services that are agnostic to the programming language and SDK. In this section, you use the AWS Python SDK to make an API request.

This is an overview of both making an API call and the parameters to configure the SDK. Subsequent sections will describe those parameters.



Locate the API reference documentation about the underlying web services and programming language-specific documentation for each SDK at <https://aws.amazon.com/documentation>.

## API Example: Hello World

In the following example, you will make a request to Amazon Polly. *Amazon Polly* provides text-to-speech service with natural-sounding speech, and it is able to provide speech in multiple languages with a variety of male and female voices. Furthermore, you can modify attributes, such as pronunciation, volume, pitch, or speed, by defining *lexicons* or supplying *Speech Synthesis Markup Language* (SSML).

This Python code example uses the AWS SDK for Python (Boto) and Amazon Polly to generate an audio clip that says, “Hello World.”

```
import boto3

#Explicit Client Configuration
polly = boto3.client('polly',
 region_name='us-west-2',
 aws_access_key_id='AKIAI05F0DNN7EXAMPLE',
 aws_secret_access_key='ABCDEF+c2L7yXeGvUyrPgYsDnWRRC1AYEXAMPLE'
)

result = polly.synthesize_speech(Text='Hello World!',
 OutputFormat='mp3',
 VoiceId='Aditi')

Save the Audio from the response
audio = result['AudioStream'].read()
with open("helloworld.mp3","wb") as file:
 file.write(audio)
```

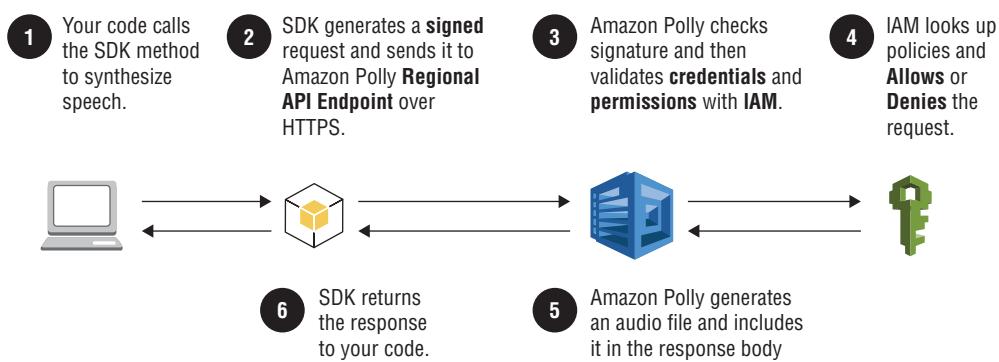
The AWS SDK maps the function call to an HTTPS request to an Amazon Polly API endpoint that is determined by the region name (`region_name`) parameter.

The SDK also adds authorization information to your request by signing the request using a key derived from the AWS secret access key.

When your request is received at the Amazon Polly API endpoint, AWS authenticates the signature and evaluates *AWS Identity and Access Management* (IAM) policies to authorize the API action.

If authorization succeeds, Amazon Polly processes the request, generates an MP3 audio file, and then returns it to the SDK client as part of the response to the HTTPS request, as shown in Figure 1.3.

**FIGURE 1.3** API request and authorization



## API Requests

Examine the request that is being transmitted in step 2 of Figure 1.3. When the SDK makes the request to Amazon Polly, it submits a JSON body using a standard HTTP POST to <https://polly.us-west-2.amazonaws.com/v1/speech>.

The SDK sets the following properties in the request:

```
POST /v1/speech HTTP/1.1
host: polly.us-west-2.amazonaws.com
content-Type: application/json
x-amz-date: 20180411T051402Z
authorization: AWS4-HMAC-SHA256 Credential=AKIAI05FODNN7EXAMPLE/20180411/
us-west-2/polly/aws4_request, SignedHeaders=content-length;content-type;host;x-
amz-date, Signature=d968197e88a6a8de69d1a7bcab414669eecd5f841e13dc90e4a7852
c2c428038

{
 "OutputFormat": "mp3",
 "Text" : "Hello World!",
 "VoiceId": "Aditi"
}
```

Notice that the API endpoint or host URL includes the AWS Region parameter (`us-west-2`). The SDK also generates an authorization header by taking in the AWS access key credentials and applying the AWS Signature Version 4 signing process to the request.

## API Responses

The API response corresponds to step 5 of Figure 1.3. For the example API request, the following headers are set in the response:

```
HTTP/1.1 200
status: 200
content-type: audio/mpeg
date: Wed, 11 Apr 2018 05:14:02 GMT
x-amzn-requestcharacters: 12
x-amzn-requestid: 924141bb-b0a6-11e8-b565-b1fabccdc9
transfer-encoding: chunked
connection: keep-alive
```

The headers include a standard HTTP response code. In this case, the status is 200. In general, AWS responds with standard HTTP response codes, such as the following:

- HTTP/200, if successful
- HTTP/403, if authorization is denied

You can also use an *x-amzn-requestid* header to troubleshoot when contacting AWS Support. The response body includes the audio stream; in this case, the audio stream is in MP3 format.

The AWS SDK wraps the web response and returns an object to the application. This step corresponds to step 6 of Figure 1.3. If the HTTP status code is not HTTP/200, the SDK generates an exception that your code can handle.



The *AWS Signature Version 4* signing process incorporates the current date into the process to sign API requests, so make sure that the clock on the computer making API requests is accurate. AWS API requests must be received within 15 minutes of the timestamp in the request to be valid.

## SDK Configuration

In the previous example, the AWS Region and AWS credentials are provided explicitly in the code. The SDK client initialization code from the earlier example is shown again here:

```
Explicit Client Configuration
polly = boto3.client('polly',
 region_name='us-west-2',
 aws_access_key_id='AKIAI05FODNN7EXAMPLE',
 aws_secret_access_key='ABCDEF+c2L7yXeGvUyrPgYsDnWRRC1AYEXAMPLE'
)
```

This explicit approach of hardcoding credentials into the code is not recommended, because it carries the risk of checking the credentials into a source-control repository. This would expose the keys to everyone who has access to the repository and could even result in public disclosure. To prevent this, configure the SDK credentials separately from the application source code.

The SDK and AWS CLI automatically check several locations for credentials, and for the region if they are not explicitly provided in the code. These locations include environment variables, programming language-specific parameter stores, and local files.

To configure an AWS access key on your local machine in a local file, create a credentials file in the `.aws` folder in the home folder for the current user. Within this file, specify credentials for the default profile. You may optionally include additional named profiles beyond the default as needed.

```
[default]
aws_access_key_id=AKIAIO5FODNN7EXAMPLE
aws_secret_access_key=ABCDEF+c2L7yXeGvUyrPgYsDnWRRC1AYEXAMPLE

From File: ~/.aws/credentials
```

Furthermore, hardcoding the AWS Region into the code makes it difficult to deploy your application in different AWS Regions. Instead, create a config file also within the `.aws` folder within your current user's home directory. Within this file, specify a region to use with the default profile.

```
[default]
region = us-west-2

From File: ~/.aws/config
```

As an alternative to creating the credentials and config files manually, you can use the AWS CLI to generate the credentials and config files for the default profile as follows:

```
aws configure
```

This command prompts for *credentials* and *region* settings. When the command completes, the config and credentials files are generated, as shown in Figure 1.4.

**FIGURE 1.4** Configuring API credentials

```
heiwad@surface:~$ aws configure
AWS Access Key ID [None]: AKIAIZY26EJTDEXAMPLE
AWS Secret Access Key [None]: EXAMPLEX0I9ck0717hgxfzbubLGH3MEck4dbCQf
Default region name [None]: us-east-1
Default output format [None]:
heiwad@surface:~$ ls ~/.aws
config credentials
heiwad@surface:~$
```

When the configuration is complete, replace this snippet of code:

```
Explicit Client Configuration
polly = boto3.client('polly',
 region_name='us-west-2',
 aws_access_key_id='AKIAIO5FODNN7EXAMPLE',
 aws_secret_access_key='ABCDEF+c2L7yXeGvUyrPgYsDnWRRC1AYEXAMPLE'
)
```

with this line of code:

```
Implicit Client Configuration
polly = boto3.client('polly')
```

By separating your code from the credentials, you make it easier to collaborate with other developers while making sure that your credentials are not inadvertently disclosed to others.

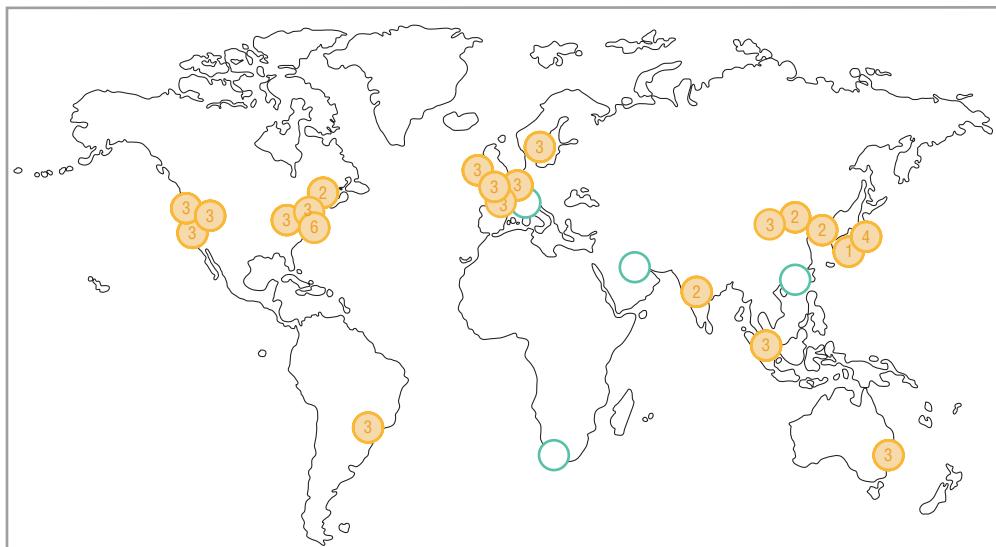


For code running on an AWS compute environment, such as Amazon Elastic Compute Cloud (Amazon EC2) or AWS Lambda, instead of using local files, assign an IAM role to the environment. This enables the SDK to load the credentials automatically from the role and to refresh the credentials as they are automatically rotated.

## Working with Regions

Now take a closer look at what it means to configure the AWS SDK with an *AWS Region*. AWS operates facilities in multiple regions across the world, as shown in Figure 1.5. Each AWS Region is located in a separate geographic area and maintains its own, isolated copies of AWS services. For many AWS services, you are required to select a specific region to process API requests and in which to provision your resources.

**FIGURE 1.5** AWS Regions, Availability Zones, and planned regions (as of February 2019)



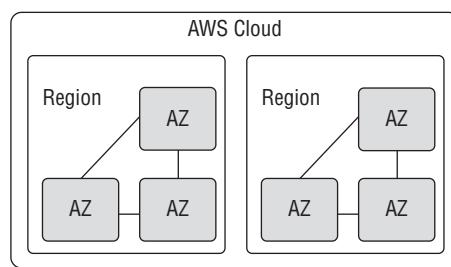
Customers expect that their data is durably held and that services remain highly available. In this section, you will explore how the structure of a region lends itself to providing reliable service and how to choose an appropriate region for your application.

## Regions Are Highly Available

Each *AWS Region* contains multiple data centers, grouped together to form *Availability Zones*. Regions are composed of multiple Availability Zones, which allows AWS to provide highly available services in a way that differentiates them from traditional architectures with single or multiple data centers.

Availability Zones are physically separated from each other and are designed to operate independently from each other in the case of a fault or natural disaster, as shown in Figure 1.6. Even though they are physically separated, Availability Zones are connected via low-latency, high-throughput redundant networking.

**FIGURE 1.6** Regions and Availability Zones



AWS customers can improve the resilience of their applications by deploying a copy of each application to a second Availability Zone within the same region. This allows the application to remain available to customers even in the face of events that could disrupt an entire data center. Similarly, many of the AWS services automatically replicate data across multiple Availability Zones within an AWS Region to provide high availability and durability of the data.

An example of an AWS service that replicates data across Availability Zones within a region is Amazon Simple Storage Service (Amazon S3). Amazon S3 enables you to upload files and store those files as objects within a bucket. By default, Amazon S3 automatically replicates objects across a minimum of three Availability Zones within the region hosting the bucket. This design protects data even against the loss of one entire Availability Zone.

## Working with Regional API Endpoints

Many AWS services expose regional API endpoints. When making web service calls to regional endpoints, the region can typically be identified in the URL that you invoke. API calls to a regional endpoint usually affect only the resources within the specific AWS Region that corresponds to that endpoint.

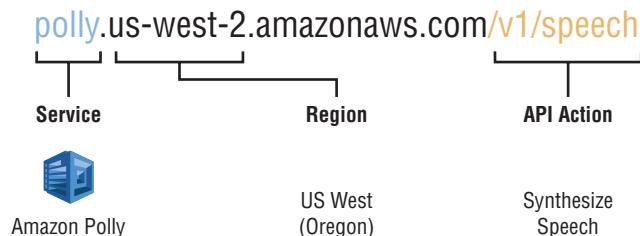
To explore this concept, revisit the previous example of making a request to Amazon Polly to synthesize speech from text.

```
Initializing SDK Client with Explicit Region Configuration
polly = boto3.client('polly', region_name='us-west-2')
result = polly.synthesize_speech(Text='Hello World!',
 OutputFormat='mp3',
 VoiceId='Aditi')
```

To explicitly configure the AWS SDK to use the US West (Oregon) Region, set the `region_name` parameter to `us-west-2` when initializing the SDK client, as in the previous example.

This configuration results in the SDK computing the following URL for the API request, as shown in Figure 1.7.

**FIGURE 1.7** A regional API endpoint and API action



You can see regional isolation in practice by uploading a lexicon to Amazon Polly. A *lexicon* stores custom pronunciation information that can be used when synthesizing speech from text. For example, you can require Amazon Polly to expand the acronym AWS to “Amazon Web Services” in the generated audio file by providing the following XML lexicon. The file tells Amazon Polly to speak the *alias* “Amazon Web Services” when it encounters the *grapheme* “AWS” in text.

```
<?xml version="1.0" encoding="UTF-8"?>
<lexicon version="1.0"
 xmlns="http://www.w3.org/2005/01/pronunciation-lexicon"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.w3.org/2005/01/pronunciation-lexicon
 http://www.w3.org/TR/2007/CR-pronunciation-lexicon-20071212/pls.xsd"
 alphabet="ipa"
 xml:lang="en-US">
<lexeme>
 <grapheme>AWS</grapheme>
 <alias>Amazon Web Services</alias>
</lexeme>
</lexicon>
File: aws-lexicon.xml
```

To use this lexicon when you synthesize speech, you must first upload it to Amazon Polly. The following shell snippet uses the AWS CLI to upload to the lexicon to the specified region:

```
aws polly put-lexicon --name awsLexicon --content file://aws-lexicon.xml
--region us-west-2
```

You can use the `awsLexicon` after it is uploaded. The following example generates a speech request that will be customized by the lexicon. This request is also being made to the `us-west-2` API endpoint.

```
Synthesizing speech with custom lexicon in the same region
aws polly synthesize-speech --text 'Hello AWS World!' --voice-id Joanna
--output-format mp3 hello.mp3 --lexicon-names="awsLexicon" --region us-west-2
{
 "ContentType": "audio/mpeg",
 "RequestCharacters": "15"
}
```

Assuming that the CLI is configured correctly with an appropriate access key, this request succeeds. In the downloaded audio file, you will hear Joanna say “Hello Amazon Web Services World,” confirming that the lexicon is in effect.

However, if you run the same API request again, but change the region to the US East (N. Virginia) Region, as in the following example, you will get a different result:

```
Trying again against a different Regional API endpoint
aws polly synthesize-speech --text 'Hello AWS World' --voice-id Joanna --output-
format mp3 hello-custom.mp3 --lexicon-names="awsLexicon" --region us-east-1
```

An error occurred (`LexiconNotFoundException`) when calling the `SynthesizeSpeech` operation: Lexicon not found

In this case, an error occurs because the `awsLexicon` resides only in the US West (Oregon) Region where you placed it. When working with AWS services that are regional in scope, you are in control over where the data resides—AWS does not automatically copy your data for these services to other regions without an explicit action on your part. If you must use the lexicon in regions other than US West (Oregon), you could upload the lexicon to each region in which you plan to use it.

## Identifying AWS Regions

When working with AWS services, the AWS Management Console refers to regions differently from the parameters used in the AWS CLI and SDK.

Table 1.1 lists several region names and the corresponding parameters for the AWS CLI and SDK.

**TABLE 1.1** Sample of Region Names and Regions

| Region Name              | Region         |
|--------------------------|----------------|
| US East (N. Virginia)    | us-east-1      |
| US West (Oregon)         | us-west-2      |
| EU (Frankfurt)           | eu-central-1   |
| EU (London)              | eu-west-2      |
| EU (Paris)               | eu-west-3      |
| Asia Pacific (Tokyo)     | ap-northeast-1 |
| Asia Pacific (Mumbai)    | ap-south-1     |
| Asia Pacific (Singapore) | ap-southeast-1 |

There are other AWS services, such as *IAM*, that are not limited to a single region. When you interact with these services in the console, the region selector in the upper-right corner of the console displays “Global.” The API endpoint for IAM is the same regardless of the region. Table 1.2 lists some API endpoints.

**TABLE 1.2** Selected IAM Service API Endpoints

| Region Name             | API Endpoint      |
|-------------------------|-------------------|
| US East (N. Virginia)   | iam.amazonaws.com |
| US East (Ohio)          | iam.amazonaws.com |
| US West (N. California) | iam.amazonaws.com |

In the case of IAM, having IAM resources available in multiple regions is a useful strategy. IAM provides a way to create API credentials, and this means you can use the same set of API credentials to access resources in different AWS Regions.

For each AWS service, you can find the regions in which that service is available, along with the corresponding API endpoints, in the AWS General Reference documentation. The following link provides a comprehensive list of AWS services and their regional API endpoints: <https://docs.aws.amazon.com/general/latest/gr/rande.html>.



The exam may ask you to identify a URL or endpoint for an AWS resource, such as an Amazon S3 bucket, that has been deployed to a specific region. While the test does not require memorization of the region list, AWS recommends that you become familiar with the naming convention for regions and how it is related to the naming convention for Availability Zones.

## Choosing a Region

One factor for choosing an AWS Region is the availability of the services required by your application. Other aspects to consider when choosing a region include latency, price, and data residency. Table 1.3 describes selection criteria to include when choosing an AWS Region.

**TABLE 1.3** Selecting an AWS Region

| Selection Criteria    | Description                                                                                                                                                                                                                                       |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Service availability  | Choose a region that has all or most of the services you intend to use. Each region exposes its own AWS Cloud service endpoints, and not all AWS services are available in all regions.                                                           |
| Proximity and latency | Choose a region closer to application users, on-premises servers, or your other workloads. This allows you to decrease the latency of API calls.                                                                                                  |
| Data residency        | Choose a region that allows you to stay compliant with regulatory or contractual requirements to store data within a specific geographic region.                                                                                                  |
| Business continuity   | Choose a pair of regions based on any specific requirements regarding data replication for disaster recovery. For example, you may select a second AWS Region as a target for replicating data based on its distance from the primary AWS Region. |
| Price                 | AWS service prices are set per region. Consider cost when service availability and latency are similar between candidate regions.                                                                                                                 |

## API Credentials and AWS Identity and Access Management

Now that you have seen how to make API calls and identified the infrastructure provided by the AWS Cloud, take a closer look at the access keys needed to make API calls. In AWS, an *access key* is a type of security credential that is associated with an identity. So, to make API calls, first you will create an identity in AWS Identity and Access Management (IAM).

To manage authentication and authorization for people or applications, IAM provides users, groups, and roles as identities that you can manage. IAM authenticates the security credentials used to sign an API call to verify that the request is coming from a known identity. Then, IAM authorizes the request by evaluating the policies associated with the identity and resources affected by the request. This section provides reviews users, groups, roles, and policies.



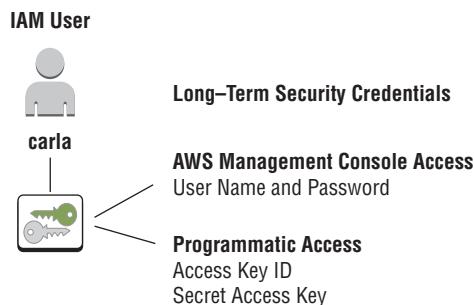
When you first create an account and sign in with your email address and password, you are authenticating as the root user for your account. Few AWS operations require root user permissions. To protect your account, do not generate an access key based on the root user. Instead, create an IAM user and generate an access key for that user. To provide administrator access, add that user to a group that provides administrator permissions.

## Users

IAM users can be assigned long-term security credentials. You might create an IAM user when you have a new team member or application that needs to make AWS API calls. Manage the API permissions of the user by associating permissions policies with the user or adding the user to a group that has permissions policies associated with it.

After you create an IAM user, you can assign credentials to allow AWS Management Console access, programmatic access, or both, as shown in Figure 1.8.

**FIGURE 1.8** IAM user long-term credentials



### AWS Management Console Access

To sign in to the console, IAM users authenticate with an IAM user name and password. As part of the sign-in process, IAM users are prompted to provide either the account ID or alias so that IAM user names only need to be unique within your account. If *multi-factor authentication* (MFA) is enabled for an IAM user, they must provide their MFA code when they attempt to sign in.



To simplify sign-in, use the special sign-in link in the IAM dashboard that prefills the account field in the console sign-in form.

## AWS IAM User API Access Keys

For *programmatic access* to AWS, create an access key for the IAM user. An *AWS access key* is composed of the following two distinct parts:

1. Access key ID
2. Secret access key

Here is an example of an AWS access key:

```
aws_access_key_id = AKIAJXR7IOGGTEIVNX7Q
aws_secret_access_key: oe/H0e2Ptj/fvwr dj6Wedo43Vsm05DHDADZ+tnP5
```

Each user may have up to two active access keys at any time. These access keys are *long-term* credentials and remain valid until you explicitly revoke them.



Given the importance of the secret access key, you can view or download it only once. If you forget the secret access key, create a new access key and then revoke the earlier key.

## Other Credentials for IAM Users

In addition to passwords, multifactor devices, and access keys, IAM users can have other types of security credentials. You can have X.509 certificates, which are used with SOAP APIs, or you can have GIT credentials as either Secure Shell (SSH) keys or passwords to interact with the AWS CodeCommit service.

## Groups

To help you manage the permissions of collections of IAM users, IAM provides *IAM groups*. IAM groups do not have their own credentials, but when an IAM user makes an API call with their access key, AWS looks up that user's group memberships and finds the relevant permissions policies. Associate users who need the same permissions with a group and then assign policies to the group instead of associating the permissions directly to each user.

For example, all developers working on a specific project could each have their own IAM user. Each of these users can be added to a group, named *developers*, to manage their permissions collectively. In this way, each team member has unique credentials while they are also given the same permissions.

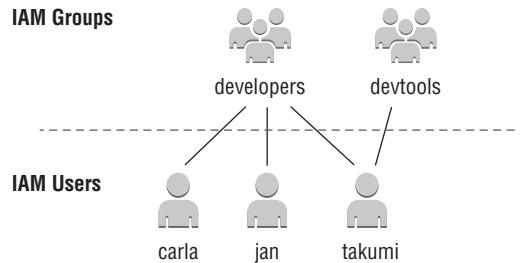
You may create additional groups. For example, you may create a second group for the team members responsible for changing the build and deployment pipeline to which you can assign a name such as *devtools*.

The relationship between IAM users and IAM groups is *many-to-many*. An individual IAM user can be a member of many IAM groups, and each IAM group can have many

IAM users associated with the group. IAM users within an IAM group inherit permissions from the policies attached to their group, plus any permissions from policies that are associated directly with that IAM user.

In the example shown in Figure 1.9, *carla* inherits permissions from the IAM user *carla* and from the group *developers*, and *takumi* inherits the union of all of the policies from *developers* and from *devtools*, in addition to any policies directly associated with *takumi*.

**FIGURE 1.9** IAM groups and IAM users



In the case that multiple permissions policies apply to the same API action, any policy that has the effect *deny* will take precedence over any policy that has the effect *allow*. This order of precedence is applied regardless of whether the policies are associated with the user, group, or resource.

## Roles

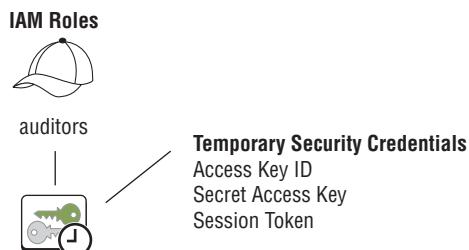
There are situations in which you might not want to create and manage new sets of long-term credentials for team members or applications.

In a large company with many employees, you can use your existing corporate identity store instead of creating new identities and credentials for each team member who manages AWS.

Alternatively, you may delegate permissions to an AWS service to perform actions on your behalf. One common example of this is when application code running on an AWS *compute* service, such as Amazon EC2, needs permissions to make AWS API calls. In this case, AWS recommends allowing Amazon EC2 to manage the credentials for each instance.

In both situations, rather than creating new IAM users, create an *IAM role* to assign permissions. IAM roles can be assumed for short-term sessions, as shown in Figure 1.10.

**FIGURE 1.10** IAM roles



To control access to an IAM role, define a *trust policy* that specifies which *principals* can assume a role. Potential principals include AWS services and also users who have authenticated using identity federation. Principals could also include users who authenticate with web identity federation, IAM users, IAM groups, or IAM roles from *other* accounts.

This example trust policy allows Amazon EC2 to request short-term credentials associated with an IAM role:

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Principal": {
 "Service": "ec2.amazonaws.com"
 },
 "Action": "sts:AssumeRole"
 }
]
}
```

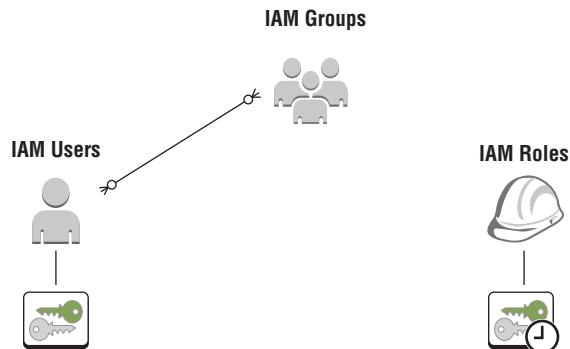
When a principal assumes a role, AWS provides new short-term security credentials that are valid for a time-limited session through the *AWS Security Token Service* (AWS STS). These credentials are composed of an access key ID, secret access key, and, additionally, a session token with a known expiration date.

This example displays the credentials that are generated when the role is assumed:

```
{
 "AccessKeyId": "ASIAJHP2KG65VIKQU2XQ",
 "SecretAccessKey": "zkvPEbYxCLVVD0seWdRnesc8krNDPHEX1cFMyI5W",
 "SessionToken": "FQoDYXdzEMf//////////wEaDL1b0Wd7VTA3J25cNyL4ARzNSRczH4U3f8gJwi1W8XiDLWJIE9EdX
4l4KXTiST40gPoWc9Do9QkcN2xRHk6/qVT6W23d0u6+5YFY9C2wnoEeTTmiQBT5SMjqku5MYlhrCDy
FQAVbo6RKUe0ZXXSG8REshuFGBtaCnmv95lFF6srCT1b4FZtTtULE7WV3LMcDs6Z2XuN+6aGTawhY5
0RMnlKRL1w6yHq++RysQwbBHkuNeK/VqjueDINFODP0je9ZnYePVjR5uLmL8ZARWYVBFrB2tpxG07/
dseUS902q1hMP8DJuEfSbaik2ASsmXSRA8v0Znuu4AsBq6ERasBw5EcpICP/Ne8zdKO/93tYF",
 "Expiration": "2018-04-18T22:55:59Z"
}
```

When these short-term credentials are used, AWS looks up the permissions policies associated with the IAM role that was assumed. This is true even if the principal that assumed the role was an IAM user—policies that were associated with the IAM user or their groups are not evaluated when the role credentials are used to make a request. The IAM role is a distinct identity with its own permissions. Furthermore, you cannot nest IAM roles or add IAM roles to IAM groups, as shown in Figure 1.11.

**FIGURE 1.11** IAM roles are distinct from IAM users and groups.



## Choosing IAM Identities

Consider the following to determine how to define authorization and authentication.

### Scenario: During Development

IAM users can be a convenient way to share access to an account with your team members or for application code that is running locally. The associated long-term credentials are easy to work with on a local development laptop, or on other hardware in your control, such as on-premises servers. To manage the permissions of collections of IAM users more simply, add those users to IAM groups.

### Scenario: When Deploying Code to AWS

Use IAM roles. AWS compute services can be configured to distribute and rotate the role credentials automatically on your behalf, making it easier for you to manage credentials securely.

### Scenario: When You Have an Existing External Identity Provider

When you have an external identity provider, such Active Directory, use IAM roles. That way, team members can use the single sign-on they already use to access AWS without needing to remember an extra password. Also, if a team member leaves, you disable their corporate access in only one place—the external directory.

Use roles in cases in which you need to make AWS API calls from untrusted machines because role credentials automatically expire. For example, use IAM roles for client-side code that must upload data to Amazon S3 or interact with Amazon DynamoDB.

Table 1.4 describes the use cases for IAM identities.

**TABLE 1.4** IAM Users and IAM Roles Usage

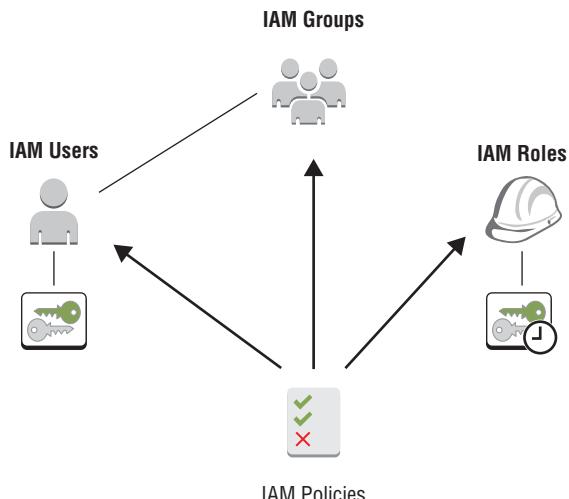
| For Code Running on...                                     | Suggestion |
|------------------------------------------------------------|------------|
| A local development laptop or on-premises server           | IAM user   |
| An AWS compute environment such as Amazon EC2              | IAM role   |
| An IAM user mobile device                                  | IAM role   |
| Enterprise environments with an external identity provider | IAM role   |



The exam tests your knowledge of the recommended practices for distributing AWS credentials to your code depending on where that code is running.

## Managing Authorization with Policies

Manage the permissions for each user, group, or role by assigning *IAM policies* that either *allow* or *deny* permissions to specific API actions, as shown in Figure 1.12. Any API action is *implicitly denied* unless there is a policy that explicitly allows it. If there is a policy that *explicitly denies* an action, that policy always takes precedence. In this way, AWS defaults to secure operation and errs on the side of protecting the resources in cases where there are conflicting policies.

**FIGURE 1.12** IAM policies and IAM identities

One method of granting permissions is to use AWS managed policies. AWS provides these policies to support common tasks and are automatically updated as new services and API operations are added.

When choosing permissions policies, AWS recommends that you adopt the principle of *least privilege* and grant someone the minimum permissions they need to complete a task. If they need more access later, they can ask for it, and you can update the permissions then.

Take the example of an application that uses Amazon Polly. If the application uses only Amazon Polly to synthesize speech, use the *AmazonPollyReadOnlyAccess* policy, which grants permissions to Amazon Polly actions that do not store any data or modify data stored in AWS. The policy is represented as a JSON document and shown here:

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "polly:DescribeVoices",
 "polly:GetLexicon",
 "polly>ListLexicons",
 "polly:SynthesizeSpeech"
],
 "Resource": [
 "*"
]
 }
]
}
```

If the application needs permission to upload (or delete) a custom lexicon, this operation modifies a state in Amazon Polly. To grant permissions to these actions, use the *AmazonPollyFullAccess* policy. The policy is shown here. Notice that the actions granted by the policy shown here are represented as "polly:\*", where the \* provides access to all Amazon Polly API actions.

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "polly:/"
],
 "Resource": [
 "*"
]
 }
]
}
```

```
 """
]
}i
]
}
```

## Custom Policies

AWS recommends that you use the AWS managed policies whenever possible. However, when you need more control, you can define custom policies.

As shown in the earlier examples, an *IAM policy* is a JSON-style document composed of one or more statements. Each statement has an effect that will either allow or deny access to specific API actions on AWS resources. A *deny statement takes precedence over any allow statements*. Use an *Amazon Resource Name* (ARN) to specify precisely the resource or resources to which a custom policy applies.

For example, the following policy authorizes access to the `DeleteLexicon` action in Amazon Polly on the resource specified by the ARN. In this case, the resource is a particular lexicon within a specific account and within a specific region.

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "AllowDeleteForSpecifiedLexicon",
 "Effect": "Allow",
 "Action": [
 "polly:DeleteLexicon"],
 "Resource": "arn:aws:polly:us-west-2:123456789012:lexicon/awsLexicon"
 }
]
}
```

To allow slightly broader permissions in a similar policy, use *wildcards* in the ARN. For example, to allow a user to delete any lexicon within the specified region and account, replace `awsLexicon` with an \* in the ARN, as shown here:

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "AllowDeleteSpecifiedRegion",
 "Effect": "Allow",
 "Action": [
 "polly:DeleteLexicon"],
```

```
 "Resource": "arn:aws:polly:us-east-2:123456789012:lexicon/*"
 }
]
}
```

An ARN always starts with arn: and can include the following components to identify a particular AWS resource uniquely:

**Partition** Usually aws. For some regions, such as in China, this can have a different value.

**Service** Namespace of the AWS service.

**Region** The region in which the resource is located. Some resources do not require a region to be specified.

**Account ID** The account in which the resource resides. Some resources do not require an account ID to be specified.

**Resource** The specific resource within the namespace of the AWS service. For services that have multiple types of resources, there may also be a resource type.

These are example formats for an ARN:

```
arn:partition:service:region:account-id:resource
arn:partition:service:region:account-id:resourcetype/resource
arn:partition:service:region:account-id:resourcetype:resource
```

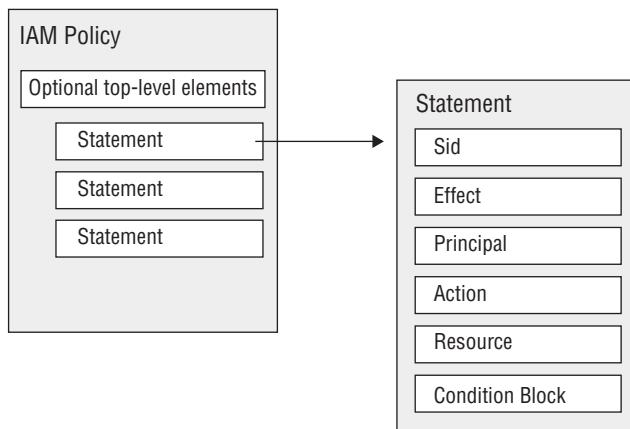
Here are some examples of ARNs for various AWS resources:

```
<!-- Amazon Polly Lexicon -->
arn:aws:polly:us-west-2:123456789012:lexicon/awsLexicon

<!-- IAM user name -->
arn:aws:iam::123456789012:user/carla
```

```
<!-- Object in an Amazon S3 bucket -->
arn:aws:s3:::bucket-name/exampleobject.png
```

A single policy document can have multiple statements. Additional components to a statement may include an optional *statement ID* (Sid) and condition blocks to restrict when the policy applies. If the policy is attached to a resource rather than to an IAM identity, then the policy must also specify a principal (to whom the policy applies), as shown in Figure 1.13.

**FIGURE 1.13** IAM policy elements

Write custom policies manually or use tools like the *Visual Policy Editor* in the AWS Management Console to generate policies more easily. To help you test the effects of policies, you can also use the IAM policy simulator at <https://pollicysim.aws.amazon.com>.

## Summary

In this chapter, you learned about the AWS Management Console, the AWS CLI, and the AWS SDKs that AWS uses to configure and manage your resources. You learned how to make API request calls to the AWS Cloud, use configuration files, select an AWS Region, manage AWS API credentials, and identify regional API endpoints. The chapter also discussed AWS account root users, IAM, IAM policies, IAM groups, IAM roles, long-term and short-term credentials, the access key ID, and the secret access key.

## Exam Essentials

**Know the ways to manage AWS resources.** Recall that the AWS SDK, AWS CLI, and the AWS Management Console are options for managing the AWS resources within your account.

**Know the importance of AWS Regions.** Be able to identify the impact of AWS Region selection on your application code, such as the relationship between region selection and user latency. Also recognize how region selection impacts API calls and API endpoints.

**Know about IAM users and IAM roles.** Know when it is appropriate to use IAM users or IAM roles for a given application that needs to make AWS API calls.

**Know how to recognize valid IAM policies.** Identify valid IAM policies and predict the effects of policy statements.

# Resources to Review

AWS Free Tier:

<https://aws.amazon.com/free>

Getting Started Resource Center: Create an AWS Account:

<https://aws.amazon.com/getting-started>

Tracking Your Free Tier Usage:

<https://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/tracking-free-tier-usage.html>

AWS Documentation:

<https://aws.amazon.com/documentation>

AWS Management Console:

<https://signin.aws.amazon.com/console>

AWS Command Line Interface (AWS CLI):

<https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-welcome.html>

SDKs, Toolkits, and Installation Directions:

<https://aws.amazon.com/tools/#sdk>

Amazon Polly:

<https://docs.aws.amazon.com/polly/latest/dg/what-is.html>

AWS Regions and Regional API Endpoints:

<https://docs.aws.amazon.com/general/latest/gr/rande.html>

AWS IAM Documentation:

<https://docs.aws.amazon.com/IAM/latest/UserGuide/getting-started.html>

AWS IAM Best Practices:

<https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html>

AWS IAM FAQs:

<https://aws.amazon.com/iam/faqs/>

AWS Signature Version 4 Signing Process:

<https://docs.aws.amazon.com/general/latest/gr/signature-version-4.html>

AWS Whitepapers (Introduction to AWS):

<https://aws.amazon.com/whitepapers/>

AWS Training and Certification:

<https://aws.amazon.com/training>

AWS Events and Webinars:

<https://aws.amazon.com/about-aws/events>

AWS Glossary:

<https://docs.aws.amazon.com/general/latest/gr/glos-chap.html>

# Exercises

## EXERCISE 1.1

### Sign Up for an Account

In this exercise, you'll sign up for an account.

1. Open your browser and go to <https://aws.amazon.com/free/>.
2. Choose **Create a Free Account**.
3. Provide personal information.
4. Provide payment Information.
5. Verify your phone number.
6. Select a support plan.
7. Choose **Sign in to the Console**.
8. Sign in to the console.

You are now signed in to the AWS Management Console.

---

## EXERCISE 1.2

### Create an IAM Administrators Group and User

In this exercise, you'll define an Administrators group and then add a user to that group. Generate API keys for this user and call this user DevAdmin.

1. Sign in to the AWS Management Console (at [signin.aws.amazon.com/console](https://signin.aws.amazon.com/console)).
  2. Select **All Services**.
  3. To open the IAM dashboard, select **IAM**.
-

4. To view the list of IAM groups, select **Groups**.  
If this is a new account, the list is empty.
5. Choose **Create New Group**.
6. For **Group Name**, enter **Administrators**.
7. Choose **Next Step**.
8. On the **Attach Policy** page, select the **AdministratorAccess** policy.
9. Choose **Next Step**.
10. On the **Review** page, choose **Create Group** to create the Administrators group.
11. To view the list of IAM users, select **Users**.  
If this is a new account, the list is empty.
12. Choose **Add user**.
13. Set the user name to **DevAdmin**.
14. Select both Access type check boxes: **Programmatic access** and **AWS Management Console access**.
15. Choose **Next: Permissions**.
16. To add this user to the Administrators group, select the **Administrators group** check box.
17. Clear the **Require password reset** check box.
18. Choose **Next: Tags**.
19. Provide a tag with a key of **project** and a value of **dev-study-guide**.  
Use tags to add customizable key-value pairs to resources so that you can more easily track and manage them.
20. Choose **Next: Review**.
21. Choose **Create user**.
22. Download the **credentials.csv** file.
23. Rename the file to **devadmin-credentials.csv**, and move the file to a folder where you would like to keep it.
24. Sign out of the AWS Management Console by clicking your name in the top bar and selecting **Sign Out**.

You now have a .csv file that contains a user name, password, access key ID, secret access key, and console login link. Use the DevAdmin user name, password, and console sign-in link to sign in to the AWS Management Console for all future exercises unless otherwise noted. Use the access key to configure the SDK in the following exercises.

---

**EXERCISE 1.3****Install and Configure the AWS CLI**

In this exercise, you'll install and configure the AWS Command Line Interface (AWS CLI). The AWS CLI requires Python2 or Python3. Install Python using pip, the Python installer.

1. Install Python from <https://www.python.org/downloads/>.
2. Open a terminal window.
3. To install the AWS CLI, run the following command:  
`pip install aws-cli --upgrade --user`
4. (Optional) If you encounter issues with step 3, review the AWS CLI Installation guide for alternative installation options here:  
<https://docs.aws.amazon.com/cli/latest/userguide/installing.html>
5. To configure the AWS CLI with a default profile for credentials, run the following command:  
`aws configure`
6. Enter the following values when prompted:
  - AWS Access Key ID: Paste the value from the CSV you downloaded in Exercise 1.2.
  - AWS Secret Access Key: Paste the value from the CSV you downloaded in Exercise 1.2.
  - Default region name: Enter us-east-1.
  - Default output format: Press Enter to leave this blank.
7. Run the CLI command to verify that your CLI is working correctly, and view the available voices for Amazon Polly.  
`aws polly describe-voices --language en-US --output table`

A table in the terminal lists the available voices for Amazon Polly for the language US English.

**EXERCISE 1.4****Download the Code Samples**

In this exercise, you'll download the code snippets to execute future exercises.

1. If you do not already have Git installed, install it from <https://git-scm.com/downloads>.
2. Open a command terminal.

3. Create a folder on the hard drive to store the examples.
4. Navigate to a folder to host the code.
5. To download the samples using Git, run the following command:

```
git clone http://example.com/example.git
```

A folder named <>example<> contains the code examples for this study guide.

---

### EXERCISE 1.5

#### Run a Python Script that Makes AWS API Calls

In this exercise, you'll run the Python script to make an AWS API call.

1. Open a terminal window and navigate to the folder with the book sample code.

2. To install the AWS SDK for Python (Boto), run the following command:

```
pip install boto3
```

3. Navigate to the chapter-01 folder where you downloaded the sample code.

4. To generate an MP3 in the chapter-01 folder, run the helloworld.py program.

```
python helloworld.py
```

5. To hear the audio, open the generated file, helloworld.mp3.

6. (Optional) Modify the Python code to use a different voice. See Exercise 1.3 for an AWS CLI command that provides the list of available voices.

You hear “Hello World” when you play the generated audio file. If you completed the optional challenge, you also hear the audio spoken in a different voice from the first audio.

---

### EXERCISE 1.6

#### Working with Multiple Regions

In this exercise, you'll use Amazon Polly to understand the effects of working with different AWS Regions.

1. Open a terminal window and navigate to the folder with the book sample code.

2. Navigate to chapter-01 in the folder where you downloaded the sample code.

3. Verify that the region is us-east-1 by running the following command:

```
aws configure get region
```

---

(continued)

**EXERCISE 1.6 (*continued*)**

4. Upload aws-lexicon.xml to the Amazon Polly service in the default region, which is US East (N. Virginia).

```
aws polly put-lexicon --name awsLexicon --content file://aws-lexicon.xml
```

5. The file helloaws.py is currently overriding the region to be EU (London). Run the Python code and observe the LexiconNotFoundException that returns.

```
python.helloaws.py
```

6. Upload the lexicon to EU (London) by setting the region to eu-west-2.

```
aws polly put-lexicon --name awsLexicon --content
file://aws-lexicon.xml --region eu-west-2
```

7. Run the following Python script again:

```
python.helloaws.py
```

Observe that it executes successfully this time and generates an MP3 file in the current folder.

8. Play the generated helloaws.mp3 file to confirm that it says, “Hello Amazon Web Services.”

9. (Optional) Delete the lexicons with the following commands:

```
aws polly delete-lexicon --name awsLexicon
```

```
aws polly delete-lexicon --name awsLexicon --region eu-west-2
```

Even though the text supplied by the API call to synthesize speech was “Hello AWS!,” the generated audio file uses the lexicon you uploaded to pronounce it as “Hello Amazon Web Services.”

---

**EXERCISE 1.7****Working with Additional Profiles**

In this exercise, you define a limited user for the account and configure a new profile in the SDK to use these credentials. Notice that the permissions are restrictive and that you need to update the permissions for that user to be more permissive.

1. Sign in to the AWS Management Console (at [aws.amazon.com](https://aws.amazon.com)) using the credentials for DevAdmin from Exercise 1.2.
  2. Select **Services**.
  3. Select **IAM** to open the IAM dashboard.
  4. Select **Users** to view the list of IAM users.
-

5. Choose **Add user**.
6. Set the user name to **DevRestricted**.
7. For **Access type**, select **Programmatic access**.
8. Choose **Next Permissions**.
9. Select **Attach existing policies directly**.
10. Select the **AmazonPollyReadOnlyAccess** policy.
11. To narrow the options, in **Filter**, enter **polly**.
12. Choose **Next: Tags**.
13. Define a tag as follows:
  - Key: **project**
  - Value: **dev-study-guide**
14. Choose **Next: Review**.
15. Choose **Create User**.
16. To configure the SDK in the following steps, download the **credentials.csv** file.
17. Rename the downloaded file to **devrestricted-credentials.csv** and move it to the same folder where you put the CSV file from Exercise 1.2.
18. Open a terminal window and navigate to the folder with the sample code.
19. Navigate to the chapter-01 folder.
20. (Optional) Review the code in **upload-restricted.py**.
21. Configure the AWS CLI with a new profile called **restricted**. Run the following command:  

```
aws configure --profile restricted
```

When prompted, enter the following values:
  - AWS Access Key ID: Copy the value from the CSV you downloaded.
  - AWS Secret Access Key: Copy the value from the CSV you downloaded.
  - Default region name: Enter **us-east-1**.
  - Default output format: Press Enter to retain the default setting.
22. Upload the lexicon.  
The upload operation is expected to fail because of the restricted permissions associated with the profile specified in the script. Run the following Python script:  

```
python upload-restricted.py
```

---

*(continued)*

**EXERCISE 1.7 (*continued*)**

23. Return to the AWS Management Console for IAM, and in the left navigation, select **Users**.
24. To view a user summary page, select **DevRestricted user**.
25. Choose **Add permissions**.
26. Select **Attach existing policies directly**.
27. To filter out other policies, in the search box, enter **polly**, and select the **AmazonPollyFullAccess** policy.
28. Choose **Next: Review**.
29. Choose **Add permissions**.
30. Repeat step 22 to upload the lexicon.

The upload is successful. After the change in permissions, you did not have to modify the credentials. After a short delay, the new policy automatically takes effect on new API calls from DevRestricted.

31. Delete the lexicon by running the following command:

```
aws polly delete-lexicon --name awsLexicon --region eu-west-2
```

In this exercise, you have configured the SDK and AWS CLI to refer to a secondary credentials profile and have tested the distinction between the AWS managed IAM policies related to Amazon Polly. You have also confirmed that it is possible to change the permissions of an IAM user without changing the access key used by that user.

---

# Review Questions

1. Which of the following is typically used to sign API calls to AWS services?
  - A. Customer master key (CMK)
  - B. AWS access key
  - C. IAM user name and password
  - D. Account number
2. When you make API calls to AWS services, for most services those requests are directed at a specific endpoint that corresponds to which of the following?
  - A. AWS facility
  - B. AWS Availability Zone
  - C. AWS Region
  - D. AWS edge location
3. When you're configuring a local development machine to make AWS API calls, which of the following is the simplest secure method of obtaining an API credential?
  - A. Create an IAM user, assign permissions by adding the user to an IAM group with IAM policies attached, and generate an access key for programmatic access.
  - B. Sign in with your email and password, and visit My Security Credentials to generate an access key.
  - C. Generate long-term credentials for a built-in IAM role.
  - D. Use your existing user name and password by configuring local environment variables.
4. You have a large number of employees, and each employee already has an identity in an external directory. How might you manage AWS API credentials for each employee so that they can interact with AWS for short-term sessions?
  - A. Create an IAM user and credentials for each member of your organization.
  - B. Share a single password through a file stored in an encrypted Amazon S3 bucket.
  - C. Define a set of IAM roles, and establish a trust relationship between your directory and AWS.
  - D. Configure the AWS Key Management Service (AWS KMS) to store credentials for each user.
5. You have a team member who needs access to write records to an existing Amazon DynamoDB table within your account. How might you grant write permission to this specific table and only this table?
  - A. Write a custom IAM policy that specifies the table as the resource, and attach that policy to the IAM user for the team member.
  - B. Attach the DynamoDBFullAccess managed policy to the IAM role used by the team member.
  - C. Delete the table and recreate it. Permissions are set when the DynamoDB table is created.
  - D. Create a new user within DynamoDB, and assign table write permissions.

6. You created a Movies DynamoDB table in the AWS Management Console, but when you try to list your DynamoDB tables by using the Java SDK, you do not see this table. Why?
- A. DynamoDB tables created in the AWS Management Console are not accessible from the API.
  - B. Your SDK may be listing your resources from a different AWS Region in which the table does not exist.
  - C. The security group applied to the Movies table is keeping it hidden.
  - D. Listing tables is supported only in C# and not in the Java SDK.
7. You make an API request to describe voices offered by Amazon Polly by using the AWS CLI, and you receive the following error message:

Could not connect to the endpoint URL:

<https://polly.us-east-1.amazonaws.com/v1/voices>

What went wrong?

- A. Your API credentials have been rejected.
  - B. You have incorrectly configured the AWS Region for your API call.
  - C. Amazon Polly does not offer a feature to describe the list of available voices.
  - D. Amazon Polly is not accessible from the AWS CLI because it is only in the AWS SDK.
8. To what resource does this IAM policy grant access, and for which actions?

```
{
 "Version": "2012-10-17",
 "Statement": [
 {"Effect": "Allow",
 "Action": "s3>ListBucket",
 "Resource": "arn:aws:s3:::example_bucket"}]}
}
```

- A. The policy grants full access to read the objects in the Amazon S3 bucket.
  - B. The policy grants the holder the permission to list the contents of the Amazon S3 bucket called example\_bucket.
  - C. Nothing. The policy was valid only until October 17, 2012 (2012-10-17), and is now expired.
  - D. The policy grants the user access to list the contents of all Amazon S3 buckets within the current account.
9. When an IAM user makes an API call, that user's long-term credentials are valid in which context?
- A. Only in the AWS Region in which their identity resides
  - B. Only in the Availability Zone in which their identity resides

- C. Only in the edge location in which their identity resides
  - D. Across multiple AWS Regions
10. When you use identity federation to assume a role, where are the credentials you use to make AWS API calls generated?
- A. Access key ID and secret access key are generated locally on the client.
  - B. The AWS Security Token Service (AWS STS) generates the access key ID, secret access key, and session token.
  - C. The AWS Key Management Service (AWS KMS) generates a customer master key (CMK).
  - D. Your Security Assertion Markup Language (SAML) identity provider generates the access key ID, secret access key, and session token.
11. You have an on-premises application that needs to sample data from all your Amazon DynamoDB tables. You have defined an IAM user for your application called `TableAuditor`. How can you give the `TableAuditor` user read access to new DynamoDB tables as soon they are created in your account?
- A. Define a custom IAM policy that lists each DynamoDB table. Revoke the access key, and issue a new access key for `TableAuditor` when tables are created.
  - B. Create an IAM user and attach one custom IAM policy per AWS Region that has DynamoDB tables.
  - C. Add the `TableAuditor` user to the IAM role `DynamoDBReadOnlyAccess`.
  - D. Attach the AWS managed IAM policy `AmazonDynamoDBReadOnlyAccess` to the `TableAuditor` user.
12. The principals who have access to assume an IAM role are defined in which document?
- A. IAM access policy
  - B. IAM trust policy
  - C. MS grant token
  - D. AWS credentials file
13. A new developer has joined your small team. You would like to help your team member set up a development computer for access to the team account quickly and securely. How do you proceed?
- A. Generate an access key based on your IAM user, and share it with your team member.
  - B. Create a new directory with AWS Directory Service, and assign permissions in the AWS Key Management Service (AWS KMS).
  - C. Create an IAM user, add it to an IAM group that has the appropriate permissions, and generate a long-term access key.
  - D. Create a new IAM role for this team member, assign permissions to the role, and generate a long-term access key.

- 14.** You have been working with the Amazon Polly service in your application by using the Python SDK for Linux. You are building a second application in C#, and you would like to run that application on a separate Windows Server with .NET. How can you proceed?

  - A. Migrate all your code for all applications to C#, and modify your account to a Windows account.
  - B. Go to the Amazon Polly service, and change the supported languages to include .NET.
  - C. Install the AWS SDK for .NET on your Windows Server, and leave your existing application unchanged.
  - D. Implement a proxy service that accepts your API requests, and translate them to Python.
- 15.** You are a Virginia-based company, and you have been asked to implement a custom application exclusively for customers in Australia. This application has no dependencies on any of your existing applications. What is a method you use to keep the customer latency to this new application low?

  - A. Set up an AWS Direct Connect (DX) between your on-premises environment and US East (N Virginia), and host the application from your own data center in Virginia.
  - B. Create all resources for this application in the Asia Pacific (Sydney) Region, and manage them from your current account.
  - C. Deploy the application to the US East (N Virginia) Region, and select Amazon EC2 instances with enhanced networking.
  - D. It does not matter which region you select, because all resources are automatically replicated globally.

# Chapter 2

AWS® Certified Developer Official Study Guide  
By Nick Alteen, Jennifer Fisher, Casey Gerena, Wes Gruver, Asim Jalil,  
Heiward Osman, Marife Pagan, Santosh Patlolla and Michael Roth  
Copyright © 2019 by Amazon Web Services, Inc.

## Introduction to Compute and Networking

---

**THE AWS CERTIFIED DEVELOPER –  
ASSOCIATE EXAM TOPICS COVERED IN  
THIS CHAPTER MAY INCLUDE, BUT ARE  
NOT LIMITED TO, THE FOLLOWING:**

**Domain 3: Development with AWS Services**

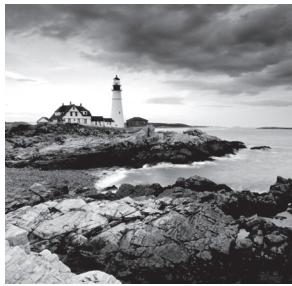
- ✓ 3.2 Translate functional requirements into application design.
- ✓ 3.4 Write code that interacts with AWS services by using APIs, SDKs, and AWS CLI.

**Domain 4: Refactoring**

- ✓ 4.2 Migrate existing application code to run on AWS.

**Domain 5: Monitoring and Troubleshooting**

- ✓ 5.2 Perform root cause analysis on faults found in testing or production.



Now that you have an AWS account and you can make application programming interface (API) calls from your local machine, it is time to explore how to run code on the AWS Cloud. AWS provides a broad set of compute options through the following services:

- Amazon Elastic Compute Cloud (Amazon EC2)
- Amazon Lightsail
- AWS Elastic Beanstalk
- Amazon Elastic Container Service (Amazon ECS)
- Amazon Elastic Container Service for Kubernetes (Amazon EKS)
- AWS Lambda

In this chapter, you will explore Amazon EC2, which provides you with environments called *instances*. You will learn about the components of an Amazon EC2 instance and explore an example of customizing an instance to run an application. Then, to learn how to customize the network environment for your instances, you will explore the network controls of Amazon Virtual Private Cloud (Amazon VPC). Finally, you will review some of the concerns related to managing your compute and networking environments.

Amazon EC2 and Amazon VPC are foundational services, and many of the concepts introduced in this chapter are transferrable to working with other AWS services.

## Amazon Elastic Compute Cloud

*Amazon Elastic Compute Cloud* (Amazon EC2) enables you to provision computing environments called *instances*. With Amazon EC2, you have the flexibility to choose the hardware resources you need. You are in control of the operating system and any other software that will run on the instance.

An Amazon EC2 instance runs on a host machine within a specific AWS Availability Zone. Typically, Amazon EC2 instances provide virtualized access to the underlying host machine resources. Using a combination of hardware and software components, instances present a virtualized interface to machine resources to the operating system. This virtualization enables multiple, different isolated guest environments to share the same underlying host machine. In addition to virtualized environments, some EC2 instance types offer *bare-metal access*. Bare-metal instances provide your applications with direct access to the processor and memory resources of the underlying server.

## Instance Types

With Amazon EC2, you choose your hardware resources from a broad set of preconfigured options by selecting a specific instance type and instance size. For example, your instance has a number of virtual CPUs (vCPUs) and a specific amount of RAM. The *instance type* is rated for a certain level of network throughput. Some instance types also include other hardware resources such as high-performance local disks, graphics cards, or even field-programmable gate arrays (FPGAs). The details of how the instance accesses the host resources, such as the specific hypervisor in use, also depend on the instance type that you select.

Even though AWS prescribes the hardware allocation for an instance type, a wide variety of instance types and sizes are available so that you can select the right level of resources for your application. For example, a t2.nano instance type allocates a fraction of a virtual CPU and 0.5 GiB of RAM to your instance. On the other end of the size spectrum, an x1e.32xlarge instance type provides 128 virtual CPUs and 3,904 GiB of RAM.

Instance types are also grouped into *instance families* to help you choose the appropriate instance for your application. Instances within a given family share similar characteristics, such as the ratio of vCPU to RAM or access to different types of storage options.

For an overview of the different instance families and their use cases, see Table 2.1.

**TABLE 2.1** Amazon EC2 Instance Families

| Amazon EC2 Instance Family | For Applications That Require...                                                                                                                                                   |
|----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| General purpose            | A balanced mix of CPU, RAM, and other resources                                                                                                                                    |
| Compute optimized          | A high amount of CPU, such as high-performance web servers, scientific modeling, and video encoding                                                                                |
| Memory optimized           | A large amount of RAM, such as in-memory databases and distributed web scale in-memory caches                                                                                      |
| Storage optimized          | A large amount of storage and input/output (I/O) throughput, such as data warehousing, analytics, and big data distributed computing                                               |
| Accelerated computing      | Dedicated Graphics Processing Unit (GPU) or Field Programmable Gate Array (FPGA) resources, such as 3D rendering, deep learning, genomics research, and real-time video processing |

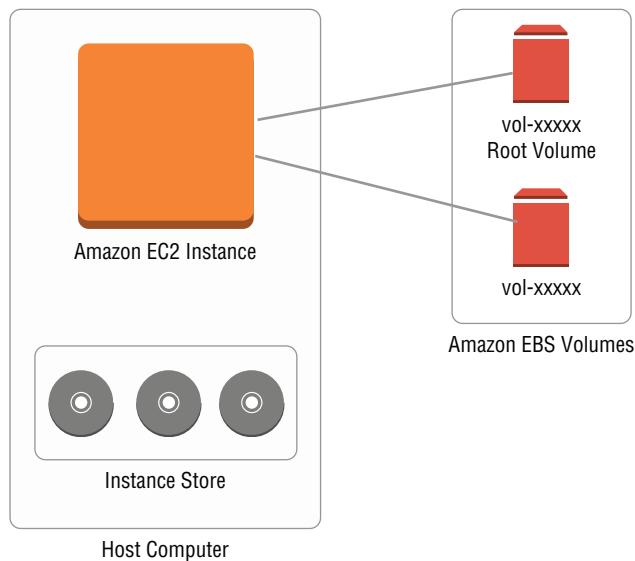
When you select an instance, choose a size that is appropriate for your current workload because Amazon EC2 instances are resizable. To change the hardware allocation, stop the instance, modify the instance type attribute, and then start the instance again.

## Storage

Your instance requires storage volumes for both the root volume and any additional storage volumes that you want to configure. You can create persistent storage volumes with the *Amazon Elastic Block Store (Amazon EBS)* service to provide block storage devices for Amazon EC2 instances. Certain instance types enable you to mount volumes based on an *instance store*, which is temporary storage local to the host machine.

For an overview of the relationship between Amazon EBS volumes, instance store volumes, and the Amazon EC2 instance, see Figure 2.1.

**FIGURE 2.1** Amazon EC2 storage



## Persistent Storage

For Amazon EC2 instances, Amazon EBS provides persistent block storage. Similar to a hard drive, block storage volumes provide read/write access at a block level and can be formatted with a file system. Also similar to a hard drive, you can attach each EBS volume to a single instance at a time. Amazon EBS is suitable for installing operating systems and applications and for data that you want to store persistently. You can also encrypt the volumes.

When you create an EBS volume, you provision a specific size for the storage volume. You choose from several types of volumes with different underlying storage technologies and performance options. You can increase the size of the volume later, even while it is being used by a running instance.

While an EBS volume is attached to a particular instance, only that instance can access the data on that volume. However, you can detach an EBS volume from one instance and then attach that volume to another instance in the same Availability Zone.

EBS volumes are decoupled from the underlying physical host running the instance. The decoupling of the storage volume from the host machine enables you to persist data even if your instance is no longer running on the physical host. Although the EC2 instance treats the EBS volume as a local disk, the underlying host machine reads and writes to the EBS volume over the network. To maintain peak performance for this connection, you can use EBS-optimized instance types. EBS-optimized instances reserve dedicated network bandwidth specifically for traffic to the EBS volume.

EBS volumes automatically replicate the data for a particular volume within the same Availability Zone as your Amazon EC2 instance. To increase durability of your data, you can use Amazon EBS to make point-in-time snapshots of an EBS volume. Data for Amazon EBS snapshots is automatically replicated across multiple Availability Zones within a region, and these snapshots can be used to create new volumes. If there's an accidental delete or other application error, snapshots enable you to recover your data.

## Temporary Storage

Certain Amazon EC2 instance types also allow you to mount *instance store* volumes—storage local to the physical host that runs your Amazon EC2 instance. An instance store volume is a good fit for high-performance storage of caches or temporary files and for use cases in which your application is already replicating the data to other locations.

This storage can have a high read/write performance because it is physically attached to the host machine that runs the instance. However, because this storage is local to the host machine, your data persists only while the instance is running on that host machine. The data persists if the instance reboots; however, AWS deletes the data on the instance store whenever you stop or terminate the instance.

## Software Images

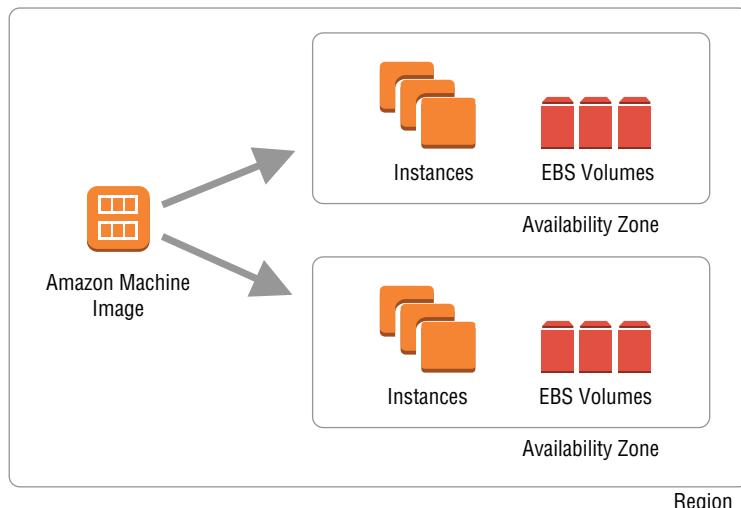
When the server first boots, it requires an operating system (OS) and the configuration of the attached storage volumes. An *Amazon Machine Image (AMI)* provides the template for the OS and applications on the root volume of your instance. AMIs also provide a block device mapping that can specify additional volumes to mount when an instance launches, as shown in Figure 2.2.

AWS provides a variety of AMIs. Paid AMIs are available through the AWS Marketplace.

You can create your own AMIs from an Amazon EC2 instance that you have previously customized or by importing your own virtual machine (VM) images. *Each AWS Region maintains its own listing of AMIs. Any AMIs that you create are available only within a specific region unless you copy them to other regions.* You can share AMIs between AWS accounts. To control which AWS accounts can use your AMIs, define the launch permissions for your AMI.

Depending on the source of the AMI and the type of software license required, the cost of the software licensing may be included in the hourly rate of the instance (such as Windows Server). For instances from the AWS Marketplace, charges are incurred for software licensing in addition to the Amazon EC2 infrastructure.

**FIGURE 2.2** Amazon Machine Images



## Network Interfaces

Virtual network interfaces called *elastic network interfaces* provide networking for your Amazon EC2 instances. Elastic network interfaces are associated with a software-defined network provided by Amazon VPC. Each Amazon EC2 instance is assigned a *primary network interface* that is associated with a subnet within an Amazon VPC. By default, if you omit the network configuration, Amazon EC2 assigns the instance to one of the subnets within the *default VPC*. The instance receives both a *private IP* address to communicate with instances inside the Amazon VPC and a *public IP* address to communicate with the internet. A *security group* protects the traffic entering and exiting the network interface. Security groups act as a stateful firewall. To make network connections to your instance, you must set security group rules to allow the connection.

You can attach additional network interfaces to an EC2 instance. Each network interface has its own MAC addresses and IP address associations. Unlike the primary network interface, you can detach secondary network interfaces from one Amazon EC2 instance and then attach it to another instance.

The number of network interfaces that you can attach to an instance and the network throughput depends on the specific instance type and size that you select. The number of network interfaces that you attach does not affect the network throughput of the instance; the bandwidth available to the instance depends on the instance type and size, not the number of network interfaces.

## Accessing Instances

By default, Linux Amazon EC2 instances provide remote access through SSH, and Windows Amazon EC2 instances provide remote access through the Remote Desktop Protocol (RDP). To connect to these services, you must have the appropriate inbound rules on the security group for the instance.

Depending on the operating system and AMI that you use to launch the instance, a default administrator is provided for your initial sign-in. To acquire the credentials needed to sign in as the default user, you must specify an Amazon EC2 key pair when you launch the instance. After you sign in, you can create additional users with the appropriate Linux or Windows tools.

### Default User

The default user for Amazon Linux instances is `ec2-user`. For other Linux operating systems, this default user may vary depending on the AMI provider. For example, the default user for Ubuntu Linux is `ubuntu`.

For Windows instances, the default user is `Administrator`. This account may have a different name depending on the language of the server. For example, if the server is configured with French as the language, the administrator account is localized to `Administrateur`.

### Amazon EC2 Key Pairs

An *Amazon EC2 key pair* has a name, and it is composed of a public key and a private key. AWS retains the public key, and it is your responsibility to store the private key securely. If you specify an Amazon EC2 key pair when you launch the instance, it secures the sign-in credentials as part of the Amazon EC2 instance provisioning process. For a Linux instance, the public key from the key pair is added to the `~/.ssh/authorized_keys` file for the default user. For a Windows instance, the password for the default administrator account is encrypted with the public key and can be decrypted with the private key.

When you create a new key pair, you can import a key pair that you generated locally, or you can have AWS generate a key pair for you. If you request that AWS generate the key pair, you can download the private key only at the time the key pair is generated. You are responsible for storing the private key file securely. You will not be able to download it again after it is created.

If you do not specify a key pair when you launch the instance, you are unable to sign in to that instance. Amazon EC2 key pairs are regional in scope, so you need key pairs in each region where you launch EC2 instances.

## Instance Lifecycle

An Amazon EC2 instance has three primary states: running, stopped, and terminated. Additionally, there are intermediate states of pending, stopping, and shutting down. An Amazon EC2 instance accrues charges for the compute resources only when it is in the

running state. However, EBS volumes persist data even when an instance is stopped, so the charges for persistent storage from any EBS volumes accrue independently from the state of the instance.

When you first launch an instance from an AMI, it goes into the pending state until it enters the running state on a host machine.

After an instance is running, for instances with EBS-backed storage, you can stop the instance. If you stop the instance, it enters the stopping state. Any data on instance store drives on that host are erased.

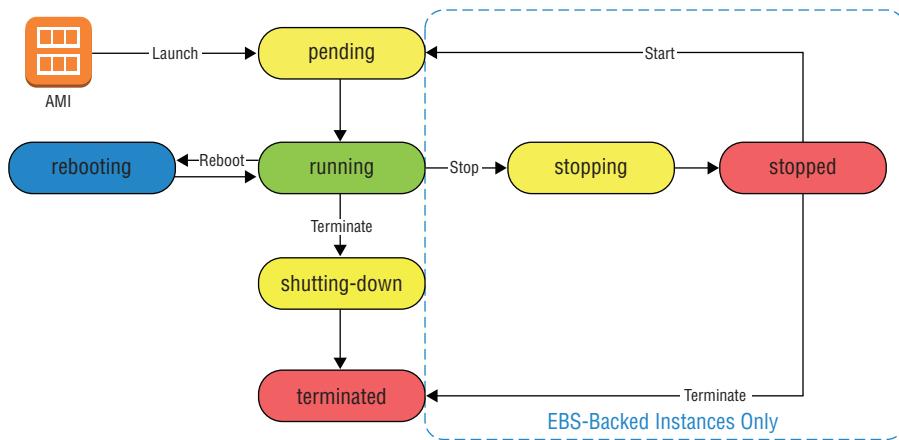
When an instance is stopped, you can modify attributes that cannot be changed, such as instance type, while the instance is running. You can also start stopped instances. When you start an instance, it enters the pending state until it is running again.

Typically, each time an instance is started, it is launched on a different physical host machine than before. If the underlying physical host is impaired and requires maintenance, stopping and then starting the Amazon EC2 instance moves the instance to a healthy host.

You can also terminate an instance. It first goes through shutting down; then eventually it is terminated. The default behavior is to delete the EBS volumes associated with the instance on termination.

To view the lifecycle of an Amazon EC2 instance, see Figure 2.3.

**FIGURE 2.3** Amazon EC2 instance lifecycle



## Running Applications on Instances

This section reviews how to connect to an EC2 instance and explores some features that are useful when you run applications or custom code on an instance. These features include ways of customizing the software on an instance, how your code can discover properties about the instance, and how to provide API credentials to your code running on an

instance. An example that ties these features together is provided. Finally, the section describes how you can monitor the status of the instance.

## Connecting to Amazon EC2 Instances

With EC2 instances, you have full administrative control to install software packages on your instance and create additional user accounts as needed. By default, to connect to a Linux instance, you can directly use the private key from the Amazon EC2 key pair with an SSH client, as shown in Figure 2.4.

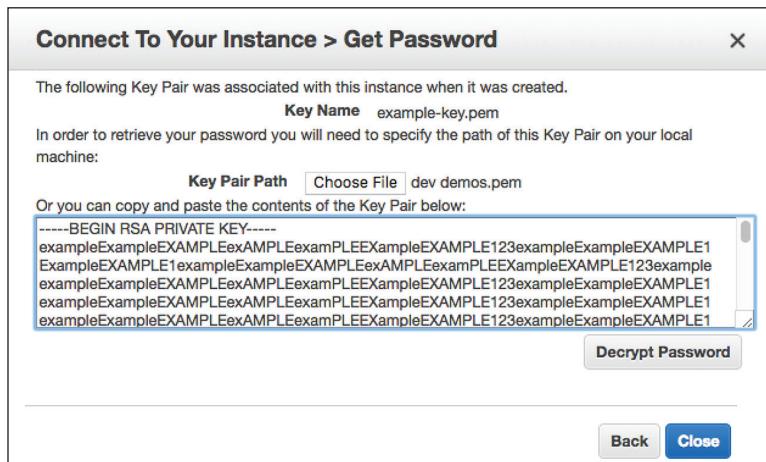
**FIGURE 2.4** Using SSH with an Amazon EC2 instance

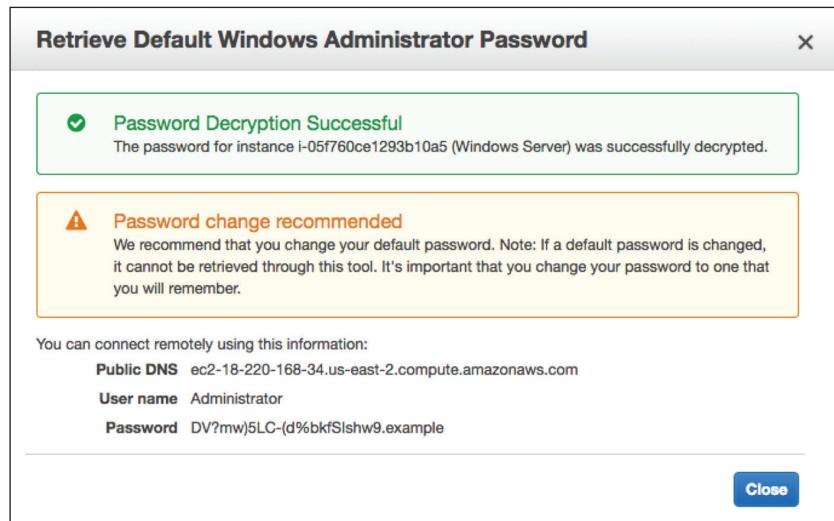
```
1. ec2-user@ip-172-31-54-133:~ (ssh)
Last login: Tue Aug 14 22:38:47 on ttys001
9801a79dfc6d:~ heiwad$ ssh -i "example-key.pem" ec2-user@ec2-35-173-135-74.compute-1.amazonaws.com
The authenticity of host 'ec2-35-173-135-74.compute-1.amazonaws.com (35.173.135.74)' can't be established.
ECDSA key fingerprint is SHA256:58aIPoSOhic9ZZhrN/01lthKxfhRgXIlgNgCDIqhMM.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'ec2-35-173-135-74.compute-1.amazonaws.com,35.173.135.74' (ECDSA) to the list of known hosts.

-| _-|_
-| (/ Amazon Linux 2 AMI
---___|_|
https://aws.amazon.com/amazon-linux-2/
[ec2-user@ip-172-31-54-133 ~]$
```

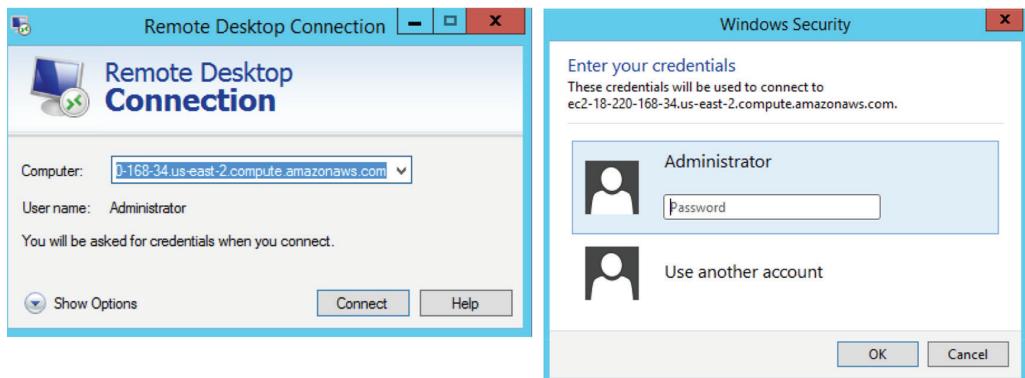
For a Windows instance, the password for the Administrator account is encrypted with the public key. You can decrypt the password by using the associated private key, as illustrated in Figure 2.5 and Figure 2.6.

**FIGURE 2.5** Decrypting a Windows password



**FIGURE 2.6** Viewing a Windows password

After you have decrypted the password, you can use Microsoft Remote Desktop to connect to the instance, as shown in Figure 2.7.

**FIGURE 2.7** Connecting to a Windows instance

## Customizing Software with User Data

You can connect to your instance and install any applications you want from an interactive session. However, one of the advantages of moving to the cloud is to automate previously manual steps. Instead of logging in to the instance, another way to customize the software

on your instance is to provide *user data* as part of the request to launch the instance. For Linux instances, user data can be a shell script or a *cloud-init* directive. On Windows instances, depending on the version of Windows Server, either EC2Config or EC2Launch processes the user data. By default, commands supplied to user data execute only at first boot of the instance.

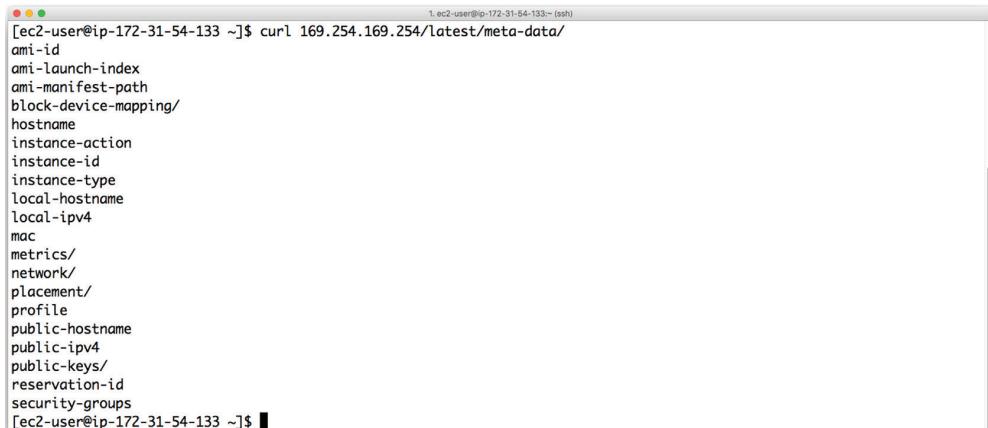
Here is an example of installing an Apache web server on an Amazon Linux 2 instance with a shell script that is provided as the user data:

```
#!/bin/bash
yum update -y
yum install httpd -y
systemctl start httpd
systemctl enable httpd
```

## Discovering Instance Metadata

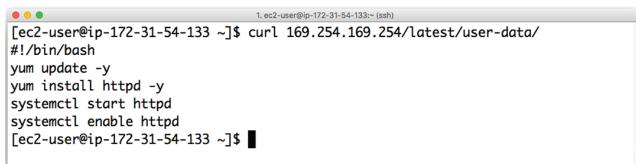
With the *instance metadata service* (IMDS), code running on an Amazon EC2 instance can discover properties about that instance. The instance metadata service exposes a special IP address, 169.254.169.254, which you can query using HTTP to perform lookups. You can query a broad range of metadata attributes, as shown in Figure 2.8. These attributes can include the instance ID and credentials derived from an Identity and Access Management (IAM) role.

**FIGURE 2.8** Amazon EC2 metadata attributes



```
1. ec2-user@ip-172-31-54-133 ~]$ curl 169.254.169.254/latest/meta-data/
ami-id
ami-launch-index
ami-manifest-path
block-device-mapping/
hostname
instance-action
instance-id
instance-type
local-hostname
local-ipv4
mac
metrics/
network/
placement/
profile
public-hostname
public-ipv4
public-keys/
reservation-id
security-groups
[ec2-user@ip-172-31-54-133 ~]$
```

With IMDS, it is also possible to retrieve the user data that was used to bootstrap an instance, as shown in Figure 2.9.

**FIGURE 2.9** Querying Amazon EC2 user data

```
[ec2-user@ip-172-31-54-133 ~]$ curl 169.254.169.254/latest/user-data/
#!/bin/bash
yum update -y
yum install httpd
systemctl start httpd
systemctl enable httpd
[ec2-user@ip-172-31-54-133 ~]$
```



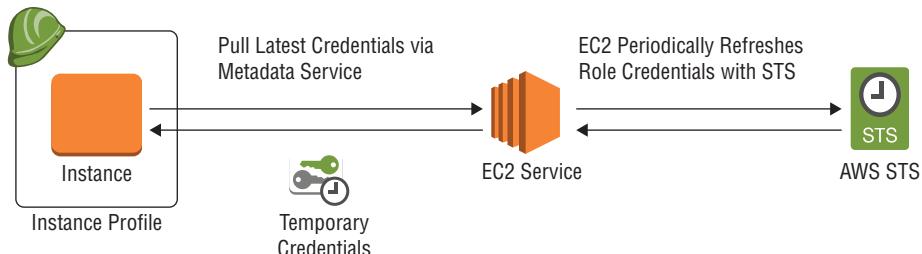
Anyone who can access an instance can view its metadata and user data. Do not store sensitive data, such as passwords or access keys, as user data.

## Assigning AWS API Credentials

You can assign an IAM role to an Amazon EC2 instance. The AWS Software Development Kit (SDK) and AWS Command Line Interface (AWS CLI) can automatically discover these credentials through the Amazon EC2 metadata service. You can skip the task of explicitly configuring credentials files on your instances during bootstrapping.

When you assign an IAM role to an instance, it is assigned indirectly, through an *instance profile*, which is a container for an IAM role. You can associate a particular instance profile with many Amazon EC2 instances. However, a particular Amazon EC2 instance can be associated with one instance profile at a time, and an instance profile can be associated with only one IAM role. You can associate or disassociate Amazon EC2 instances with an instance profile at launch or even when the instances are running.

When an instance profile with an IAM role is associated with an instance, the Amazon EC2 service makes the necessary calls to the *AWS Security Token Service (AWS STS)* automatically to generate short-term credentials for that instance. These credentials are based on the IAM role associated with the instance profile. The credentials are exposed to the instance through the Amazon EC2 metadata service, as shown in Figure 2.10.

**FIGURE 2.10** Instance profile and IAM role credentials

## Serving a Custom Webpage

This example combines Amazon EC2 user data, the Amazon EC2 metadata service, and IAM roles to configure an Amazon EC2 instance. In the example, a web server displays a static webpage that shows custom information.

The following is a bootstrapping script that enables you to configure an Amazon EC2 instance running Amazon Linux 2. At first boot of the instance, the script generates a static page that displays the instance ID, instance type, Availability Zone, and public IP address of the instance at the time the script was executed.

The first line of the script declares the type of script. Then the script installs the Apache web server and configures it to run as a service. Because the script is running as the root user, there is no requirement to preface the commands with sudo.

Next, the script makes several calls to the Amazon EC2 metadata service and saves the results into environment variables to be used later in the script.

To generate an MP3 file that speaks out the instance ID, the script makes an API call to Amazon Polly. For this API call to succeed, you must assign the instance an IAM role that allows permissions to the Amazon Polly `SynthesizeSpeech` action. The managed `AmazonPollyReadOnlyAccess` policy can grant these permissions.

Next, the script generates a static HTML page. This page references the values that were previously stored as environment variables. After this script completes, your Amazon EC2 instance can respond to HTTP requests and show the customized page. To see this page, you must verify that the Amazon EC2 instance has port 80 open in its security group and is assigned a public IP address.

```
#!/bin/bash

Install Apache Web Server
yum update -y
yum install httpd -y
systemctl start httpd
systemctl enable httpd

Discover configuration using the EC2 metadata service
ID=$(curl 169.254.169.254/latest/meta-data/instance-id)
TYPE=$(curl 169.254.169.254/latest/meta-data/instance-type)
AZ=$(curl 169.254.169.254/latest/meta-data/placement/availability-zone)
IPV4=$(curl -f 169.254.169.254/latest/meta-data/public-ipv4)

Set up the Web Site
cd /var/www/html

Make AWS Cloud API calls to generate an audio file
VOICE=$(aws polly describe-voices --language-code en-US \
--region us-west-2 --query Voices[0].Id --output text)
aws polly synthesize-speech --region us-west-2 --voice-id $VOICE \
--text "Hello from EC2 instance $ID." --output-format mp3 instance.mp3
```

```
Generate customized index.html for this instance
echo "<html><body><H1>Welcome to your EC2 Instance</H1><p><p>" > ./index.html
echo "<audio controls>" >> ./index.html
echo '<source src="instance.mp3" type="audio/mp3">' >> ./index.html
echo 'Here is an audio greeting. ' >> ./index.html
echo "</audio><p><p>" >> ./index.html
echo "There are many other instances, but" >> ./index.html
echo "$ID is yours.<p><p>" >> ./index.html
echo "This is a $TYPE instance" >> ./index.html
echo " in $AZ. <p><p>" >> ./index.html
if ["$IPV4"];
then
 echo "The public IP is $IPV4.<p><p>" >> ./index.html
else
 echo "This instance does NOT have" >> ./index.html
 echo "a public IP address.<p><p>" >> ./index.html
fi
echo "--Audio provided by the $VOICE voice.<p><p>" >> ./index.html
echo "</body></html>" >> ./index.html
```

## Monitoring Instances

Now that you have an application running on your instance, you may be interested in understanding how that application performs, or if it is still running at all. Amazon EC2 performs automated status checks of the software and hardware of the underlying host machine every minute. These status checks are to verify that the instance can connect to the network and can run successfully on the host machine. If the status checks find no underlying issues, they return the status OK. If an issue is detected that prevents normal operation of the Amazon EC2 instance, the status checks return the status as impaired. The results of these status checks are available in Amazon CloudWatch.

For each of your instances, the Amazon EC2 service automatically collects metrics related to CPU utilization, disk reads and writes, and network utilization and makes them available in *CloudWatch*. You can supplement these built-in metrics with data from the guest operating system on your instance, such as memory utilization and logs from your application, by installing and configuring the CloudWatch agent on the instance.

Using CloudWatch, you can automate actions based on a metric through CloudWatch alarms. For example, you can configure an CloudWatch alarm that applies the recover instance action if status checks show that the host running the instance is impaired.

# Customizing the Network

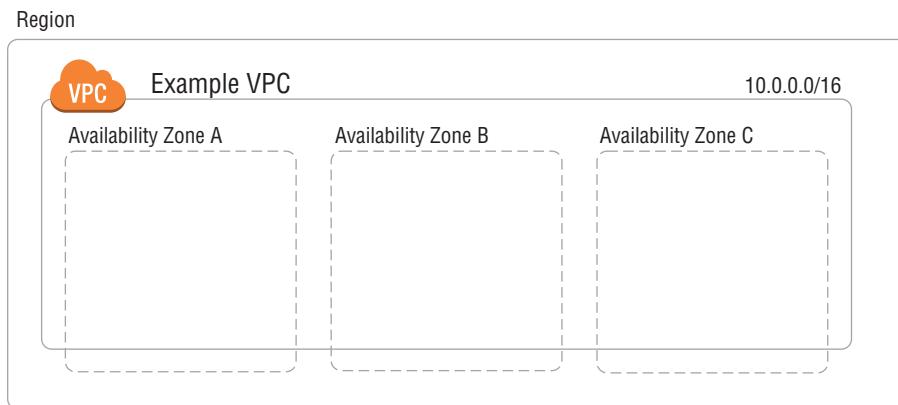
While the default Amazon VPC service provides a quick way to start with Amazon EC2 instances, it is important to understand how multiple Amazon EC2 instances communicate within an Amazon VPC network. The AWS Certified Developer – Associate exam may test your knowledge of Amazon VPC by asking troubleshooting questions related to the network. In this section, you'll explore the Amazon VPC that enables you to create software-defined networks within an AWS Region.

## Amazon Virtual Private Cloud

*Amazon Virtual Private Cloud* (Amazon VPC) provides logically isolated networks within your AWS account. These networks are software defined and can span all of the Availability Zones within a specific AWS Region. For each VPC, you have full control over whether the Amazon VPC is connected to the internet, to a private on-premises network, or to other Amazon VPCs. Until you explicitly create these connections, instances in your VPC are able to communicate with other instances in the same VPC only.

You define an Amazon VPC with one or more blocks of addresses specified in the Classless Inter-Domain Routing (CIDR) notation. If, for example, you specified 10.0.0.0/16 as the block for a VPC, this means that the VPC includes IP addresses in the range from 10.0.0.0 through 10.0.255.255. For an example of an Amazon VPC spanning multiple Availability Zones in a region, see Figure 2.11.

**FIGURE 2.11** Amazon VPC overview



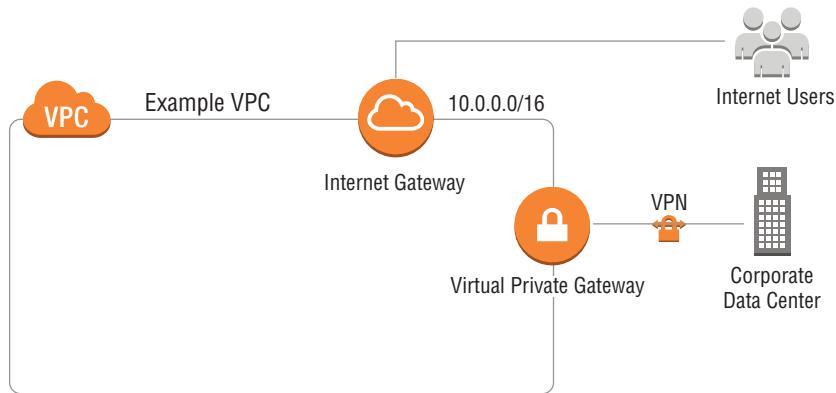
## Connecting to Other Networks

By default, an Amazon VPC is an isolated network. Instances within an Amazon VPC cannot communicate with the internet or other networks until you explicitly create connections. Table 2.2 provides an overview of various types of connections that you can establish between an Amazon VPC and other networks.

**TABLE 2.2** Amazon VPC Connection Types

| Connection Type              | Description                                                                                                                             |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| Internet Gateway             | A highly available connection that allows outbound and inbound requests to the internet from your Amazon VPC                            |
| Egress Only Internet Gateway | A special type of internet gateway for IPv6 that allows outbound traffic and corresponding responses but blocks inbound connections     |
| Virtual Private Gateway      | Allows you to establish a private connection to your corporate network by using a VPN connection or through Direct Connect (DX)         |
| Amazon VPC Endpoints         | Allows traffic from your Amazon VPC to go to specific AWS services or third-party SaaS services without traversing an internet gateway  |
| Amazon VPC Peering           | Privately routes traffic from one Amazon VPC to another Amazon VPC by establishing a peer relationship between this VPC and another VPC |
| AWS Transit Gateway          | Allows you to centrally manage connectivity between many VPCs and an on-premises environment using a single gateway                     |

For an example of an Amazon VPC with a connection to an internet gateway and a VPN connection to an on-premises network provided by a virtual private gateway, see Figure 2.12.

**FIGURE 2.12** Amazon VPC with gateway connections

## IP Addresses

When working with Amazon VPC, all instances placed within a particular VPC are assigned one or more IP addresses. There are four different types of IP addresses available for use with Amazon VPC. Primarily, these IP addresses are based on IPv4; however, you can enable support for IPv6.

## Private IP Addresses

*Private IP addresses* are IPv4 addresses that are not reachable from the internet. These addresses are unique within a VPC and used for traffic that is to be routed internally within the VPC, for private communication with corporate networks, or for private communication with other VPCs.

When you create a VPC, you assign one or more blocks of addresses to the VPC, and typically these blocks will be within the range of IPv4 addresses reserved for private networks as specified in RFC1918. When an instance is launched, it is launched into a subnet within the VPC, and the instance is assigned a private IP address automatically from the block of addresses assigned to that particular subnet. When an instance is assigned a private IPv4 address, this association persists for the lifecycle of the instance—even when the instance is stopped.

## Public IP Addresses

Whether an EC2 instance is assigned public IP addresses automatically, in addition to the private IP address, depends on the following factors:

- Configuration passed when launching the instance
- Options for the subnet in which that instance is launched

Unlike the private IP address, the public IP address is an IPv4 address that is reachable from the internet.

AWS manages the association between an instance and a public IPv4 address, and the association persists only while the instance is running. You cannot manually associate or disassociate public IP addresses from an instance.

## Elastic IP Addresses

An *Elastic IP address* is similar to a public IP address in that it is an IPv4 address that is reachable from the internet. However, unlike public IP addresses, you manage the association between instances and Elastic IP addresses. You control when these addresses are allocated, and you can associate, disassociate, or move these addresses between instances as needed.

You may also assign Elastic IP addresses to infrastructure such as NAT gateways. These addresses can come from a pool of IP addresses that AWS manages or from blocks of IPv4 addresses you have brought to your AWS account.

## IPv6 Addresses

In addition to IPv4 addresses, you can associate an Amazon-provided block of IPv6 addresses to your VPC. When you enable IPv6 in your VPC, the network operates in *dual-stack mode*, meaning that IPv4 and IPv6 commutations are independent of each other. Your resources can communicate over IPv4, IPv6, or both.

## Subnets

Within an Amazon VPC, you define one or more subnets. A subnet is associated with a specific Availability Zone within the region containing the Amazon VPC. Each subnet has its own block of private IP addresses defined using CIDR notation. This block is a subset of the overall IP address range assigned to the Amazon VPC and does not overlap with any other subnet in the same Amazon VPC.

For example, a subnet may be assigned the CIDR block range 10.0.0.0/24, which would include addresses in the range 10.0.0.0–10.0.0.255. Out of the 256 possible addresses, Amazon VPC reserves the first four IP addresses and the last IP address in the range, leaving 251 IP addresses in the subnet.

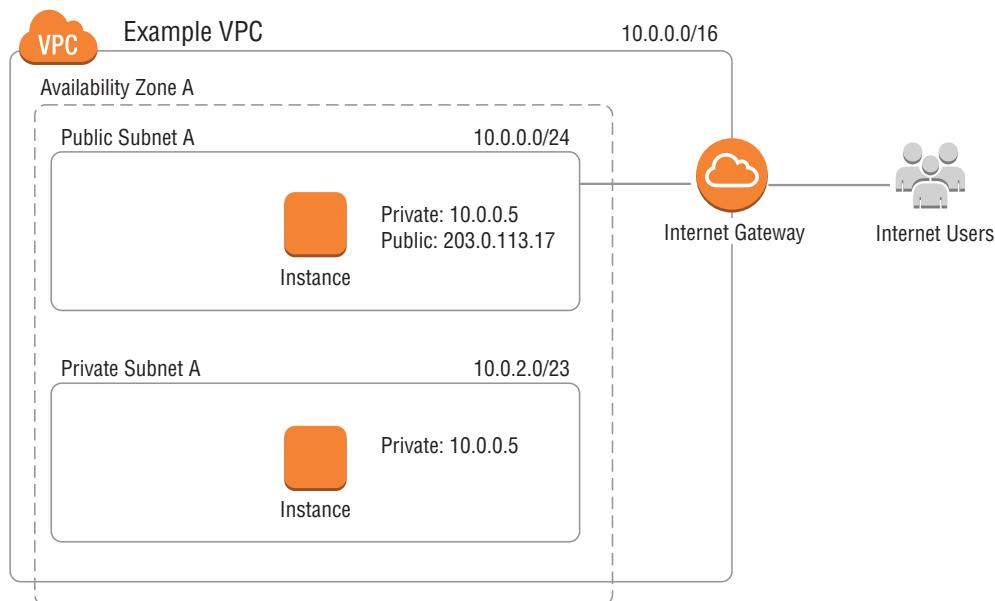
When you launch an Amazon EC2 instance into a subnet, its primary network interface assigns a private IPv4 address automatically from the CIDR range assigned to the subnet.

Typically, you create at least two types of configurations for subnets in a VPC. The first is for subnets in which you place instances that you want to reach directly from the internet. This could be an instance running as a web server, for example. Subnets of this type are known as *public subnets*.

The second type of configuration is usually a subnet that backend instances use that must be accessible to your other instances but should not be directly accessible from the internet. Subnets of this type are known as *private subnets*. For example, if you had an instance that was dedicated to running a database, such as MySQL, you could place that instance in a private subnet. It would be accessible from the web server in the public subnet, but it would not accept traffic from the internet.

For an example of an Amazon VPC with a public and a private subnet, see Figure 2.13.

**FIGURE 2.13** Amazon VPC with public and private subnets



In addition to Amazon EC2 instances, many AWS managed services, such as Amazon Relational Database Service (Amazon RDS) or Amazon ElastiCache, also enable you to expose your resources in specific subnets and, in particular, into private subnets. You can create these resources and access them privately from instances within your Amazon VPC.

## Route Tables

Network traffic exiting a subnet is controlled with routes that are defined in a route table. Routes define how the implicit router in the Amazon VPC routes IP traffic from a subnet to destinations outside that subnet. Each route table includes a rule called the *local route*. This rule or *route* is what allows traffic from instances in one subnet within the Amazon VPC to send traffic to instances in any other subnets within the same Amazon VPC. A route is composed of two parts: a destination and a target for the network traffic.

Unless explicitly associated with a specific route table, subnets associate with a default route table called the *main route table*. By default, the main route table includes only the local route. This means that subnets that are associated with the default route table have no connection to the internet. They can route traffic privately only within the Amazon VPC. However, you can modify this table or define additional route tables and rules as required.

For an example of the main route table for an Amazon VPC, see Table 2.3.

**TABLE 2.3** Main Route Table Example

| Destination | Target |
|-------------|--------|
| 10.0.0.0/16 | local  |

Route tables and the configured rules differentiate public subnets from private subnets. For example, you might create a public subnet by associating the subnet with a route table that includes a rule to route internet-bound traffic through an internet gateway. To represent any IP address on the internet in the rule, you can use the 0.0.0.0/0 CIDR block. Table 2.4 is an example of a route table that contains the defined rules.

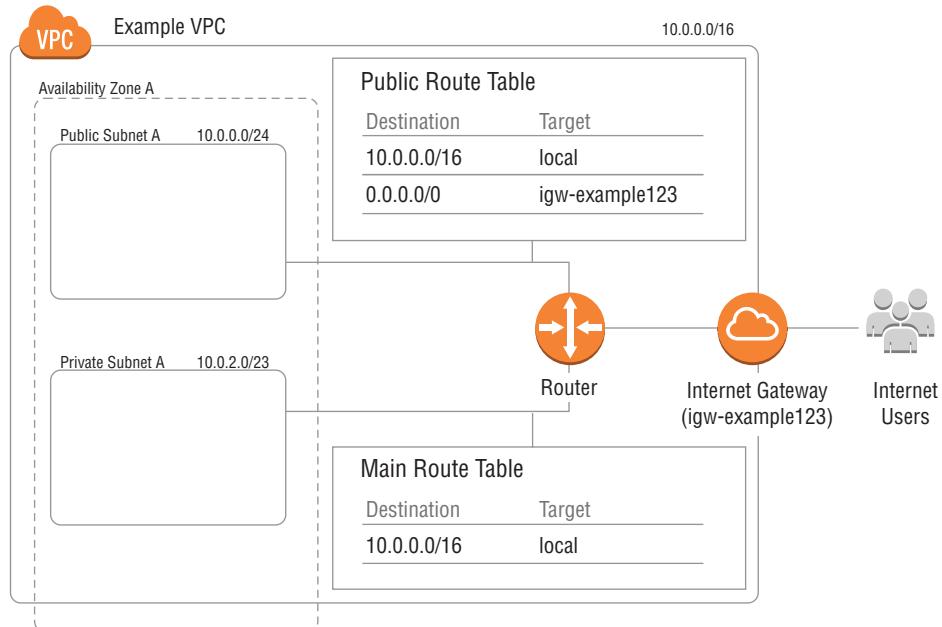
**TABLE 2.4** Public Route Table Example

| Destination | Target         |
|-------------|----------------|
| 10.0.0.0/16 | local          |
| 0.0.0.0/0   | igw-example123 |

When you launch an Amazon EC2 instance into a public subnet, assign a public IP address to the instance. Even though the subnet has a route to an internet gateway, the instance is not able to communicate with the internet without a public IP address. Route table rules are evaluated in order of specificity.

To review a diagram of an Amazon VPC that has a public and a private subnet configured with the route table rules, see Figure 2.14.

**FIGURE 2.14** Amazon VPC with public and private subnets with rules



## Security Groups

*Security groups* act as a stateful firewall for your Amazon EC2 instances. When you define security group rules, you specify the source or destination of the network traffic in addition to the protocols and ports that you allow. If you change the security group rules, that change propagates to any instances associated with that security group.

By using inbound security group rules, you can control the source, protocols, and ports of allowed network traffic. For example, you could allow TCP connections that originate from the IPv4 address of your home network so that you can administer an Amazon EC2 instance using SSH.

Outbound rules enable you to control destination, protocols, and ports of allowed network traffic. Security groups include a default outbound rule that allows all outbound requests on all protocols and ports to all destinations. To control outbound requests more tightly, you can remove this default rule and add specific outbound rules in its place.

When you specify a source or destination for a security group rule, you can use IPv4 or IPv6 address ranges. Alternatively, you can use an identifier for a security group as a source or destination.

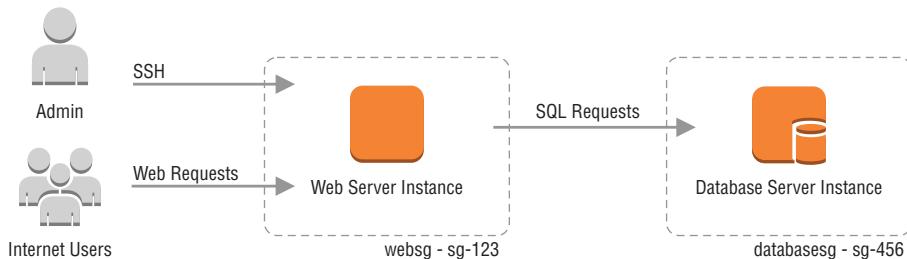
Assume that you have two EC2 instances—one instance is running a web server, and a second instance is running a database application. To allow network connections to these instances, you create two security groups: a security group for your web server instances called `websg` and a second security group for your database instances called `databasessg`.

For `websg`, you set inbound rules that allow web requests from anywhere. You also allow inbound SSH but only from a specific IP address that your administrator uses. You have not yet modified the default outbound rule for `websg`, so all outgoing connections are allowed.

For `databasessg`, you write an inbound rule that allows incoming traffic on TCP port 3306 originating from `websg`. Remove the default outbound rule and instead add rules to allow outbound connections to download software updates over HTTP and HTTPS. All other outbound connections from `databasessg` will be blocked.

To view the diagram of the security groups and rules for this scenario, see Figure 2.15. Also, see Table 2.5, Table 2.6, Table 2.7, and Table 2.8 for the corresponding inbound and outbound rules for these security groups.

**FIGURE 2.15** Security groups



**TABLE 2.5** Inbound Rules for `websg`

| Protocol | Port | Source       | Comments                                                  |
|----------|------|--------------|-----------------------------------------------------------|
| TCP      | 80   | 0.0.0.0/0    | Allow incoming HTTP requests from internet users          |
| TCP      | 443  | 0.0.0.0/0    | Allow incoming HTTPS requests from internet users         |
| TCP      | 22   | 10.10.0.6/32 | Allow incoming SSH only from the administrator's computer |

**TABLE 2.6** Outbound Rule for websg

| Protocol | Port | Destination | Comments                        |
|----------|------|-------------|---------------------------------|
| All      | All  | 0.0.0.0/0   | Allow all outbound IPv4 traffic |

**TABLE 2.7** Inbound Rule for databasessg

| Protocol | Port | Source | Comments                             |
|----------|------|--------|--------------------------------------|
| TCP      | 3306 | sg-123 | Allow inbound SQL queries from websg |

**TABLE 2.8** Outbound Rules for databasessg

| Protocol | Port | Destination | Comments                                  |
|----------|------|-------------|-------------------------------------------|
| TCP      | 80   | 0.0.0.0/0   | Allow all outbound HTTP for updates       |
| TCP      | 443  | 0.0.0.0/0   | Allow outbound HTTPS requests for updates |

If you specify an inbound rule, replies to that incoming connection are permitted. Similarly, if you specify an outbound rule, the replies to the outbound request are permitted. In the previous example, when a web server instance makes a connection to a database, only the outbound rule for the web server and the inbound rule for the database must allow the flow. The inbound rule for the web server and the outbound rule for the database will not be evaluated for this flow.

*Security groups only support rules to allow traffic.* Therefore, if you assign multiple security groups to your instance, the security group rules combine in the most permissive way; each group contributes to opening up more access to the instance.

If you fail to specify a security group when you launch the Amazon EC2 instance, the instance associates with the default security group for the Amazon VPC. If your Amazon EC2 instance has more than one network interface, you can manage the security groups for each network interface independently from the others.

## Network Access Control Lists

In addition to routes, *network access control lists* (network ACLs) allow an administrator to control traffic that enters and leaves a subnet. A network ACL consists of inbound and outbound rules that you can associate with multiple subnets within a specific Amazon VPC. Network ACLs act as a stateless firewall for traffic to or from a specific subnet.

Whereas security group rules provide only the capability to allow traffic, network ACL rules support the ability to allow specific types of traffic and to deny specific traffic.

However, unlike security groups, network ACLs are stateless and do not track connections and their replies. This means that to allow for a particular traffic flow, both inbound and outbound rules must allow it for that network ACL. For inbound rules, you can specify the protocol, port range, and source IP address range. For outbound rules, you specify the protocol, port range, and destination IP address range. For each rule, you also choose whether the rule allows or denies traffic. Rules in a network ACL are numbered and evaluated in order from the smallest to largest rule number.

If you do not specify a network ACL, the subnet is associated with the default network ACL for the Amazon VPC. This network ACL comes with rules that allow all inbound and outbound traffic. Table 2.9 shows an example of the inbound rules for a default network ACL. The final rule for the network ACL, rule 100, is a universal rule that explicitly denies traffic that does not match any other rule. Because there is a rule to allow all traffic and rules are evaluated in order, this universal rule has no effect. However, if you remove or modify rule 100, then the final rule would apply to any traffic that did not match any of the other rules.

**TABLE 2.9** Default Network ACL Inbound Rules

| Rule Number | Type        | Protocol | Port Range | Source    | Allow/Deny |
|-------------|-------------|----------|------------|-----------|------------|
| 100         | All traffic | All      | All        | 0.0.0.0/0 | Allow      |
| *           | All traffic | All      | All        | 0.0.0.0/0 | Deny       |

Table 2.10 shows an example of the outbound rules for the default network ACL for an Amazon VPC. As before, the final rule is a universal rule that denies traffic unless it has been explicitly allowed by a preceding rule.

**TABLE 2.10** Default Network ACL Outbound Rules

| Rule Number | Type        | Protocol | Port Range | Destination | Allow/Deny |
|-------------|-------------|----------|------------|-------------|------------|
| 100         | All traffic | All      | All        | 0.0.0.0/0   | Allow      |
| *           | All traffic | All      | All        | 0.0.0.0/0   | Deny       |

Figure 2.16 shows an example of an Amazon VPC with security groups protecting Amazon EC2 instances and network ACLs protecting subnets.

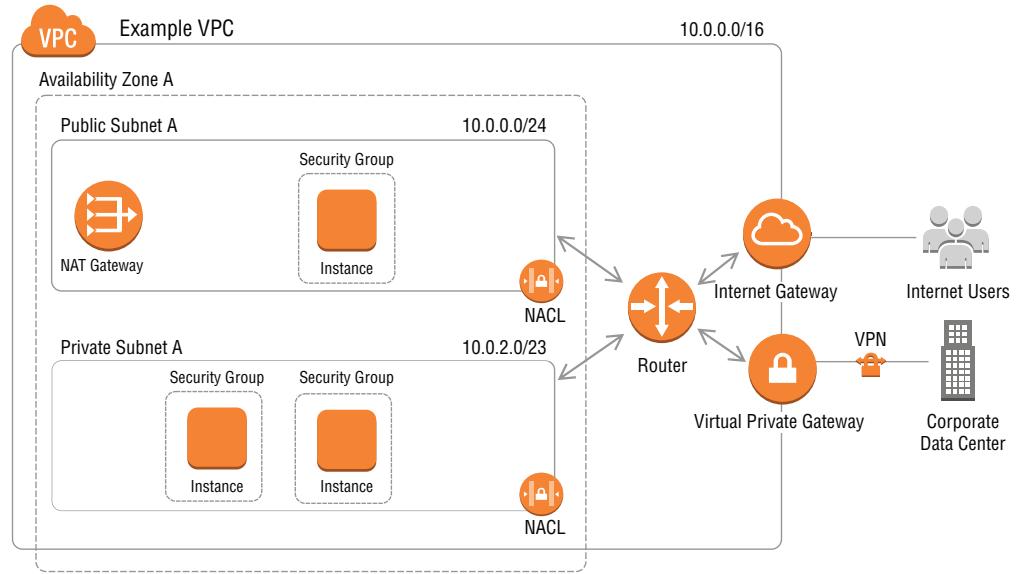
**FIGURE 2.16** Network ACLs and security groups

Figure 2.17 shows the same Amazon VPC, represented in a different way to highlight the features that control network traffic within an Amazon VPC.

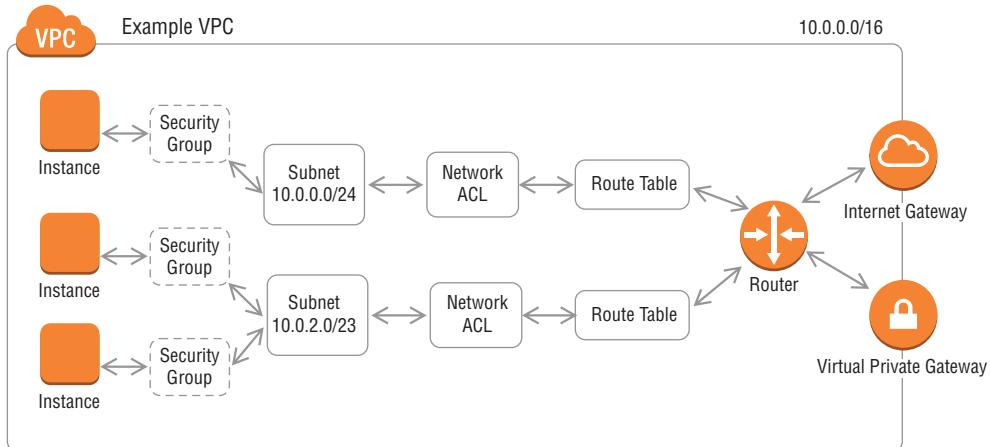
**FIGURE 2.17** Controlling network traffic within an Amazon VPC

Table 2.11 summarizes key aspects of security groups and network ACLs.

**TABLE 2.11** Security Groups and Network ACLs

| Feature          | Security Group                                                          | Network ACL                                                                               |
|------------------|-------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|
| Applies to       | Amazon EC2 instance or elastic network interface.                       | Subnet                                                                                    |
| Type of firewall | Stateful: Replies to an allowed traffic flow are automatically allowed. | Stateless: Must provide both inbound and outbound rules to allow a specific traffic flow. |
| Rules            | Only allow traffic.                                                     | Allow or deny traffic.                                                                    |

## Network Address Translation

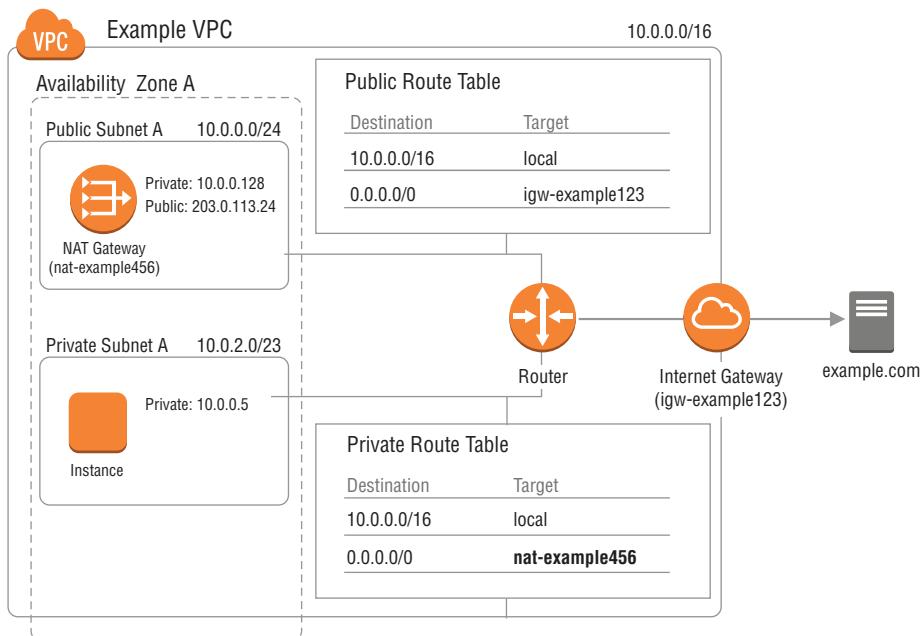
*Network address translation* (NAT) allows for instances in a private subnet to make outbound requests to the internet without exposing those instances to inbound connections from internet users. To provide NAT for outbound requests from private subnets, you can use an Amazon EC2 instance configured to perform NAT or a NAT gateway. The instances in the private subnet maintain their own private IP addresses and effectively share the public IP address of the NAT when making internet requests.

For the NAT to perform its job, you must place the NAT instance or a NAT gateway in a correctly configured public subnet to forward traffic to the internet. Make sure that the public subnet has a route to an internet gateway, as previously shown in Table 2.4. To support outbound network requests, you can associate the private subnet with a route table, similar to the one shown in Table 2.12.

**TABLE 2.12** Private Route Table Example

| Destination | Target         |
|-------------|----------------|
| 10.0.0.0/16 | local          |
| 0.0.0.0/0   | nat-example456 |

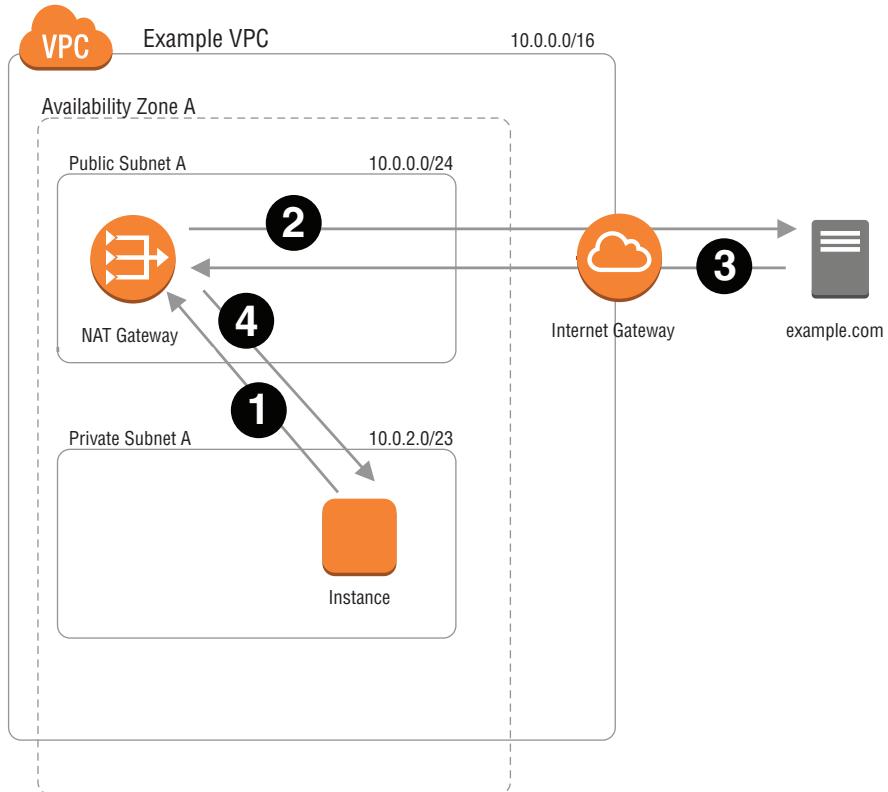
For the Amazon VPC configuration example, Figure 2.18 shows the corresponding route tables for the public and private subnets using the NAT gateway.

**FIGURE 2.18** Example of Amazon VPC with NAT

Internet-bound requests route to the NAT gateway through the route table of the private subnet. The NAT, located in a public subnet, then makes a corresponding request out to the internet. This second outbound request appears to have originated from the public IP address of the NAT when the external website received it. When the website responds, NAT receives the reply and forwards it to the instance that initiated the original request. Figure 2.19 shows the network flow.

Using an Amazon EC2 instance for NAT instead of a NAT gateway requires you to disable the *source/destination check* setting for the NAT instance.

This setting protects Amazon EC2 instances by requiring that the instance be the source or destination of any network traffic it receives. In the case of the NAT instance, while the packets are routed to the instance via a route table, they are addressed to destinations on the internet rather than the NAT. Disabling this setting allows the network traffic to be delivered to the NAT instance.

**FIGURE 2.19** NAT gateway in Amazon VPC

## DHCP Option Sets

The *Dynamic Host Configuration Protocol* (DHCP) provides a standard for passing configuration information to hosts on a TCP/IP network. The options field of a DHCP message contains the configuration parameters.

Some of those parameters are the address of domain name servers (DNS), the domain names of the instances, and the addresses of Network Time Protocol (NTP) servers. By default, the Amazon VPC uses the DNS that AWS provides. However, you can override these settings by specifying a custom set of DHCP options.

## Monitoring Amazon VPC Network Traffic

You can monitor the network flows within your Amazon VPC by enabling Amazon VPC Flow Logs. You can then publish these logs to Amazon CloudWatch Logs or store them as log files in Amazon Simple Storage Service (Amazon S3). Enable Amazon VPC Flow Logs on a particular Amazon VPC, on a subnet, or on a specific elastic network interface, such as for an Amazon EC2 instance.

For each network session, the Flow Logs capture metadata, such as the source, destination, protocol, port, packet count, byte count, and time interval. The log entry specifies whether the traffic was accepted or rejected. This information helps you debug the network configuration.

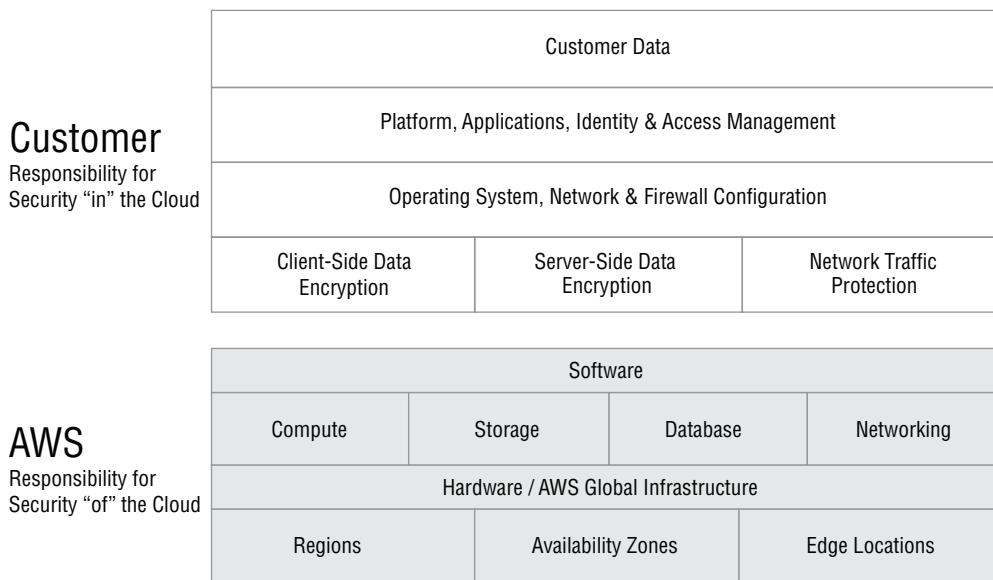
# Managing Your Resources

You now know that you can customize the software running on your Amazon EC2 instances. Additionally, with Amazon VPC, you can control the virtual network for those instances through security groups, network ACLs, subnets, and route tables. Because you can configure your operating environment in AWS, you share the responsibility for securing your applications running in the AWS Cloud with AWS.

## Shared Responsibility Security Model

AWS is responsible for the security *of* the cloud. This involves securing physical access to the underlying infrastructure, such as the AWS Regions and Availability Zones. This responsibility includes procedures for restricting access to the servers, physical networks, and decommissioning of hardware that is no longer useful. As part of securing the cloud infrastructure, AWS is also responsible for maintaining the underlying software for each service provided.

As the AWS customer, you are responsible for security *on* the cloud. This responsibility includes making secure choices when configuring your infrastructure and developing your applications. These responsibilities can include configuring the relevant encryption options and configuring your firewall rules. Even though this is your responsibility, you can simplify this task by taking advantage of AWS tools for encryption, defining firewall rules, and managing access and authorization to your AWS resources. For a summary of AWS and customer responsibilities, see Figure 2.20.

**FIGURE 2.20** Shared responsibility security model

For example, with Amazon EC2, AWS is responsible for the software on the physical host machines up through maintaining the virtualization layer. However, beyond that, it is your responsibility to ensure that the guest operating system and everything running on it are secured. Your responsibilities include the following tasks:

- Making sure that any sensitive data is secured
- Making sure that the guest operating system is patched regularly
- Managing the guest operating system's user accounts
- Securing any applications that are installed on that instance

AWS provides tools to help you manage these concerns. For example, use AWS Systems Manager to automate the patching of instances on your behalf. You can also use network controls, such as security groups, to restrict access to the instance. However, it is your responsibility to configure these features in a way that meets the security requirements for your specific application.

## Comparing Managed and Unmanaged Services

Even though services such as Amazon EC2 provide many low-level customizations, other AWS services provide a managed experience. When you use AWS managed services, you may find that you have less responsibility. For example, for Amazon RDS, AWS manages the software installed on the underlying database instance. You interact with the database using your SQL client rather than a shell session, and you do not install your own software

on the database instance. In this case, parts of your operational burden for security are reduced—AWS manages the operating system and software on the database instance.

## Developer Tools

In addition to providing the low-level infrastructure pieces, such as Amazon EC2 instances and VPC networking components, the AWS Cloud provides higher-level services to help developers be more productive.

With AWS Cloud9, you can create developer environments that execute on either an Amazon EC2 instance or another server. AWS Cloud9 provides a web interface for editing code, debugging, and running commands. It supports more than 40 programming languages, and it can automatically configure the AWS SDK to use short-term managed credentials.

To access an AWS Cloud9 environment, sign in to the AWS Management Console. Within the AWS Cloud9 environment, the files you edit and run are on the remote instance or server. If you choose to run the AWS Cloud9 environment on an EC2 instance, you can customize the underlying EC2 instance as needed, even after the environment has been created. For example, you can edit the security groups of the instance or increase the size of the Amazon EBS volume attached to the instance.

This service simplifies the common task of developing code, running it on Amazon EC2, and collaborating with other developers. If you want to reduce cost, one advantage of AWS Cloud9 is that it can automatically stop the underlying instance a short time after you close your browser window, and it automatically starts the instance the next time you try to connect. You can further explore EC2 instances, VPCs, and AWS Cloud9 environments in the exercises that accompany this chapter.

## Summary

Amazon Elastic Compute Cloud (Amazon EC2) instances are compute environments that provide you with full control over the operating system and software. The instance type and instance size determine the hardware available to an instance. This includes properties such as vCPU, RAM, access to local storage, and network bandwidth. Amazon Elastic Block Store (Amazon EBS) provides persistent storage for EC2 instances. An Amazon Machine Image (AMI) provides the template for the software on the instance. Additionally, user data allows you to run a script on the instance to automatically update the software on the instance. To make AWS API calls from code running on an EC2 instance, assign an AWS Identity and Access Management (IAM) role to the instance by way of an instance profile. Use Amazon CloudWatch to collect instance monitoring and utilization metrics.

Amazon Virtual Private Cloud (Amazon VPC) enables your EC2 instances to be placed into isolated networks where you have control over the connectivity to other networks, such as the internet, on-premises networks, or other VPCs. Within a VPC, the network is segmented

into subnets. Instances within a subnet in a VPC are assigned private IPv4 addresses. They can be assigned public IPv4 addresses, Elastic IP addresses, or IPv6 addresses.

Routing between the instances in the VPC and other networks is controlled on a subnet level using routes and route tables. This configuration enables you to define some subnets as public and others as private. In addition to routing, network traffic can also be controlled by two sets of controls that act as firewalls in a VPC. Network access control lists (network ACLs) act as a stateless firewall on all traffic that leaves or enters a subnet. Security groups act as a stateful firewall that protects individual traffic flows at an instance level.

The responsibility for keeping your instances secure is shared between AWS and you, the customer. AWS is responsible for securing access to the infrastructure and providing you with controls that you can use to secure your instances. As an AWS customer, you are responsible for configuring your resources in a way that is secure and meets your application needs.

## Exam Essentials

**Know the basics of Amazon EC2, such as resource types, instance types, AMIs, and storage.** Be familiar with launching and connecting to Amazon EC2 instances. Understand the resource types of Amazon EC2 instance types. Be familiar with the purpose of an AMI in relation to launching an instance. Understand the distinction between persistent and ephemeral storage related to a particular Amazon EC2 instance.

**Know about user data, instance metadata, and credentials.** Be familiar with using user data to customize the software by executing scripts on instances. Any scripts or code running on an instance can use the Amazon EC2 metadata service to discover the instance configuration. Use IAM roles to provide AWS Cloud API credentials automatically to code running on an Amazon EC2 instance.

**Know how Amazon EC2 communicates with Amazon VPC.** Understand the relationship between an EC2 instance and the Amazon VPC network. There may be questions that ask you to troubleshoot issues related to connecting to an Amazon EC2 instance. Be familiar with how Amazon VPC enables communication between Amazon EC2 instances within the same Amazon VPC and isolates those instances from other Amazon VPCs. Recognize how route tables, network access control lists, and security groups control network traffic.

**Know about public and private subnets.** Within an Amazon VPC, you must be able to distinguish between public and private subnets. Public subnets allow you to assign public IPv4 addresses to Amazon EC2 instances. By contrast, instances in a private subnet have only private IP addresses. The key distinction is that public subnets have a route table entry that forwards internet-bound traffic to an internet gateway. Private subnets do not have a direct route to the internet. Instead, these subnets have a route that forwards internet-bound traffic through a NAT gateway or NAT instance.

**Know about security groups and network ACLs.** Be familiar with security groups and network ACLs. Security groups are used with Amazon EC2 instances, acting as stateful firewalls. They provide only rules that allow traffic. In comparison, network ACLs allow traffic between subnets and are stateless. They can allow or deny specific types of traffic.

**Know about responsibilities shared between you and AWS.** Be familiar with the separation between AWS responsibility and your responsibility concerning Amazon EC2 instances. AWS is responsible for providing secure building blocks up until the hypervisor layer for the Amazon EC2 instance. This includes securing the physical facilities and machines and any hardware decommissioning. You are responsible for patching the guest operating system and applications. You are also responsible for configuring firewall rules, encryption, and access to the instance in a way that meets their requirements.

## Resources to Review

Amazon EC2 Instance Types:

<https://aws.amazon.com/ec2/instance-types/>

Amazon EC2 User Guide for Linux Instances:

<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/index.html>

Amazon EC2 User Guide for Windows Instances:

<https://docs.aws.amazon.com/AWSEC2/latest/WindowsGuide/index.html>

Amazon EC2 Foundations on YouTube:

<https://www.youtube.com/watch?v=bgoPfn-Ppd8>

Amazon EC2 Instance Metadata and User Data:

<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-metadata.html>

Amazon EC2 FAQs:

<https://aws.amazon.com/ec2/faqs/>

Amazon VPC User Guide:

<https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html>

Amazon VPC Fundamentals and Connectivity on YouTube:

<https://www.youtube.com/watch?v=Tff1mekx0J4>

Amazon VPC Security:

[https://docs.aws.amazon.com/vpc/latest/userguide/VPC\\_Security.html](https://docs.aws.amazon.com/vpc/latest/userguide/VPC_Security.html)

IP Addressing in Your VPC:

<https://docs.aws.amazon.com/vpc/latest/userguide/vpc-ip-addressing.html>

Amazon VPC FAQs:

<https://aws.amazon.com/vpc/faqs/>

AWS Cloud9 User Guide:

<https://docs.aws.amazon.com/cloud9/latest/user-guide/welcome.html>

# Exercises

These exercises provide hands-on experience with the fundamentals of working with Amazon EC2 and Amazon VPC. You will create an isolated network in the AWS Cloud and then launch Amazon EC2 instances into that network. The exam has questions that test your knowledge of how to troubleshoot common network-connectivity issues relating to Amazon EC2 instances.

For the following exercises, verify that the region is US West (Oregon). The directions for these exercises assume that you have already completed Exercises 1.1, 1.2, 1.3, and 1.4 in Chapter 1, “Introduction to AWS Cloud API.”

You can complete these exercises within the AWS Free Tier, provided that you follow the steps to clean up resources promptly.



The results from these exercises are used in a later chapter, so follow all the activities and directions exactly.

## EXERCISE 2.1

### Create an Amazon EC2 Key Pair

In this exercise, you’ll generate and save an Amazon EC2 key pair. You are responsible for saving the private key and using it when you want to connect to your Amazon EC2 instances.

1. Sign in to the AWS Management Console using the **DevAdmin** IAM user you created in Exercise 1.2.
2. To open the Amazon EC2 console, select **Services > EC2**.
3. Select **Network & Security > Key Pairs**.
4. Select **Create Key Pair**.
5. For **Key pair name**, enter **devassoc**, and then choose **Create**.

The key pair automatically downloads to your Downloads folder.

6. Move this key to a safe location on your computer. You need it to connect to your Amazon EC2 instances using Secure Shell (SSH) or Remote Desktop Protocol (RDP).

**EXERCISE 2.2****Create an Amazon VPC with Public and Private Subnets**

In this exercise, you'll create an Amazon Virtual Private Cloud (Amazon VPC). Within that Amazon VPC, you will have a public subnet directly connected to the internet through an internet gateway. You will also have a private subnet that only has an indirect connection to the internet using network address translation (NAT).

1. To display the Amazon VPC dashboard, select **Services > VPC**.

2. Select **Launch VPC Wizard**.

If a field does not contain an explicit value in these directions, retain the default value.

3. Select **VPC with Public and Private Subnets** and then click **Select**.

4. Enter the following details for Amazon VPC:

- a. For **Amazon VPC name**, enter **devassoc**.
- b. In the public and private subnets drop-down lists, select the first **AZ**.
- c. In the **Elastic IP Allocation ID** prompt, select **Use a NAT instance instead**.
- d. For the **Amazon EC2 Key Pair Name**, select **devassoc**.

5. Choose **Create VPC**.

6. When the Amazon VPC is created, choose **OK**.

7. Copy the VPC ID of the Amazon VPC named devassoc to a text document.

8. To view the list of subnets, select **Subnets**. In the filter box, paste the VPC ID you copied and then press the **Enter** key to filter the results.

Two subnets are listed: **Public subnet** and **Private subnet**.

9. Copy the **Subnet ID** of the public and private subnet to the text file.

After you have created Amazon VPC, your text document will look like the following:

VPC ID: VPC-06bb2198eaexample

Public subnet ID: subnet-0625e239a2example

Private subnet ID: subnet-0e78325d9eexample

---

**EXERCISE 2.3****Use an IAM Role for API Calls from Amazon EC2 Instances**

In this exercise, you'll create an IAM role for the web server. This role enables you to make AWS Cloud API calls from code running on the Amazon EC2 instance of the web server. You are not required to save IAM credentials in a file on the instance. To do this, create a new IAM role and call it the devassoc-webserver role. The role provides permissions needed for the API calls.

1. Select **Services > IAM**.
2. Select **Roles** and choose **Create Role**.
3. Under **Choose the service that will use this role**, select the option that allows Amazon EC2 instances to call AWS services on your behalf, and then choose **Next: Permissions**.
4. Select the following AWS managed policies to attach them to the devassoc-webserver role and then choose **Next: Tags**:

**AmazonPollyReadOnlyAccess:** Grant read-only access to resources, list lexicons, fetch lexicons, list available voices, and synthesize speech to apply lexicons to the synthesized speech.

**TranslateReadOnly:** Allow permissions to detect the dominant language in text, translate text, and list and retrieve custom terminologies.



These permissions are required to complete future exercises.

5. Enter the following tag details and then choose **Next: Review**:

**Key: project**

**Value: devassoc**

6. For the **Role name**, enter **devassoc-webserver** and then choose **Create Role**.

**EXERCISE 2.4****Launch an Amazon EC2 Instance as a Web Server**

In this exercise, you'll launch an Amazon EC2 instance as a web server and connect to it.

1. Select **Services > EC2**.
2. Select **Launch Instance**.
3. Select **Amazon Linux 2 AMI** and then choose **Next**.
4. Select **t2.micro** and then choose **Next: Configure Instance Details**.

*(continued)*

**EXERCISE 2.4 (continued)**

5. On the **Configure Instance Details** page, set the instance:
  - Select **Network** > **devassoc**.
  - Select **Subnet** > **Public Subnet**.
  - Select **Auto-assign Public IP** > **Enable**.
  - Select **IAM Role** > **devassoc-webserver**.
6. Expand **Advanced Details** and then paste the **User Data** script.

```
#!/bin/bash
yum install httpd -y
systemctl start httpd
systemctl enable httpd
```



Paste this snippet from chapter-02/server-short.txt, located in the folder in which you downloaded the sample code for this guide.

7. Select **Next: Add Storage**.
8. Select **Next: Add Tags**.
9. On the **Add Tags** page, choose **Add Tag** and then enter the following:

**Key:** Name

**Value:** webserver
10. Choose **Next: Configure Security Group**.
11. On the **Configure Security Group** page:
  - For **Security group name**, enter **restricted-http-ssh**.
  - For **Description**, enter **HTTP and SSH from my IP address only**.
12. For the existing SSH rule, select **Source** > **My IP**.
13. Select **Add Rule**, and then configure the second rule:
  - For **Type**, select **HTTP**.
  - For **Source**, select **My IP**.
14. Choose **Review and Launch**.
15. On the **Review Instance Launch** page, verify the settings and then choose **Launch**.
16. Under **Select a key pair**, select **devassoc** and select the box acknowledging that you have access to the key pair.
17. To launch your EC2 instance, choose **Launch Instances**.
18. To find your instance, choose **View Instances**.

19. From the list of instances, select **webserver**. Wait until **Instance Status** for your instance reads as running and **Status Checks** changes to 2/2 checks passed.
  20. Copy the **Public IPv4 address** of the instance to a text document.
  21. Paste the **IP address** of the **webserver** instance into a browser window.

A test webpage is displayed. If you do not see a page, wait 30 seconds and then refresh the page.
  22. Disable your mobile phone's Wi-Fi and then attempt to access the IP address of the **webserver** instance from your mobile phone with mobile data.
- The page fails to load because the security group rule allows HTTP access from only a particular IP address.
- 

## EXERCISE 2.5

### Connect to the Amazon EC2 Instance

In this exercise, you'll connect to the Amazon EC2 Instance using SSH.

1. Select **Services > EC2**.
2. Select **Instances**.
3. Select the **webserver** instance from the list of instances.
4. Select **Actions > Connect**.
5. In the **Connect to Your Instance** dialog box, follow the directions to establish an SSH connection.
6. From within your SSH session, run this command to view the available metadata fields from the Amazon EC2 metadata service:  
`curl 169.254.169.254/latest/meta-data/`
7. Run this command to query the Amazon EC2 instance ID:  
`curl 169.254.169.254/latest/meta-data/instance-id`
8. Call **AWS Cloud API** using the AWS CLI. This command translates text from English to French and uses credentials from the AWS role you assigned to the instance. Enter the following command as a single line:  
`aws translate translate-text --text "Hello world." --source-language-code en --target-language-code fr --region us-west-2`
9. To review the credentials that are being passed to the instance, query the Amazon EC2 Metadata service:  
`curl 169.254.169.254/latest/meta-data/iam/security-credentials/devassoc-webserver`

**EXERCISE 2.6****Configure NAT for Instances in the Private Subnet**

In this exercise, you'll create a security group for the NAT instance. NAT allows Amazon EC2 instances in the private subnet to make web requests to the internet, to update software packages, and to make API calls.

1. Select **Services > VPC**.
2. From the **Security** section, select **Security Groups**.
3. Select **Create Security Group** and configure the properties as follows:
  - Set the **Name** tag to **nat-sg**.
  - Set the **Group name** to **nat-sg**.
  - Set the **Description** to **Allow NAT instance to forward internet traffic**.
  - Set **Amazon VPC** to **devassoc**.
4. Choose **Create** to save the group, and then choose **Close** to return to the list of security groups.
5. Select the **nat-sg** security group.
6. To modify the inbound rules, select the **Inbound Rules** tab and select **Edit rules**.
7. Select **Add Rule**, and set the following properties for the first rule:
  - From **Type**, select **HTTP (80)**.
  - For **Source**, enter **10.0.0.0/16**.
  - For **Description**, enter **Enable internet bound HTTP requests from VPC instances**.
8. Select **Add Rule**, and set the following properties for this rule:
  - From **Type**, select **HTTPS (443)**.
  - For **Source**, enter **10.0.0.0/16**.
  - For **Description**, enter **Enable internet bound HTTPS requests from VPC instances**.
9. Select **Add Rule**, and set the following properties for this rule:
  - From **Type**, select **All ICMP – IPv4**.
  - For **Source**, enter **10.0.0.0/16**.
  - For **Description**, **Enable outbound PING requests from VPC instances**.
10. Choose **Save rules**.

- 
11. Select **Services > EC2**.
  12. Select **Instances**.
  13. Paste the **Public subnet ID** into the filter box.

Two results are displayed. The result with an empty name is your **NAT instance**.

14. To edit the name of the NAT instance, hover over the **name** field and select the **pencil** icon.
  15. Enter the name **devassoc-nat** and press **Enter**.
  16. Modify the security groups for **devassoc-nat** to include the **nat-sg** group as follows:
    - Select the **devassoc-nat** instance and select **Actions**.
    - Select **Networking > Change Security Groups**.
    - Select **nat-sg**. You can clear the default.
    - Select **Assign Security Group**.
- 

## EXERCISE 2.7

### Launch an Amazon EC2 Instance into the Private Subnet

In this exercise, you'll launch an Amazon EC2 instance into the private subnet and then verify that the security group allows HTTP from anywhere. Because this instance is in the private subnet, it does not have a public IP address. Even though the instance can make outbound requests to the internet through the NAT instance, it is not reachable for inbound connections from the internet.

1. Select **Services > EC2**.
2. Choose **Launch Instance**.
3. Select **Amazon Linux 2 AMI**.
4. Select **t2.micro** and then choose **Next: Configure Instance Details**.
5. On the **Instance Details** page, provide the following values:
  - Select **Network > devassoc VPC**.
  - Select **Subnet > Private Subnet**.
  - Select **IAM Role > devassoc-webserver**.
  - Select **Advanced Details > User Data > As File**.
  - From the folder where you downloaded the samples for this guide, select **Choose File > chapter-02/server-polly.txt**.

---

(continued)

**EXERCISE 2.7 (continued)**

6. Choose **Next: Add Storage**.
7. Choose **Next: Add Tags**.
8. Select **Tags > Add tag** and set the following values:
  - For **Key**, enter **Name**.
  - For **Value**, enter **private-instance**.
9. Choose **Next: Configure Security Group**.
10. For **Security Group**, set the following values:
  - For **Security group name**, enter **open-http-ssh**.
  - For **Description**: enter **HTTP and SSH from Anywhere**.
11. For the SSH rule, select **Source > Anywhere**.
12. Select **Add Rule** and then configure the second rule:
  - For **Type**, select **HTTP**.
  - For **Source**, select **Anywhere**.
13. Choose **Review and Launch**.
14. Choose **Launch**.
15. Under **Select a key pair**, choose **devassoc** and select the check box acknowledging that you have access to the key pair.
16. Choose **Launch Instances**.
17. Select **View Instances**.
18. Select **private-instance**.
19. Copy the **Private IP** of the instance to the text document.

Notice that the instance has no public IP address.

---

**EXERCISE 2.8****Make Requests to Private Instance**

In this exercise, you will explore connectivity to the private instance.

1. From your web browser, navigate to the private IP of the instance. Though the security group is open to requests from anywhere, this will fail because the private IP address is not routable over the internet.
-

2. Select **Services > EC2**.
3. From the list of instances, select **webserver**.
4. Select **Connect** and then follow the directions to establish an SSH connection.
5. From within the SSH session, make an HTTP request to the private server with curl. Replace the variable **private-ip-address** with the private IP address of **private-instance** address that you copied earlier.  
`curl private-ip-address`
6. Download the MP3 audio from the **private-instance** to **webserver** using curl as follows:  
`curl private-ip-address/instance.mp3 --output instance.mp3`
7. Make the file available for download from **webserver**:  
`sudo cp instance.mp3 /var/www/html/instance.mp3`
8. In your web browser, enter the following address. Substitute **public-ip-of-webserver** with the public IPv4 address of **webserver**, and listen to the MP3.  
`http://public-ip-of-webserver/instance.mp3`

Though the private web server is not reachable from the internet, you have confirmed that it is reachable to other instances within the same Amazon VPC. As part of the bootstrapping, the private instance made AWS API calls, which require the ability to make both web requests via the NAT gateway and credentials from an IAM role. You have confirmed that these requests succeeded by downloading the resulting MP3 file from **private-instance** and placing it on **webserver**.

---

## EXERCISE 2.9

### Launch an AWS Cloud9 Instance

In this exercise, you'll launch an Amazon EC2 instance that you will create in the AWS Cloud9 service. You will connect to this Amazon EC2 instance from the AWS Cloud9 console. You will then use the AWS Cloud9 IDE to edit files, build software, and execute commands on the terminal from your web browser.

1. To display the AWS Cloud9 dashboard, select **Services > Cloud9**.
2. Select **Create Environment**.
3. For **Name**, enter **devassoc-c9** and then select **Next step**.
4. Select **Network settings > Advanced**.
5. Select **Network (VPC) > Amazon VPC ID** (copied earlier).

---

(continued)

**EXERCISE 2.9 (continued)**

6. Select **Subnet** > **Subnet ID for the Public VPC** (copied earlier).
7. Select **Next step** > **Create environment**.
8. When the AWS Cloud9 environment loads, run the following in the AWS Cloud9 terminal. Make sure to replace the IP address in the example command with the address you copied earlier for **private-instance**.

```
curl private-ip
```
9. From the **private-instance**, download the MP3 audio to **devassoc-c9** using curl as follows:

```
curl private-ip-address/instance.mp3 --output instance.mp3
```
10. To preview the file, in the navigation pane, double-click **instance.mp3**.
11. Open **README.md** in a text editor.

You now have a managed development environment in AWS that is connected to your isolated VPC.

---

**EXERCISE 2.10****Perform Partial Cleanup**

In this exercise, you will clean up unused instances and keep this Amazon VPC for future use. This partial cleanup reduces costs while providing an environment to complete future exercises. After partial cleanup, you may generate charges related to the Elastic IP address that was allocated for devassoc-nat but is not in use while that instance is stopped.

Complete the following tasks as part of the cleanup:

**webserver**: Terminate.

**private-instance**: Terminate.

**devassoc-nat**: Stop. You must start this instance again before completing any exercises that require Amazon EC2 to launch or interact with instances in the private subnet.

**devassoc-c9**: No action. The AWS Cloud9 service will automatically stop and start this instance.

1. Navigate to the **Services** > **EC2**.
  2. To view your Amazon EC2 instances, select **Instances**. Clear any filters if they are present.
-

- 
3. Select **webserver** and **private-instance**.
  4. Select **Actions > Instance-State** and **Terminate**.
  5. Clear **public-webserver** and **private-webserver**.
  6. Select **devassoc-nat**.
  7. Select **Actions > Instance-State** and **Stop**.
- 

## EXERCISE 2.11

### (Optional) Complete Cleanup

If you plan to perform future exercises in this guide, this exercise is optional.

In this exercise, you will remove all of the EC2 and VPC resources that remain after Exercise 2.10.

1. Navigate to the Amazon EC2 console, and view the list of running instances.
2. Select **devassoc-nat**.
3. Select **Actions > Instance-State** and **Terminate**.
4. In the **Terminate Instances** dialog box, expand **Release attached Elastic IPs** and select **Release Elastic IPs**.
5. Select **Yes, Terminate**.
6. Navigate to the AWS Cloud9 dashboard (**Services > Cloud9**).
7. Select the **devassoc-c9** environment.
8. On the **Environment Details** page, select **Delete** and follow the on-screen directions to delete the instance.
9. To view the Amazon VPC dashboard, select **Services > VPC**.
10. Navigate to the **Elastic IPs** list.
11. Select any Elastic IPs that are *not associated with an instance*.
12. To release the Elastic IPs, select **Actions and Release Addresses**.
13. Select **Release**.
14. Select **Your VPCs**.
15. Select **devassoc**.
16. Select **Actions > Delete VPC**.
17. Select **Delete VPC**.

If the Amazon VPC deletion fails, wait up to 30 minutes after deleting the Amazon EC2 instances and then try again.

---

## Review Questions

1. When you launch an Amazon Elastic Compute Cloud (Amazon EC2) instance, which of the following is the most specific type of AWS entity in which you can place it?
  - A. Region
  - B. Availability Zone
  - C. Edge location
  - D. Data center
2. You have saved SSH connection information for an Amazon Elastic Compute Cloud (Amazon EC2) instance that you launched in a public subnet. You previously stopped the instance the last time you used it. Now that you have started the instance, you are unable to connect to the instance using the saved information. Which of the following could be the cause?
  - A. Your SSH key pair has automatically expired.
  - B. The public IP of the instance has changed.
  - C. The security group rules have expired.
  - D. SSH is enabled only for the first boot of an Amazon EC2 instance.
3. You are working from a new location today. You are unable to initiate a Remote Desktop Protocol (RDP) to your Windows instance, which is located in a public subnet. What could be the cause?
  - A. Your new IP address may not match the inbound security group rules.
  - B. Your new IP address may not match the outbound security group rules.
  - C. RDP is not available for Windows instances, only SSH.
  - D. RDP is enabled only for the first 24 hours of your instance runtime.
4. You have a backend Amazon EC2 instance providing a web service to your web server instances. Your web servers are in a public subnet. You would like to block inbound requests from the internet to your backend instance but still allow the instance to make API requests over the public internet. What steps must you take? (Select TWO.)
  - A. Launch the instance in a private subnet and rely on a NAT gateway in a public subnet to forward outbound internet requests.
  - B. Configure the security group for the instance to explicitly deny inbound requests from the internet.
  - C. Configure the network access control list (network ACL) for the public subnet to explicitly deny inbound web requests from the internet.
  - D. Modify the inbound security group rules for the instance to allow only inbound requests from your web servers.

5. You have launched an Amazon Elastic Compute Cloud (Amazon EC2) instance and loaded your application code on it. You have now discovered that the instance is missing applications on which your code depends. How can you resolve this issue?
  - A. Modify the instance profile to include the software dependencies.
  - B. Create an AWS Identity and Access Management (IAM) user, and sign in to the instance to install the dependencies.
  - C. Sign in to the instance as the default user, and install any additional dependencies that you need.
  - D. File an AWS Support ticket, and request to install the software on your instance.
6. How can code running on an Amazon Elastic Compute Cloud (Amazon EC2) instance automatically discover its public IP address?
  - A. The public IP address is presented to the OS on the instance automatically. No extra steps are required.
  - B. The instance can query another Amazon EC2 instance in the same Amazon Virtual Private Cloud (Amazon VPC).
  - C. You must use a third-party service to look up the public IP.
  - D. The instance can make an HTTP query to the Amazon EC2 metadata service at 169.254.169.254.
7. How can you customize the software of your Amazon Elastic Compute Cloud (Amazon EC2) instance beyond what the Amazon Machine Image (AMI) provides?
  - A. Provide a user data attribute at launch that contains a script or directives to install additional packages.
  - B. Additional packages are installed automatically by placing them in a special Amazon Simple Storage Service (Amazon S3) bucket in your account.
  - C. You do not have permissions to install new software on Amazon EC2 aside from what is in the AMI.
  - D. Unlock the instance using the AWS Key Management Service (AWS KMS) and then sign in to install new packages.
8. You have a process running on an Amazon Elastic Compute Cloud (Amazon EC2) instance that exceeds the 2 GB of RAM allocated to the instance. This is causing the process to run slowly. How can you resolve the issue?
  - A. Stop the instance, change the instance type to one with more RAM, and then start the instance.
  - B. Modify the RAM allocation for the instance while it is running.
  - C. Take a snapshot of the data and then launch a new instance. You cannot change the RAM allocation.
  - D. Send an email to AWS Support to install additional RAM on the server.

- 9.** You have launched an Amazon Elastic Compute Cloud (Amazon EC2) Windows instance, and you would like to connect to it using the Remote Desktop Protocol. The instance is in a public subnet and has a public IP address. How do you find the password to the Administrator account?
- A.** Decrypt the password by using the private key from the Amazon EC2 key pair that you used to launch the instance.
  - B.** Use the password that you provided when you launched the instance.
  - C.** Create a new AWS Identity and Access Management (IAM) role, and use the password for that role.
  - D.** Create an IAM user, and use the password for that user.
- 10.** What steps must you take to ensure that an Amazon EC2 instance can receive web requests from customers on the internet? (Select THREE.)
- A.** Assign a public IP address to the instance.
  - B.** Launch the instance in a subnet where the route table routes internet-bound traffic to an internet gateway.
  - C.** Launch the instance in a subnet where the route table rules send internet-bound traffic to a NAT gateway.
  - D.** Set the outbound rules for the security group to allow HTTP and HTTPS traffic.
  - E.** Set the inbound rules for the security group to allow HTTP and HTTPS traffic.
- 11.** Which of the following are true about Amazon Machine Images (AMI)? (Select TWO.)
- A.** AMI can be used to launch one or multiple Amazon EC2 instances.
  - B.** AMI is automatically available in all AWS Regions.
  - C.** All AMIs are created and maintained by AWS.
  - D.** AMIs are available for both Windows and Linux instances.
- 12.** Which of the following are true about Amazon Elastic Compute Cloud (Amazon EC2) instance types? (Select TWO.)
- A.** All Amazon EC2 instance types include instance store for ephemeral storage.
  - B.** All Amazon EC2 instance types can use EBS volumes for persistent storage.
  - C.** Amazon EC2 instances cannot be resized once launched.
  - D.** Some Amazon EC2 instances may have access to GPUs or other hardware accelerators.
- 13.** Which of the following actions are valid based on the Amazon Elastic Compute Cloud (Amazon EC2) instance lifecycle? (Select TWO.)
- A.** Starting a previously terminated instance
  - B.** Starting a previously stopped instance
  - C.** Rebooting a stopped instance
  - D.** Stopping a running instance

- 14.** You have a development Amazon Elastic Compute Cloud (Amazon EC2) instance where you have installed Apache Web Server and MySQL. How do you verify that the web server application can communicate with the database given that they are both running on the same instance?
- A.** Modify the security group for the instance.
  - B.** Assign the instance a public IP address.
  - C.** Modify the network access control list (network ACL) for the instance.
  - D.** No extra configuration is required.
- 15.** What type of route must exist in the associated route table for a subnet to be a public subnet?
- A.** A route to a VPN gateway
  - B.** Only the local route is required.
  - C.** A route to an internet gateway
  - D.** A route to a NAT gateway or NAT instance
  - E.** A route to an Amazon VPC endpoint
- 16.** What type of route must exist in the associated route table for a subnet to be a private subnet that allows outbound internet access?
- A.** A route to a VPN gateway
  - B.** Only the local route is required.
  - C.** A route to an internet gateway
  - D.** A route to a NAT gateway or NAT instance
  - E.** A route to an Amazon Virtual Private Cloud (Amazon VPC) endpoint
- 17.** Which feature of Amazon Virtual Private Cloud (Amazon VPC) enables you to see which network requests are being accepted or rejected in your Amazon VPC?
- A.** Internet gateway
  - B.** NAT gateway
  - C.** Route table
  - D.** Amazon VPC Flow Log
- 18.** Which AWS service enables you to track the CPU utilization of an Amazon Elastic Compute Cloud (Amazon EC2) instance?
- A.** AWS Config
  - B.** AWS Lambda
  - C.** Amazon CloudWatch
  - D.** Amazon Virtual Private Cloud (Amazon VPC)
- 19.** What happens to the data stored on an Amazon Elastic Block Store (Amazon EBS) volume when you stop an Amazon Elastic Compute Cloud (Amazon EC2) instance?
- A.** The data is moved to Amazon Simple Storage Service (Amazon S3).
  - B.** The data persists in the EBS volume.
  - C.** The volume is deleted.
  - D.** An EBS-backed instance cannot be stopped.

- 20.** Which programming language can you use to write the code that runs on an Amazon EC2 instance?
- A. C++
  - B. Java
  - C. Ruby
  - D. JavaScript
  - E. Python
  - F. All of the above
- 21.** You have launched an Amazon EC2 instance in a public subnet. The instance has a public IP address, and you have confirmed that the Apache web server is running. However, your internet users are unable to make web requests to the instance. How can you resolve the issue? (Select TWO.)
- A. Modify the security group to allow outbound traffic on port 80 to anywhere.
  - B. Modify the security group for the web server to allow inbound traffic port 80 from anywhere.
  - C. Modify the security group for the web server to allow inbound traffic on port 443 from anywhere.
  - D. Modify the security group to allow outbound traffic from port 443 to anywhere.
- 22.** Which of the following are the customer's responsibility concerning Amazon EC2 instances? (Select TWO.)
- A. Decommissioning storage hardware
  - B. Patching the guest operating system
  - C. Securing physical access to the host machine
  - D. Managing the sign-in accounts and credentials on the guest operating system
  - E. Maintaining the software that runs on the underlying host machine

# Chapter

# 3



# Hello, Storage

---

**THE AWS CERTIFIED DEVELOPER –  
ASSOCIATE EXAM TOPICS COVERED IN  
THIS CHAPTER MAY INCLUDE, BUT ARE  
NOT LIMITED TO, THE FOLLOWING:**

**Domain 2: Security**

- ✓ 2.2 Implement encryption using AWS services.
- ✓ 2.3 Implement application authentication and authorization.

**Domain 3: Development with AWS Services**

- ✓ 3.2 Translate functional requirements into application design.
- ✓ 3.3 Implement application design into application code.
- ✓ 3.4 Write code that interacts with AWS services by using APIs, SDKs, and AWS CLI.

**Domain 4: Refactoring**

- ✓ 4.1 Optimize application to best use AWS services and features.

**Domain 5: Monitoring and Troubleshooting**

- ✓ 5.2 Perform root cause analysis on faults found in testing or production.



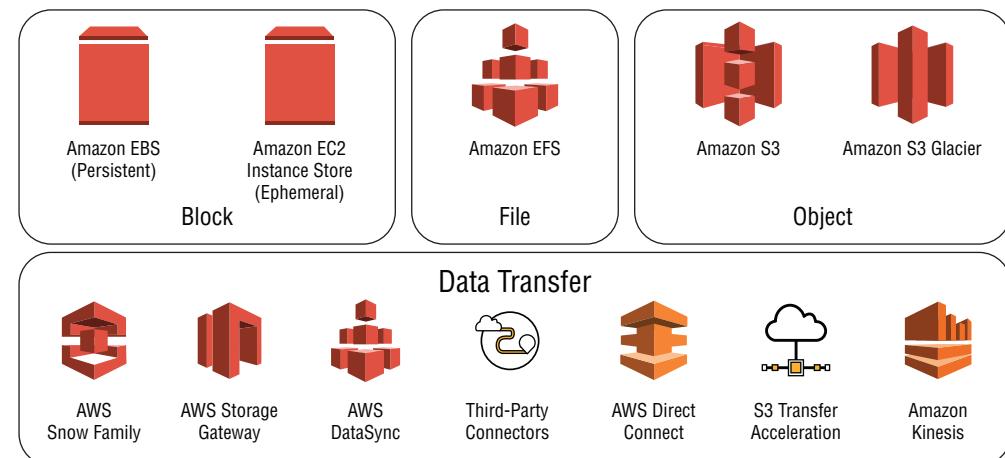
# Introduction to AWS Storage

Cloud storage is a critical component of cloud computing, holding the information used by applications built by developers. In this chapter, we will walk you through the portfolio of storage services that AWS offers and decompose some phrases that you might have heard, such as *data lake*.

The internet era brought about new challenges for data storage and processing, which prompted the creation of new technologies. The latest generation of data stores are no longer multipurpose, single-box systems. Instead, they are complex, distributed systems optimized for a particular type of task at a particular scale. Because no single data store is ideal for all workloads, choosing a data store for the entire system will no longer serve you well. Instead, you need to consider each individual workload or component within the system and choose a data store that is right for it.

The AWS Cloud is a reliable, scalable, and secure location for your data. Cloud storage is typically more reliable, scalable, and secure than traditional, on-premises storage systems. AWS offers *object storage*, *file storage*, *block storage*, and data transfer services, which we will explore in this chapter. Figure 3.1 shows the storage and data transfer options on AWS.

**FIGURE 3.1** The AWS storage portfolio



This chapter covers how to provision storage using just-in-time purchasing, which helps you avoid overprovisioning and paying for unused storage into which you expect to grow eventually.

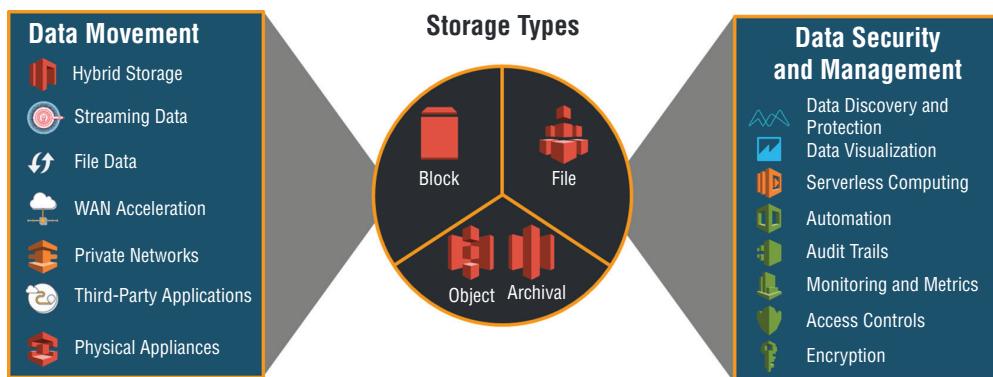
# Storage Fundamentals

Before we explore the various AWS storage services, let's review a few storage fundamentals. As a developer, you are likely already familiar with block storage and the differences between hot and cold storage. Cloud storage introduces some new concepts such as object storage, and we will compare these new concepts with the traditional storage concepts with which you are already familiar. If you have been working on the cloud already, these fundamentals are likely a refresher for you.

The goal of this chapter is to produce a mental model that will allow you, as a developer, to make the right decisions for choosing and implementing the best storage options for your applications. With the right mental model, people can usually make the best decisions for their solutions.

The AWS storage portfolio mental model starts with the core data building blocks, which include block, file, and object storage. For block storage, AWS has *Amazon Elastic Block Store* (Amazon EBS). For file storage, AWS has *Amazon Elastic File System* (Amazon EFS). For object storage, AWS has *Amazon Simple Storage Service* (Amazon S3) and Amazon S3 Glacier. Figure 3.2 illustrates this set of storage building blocks.

**FIGURE 3.2** A complete set of storage building blocks



## Data Dimensions

When investigating which storage options to use for your applications, consider the different dimensions of your data first. In other words, find the right tool for your data instead of squeezing your data into a tool that might not be the best fit.

So, before you start considering storage options, take time to evaluate your data and decide under which of these dimensions your data falls. This will help you make the correct decisions about what type of storage is best for your data.



Think in terms of a data storage mechanism that is most suitable for a particular workload—not a single data store for the entire system. Choose the right tool for the job.

## Velocity, Variety, and Volume

The first dimension to consider comprises the three Vs of big data: velocity, variety, and volume. These concepts are applicable to more than big data. It is important to identify these traits for any data that you are using in your applications.

**Velocity** *Velocity* is the speed at which data is being read or written, measured in *reads per second* (RPS) or *writes per second* (WPS). The velocity can be based on batch processing, periodic, near-real-time, or real-time speeds.

**Variety** *Variety* determines how structured the data is and how many different structures exist in the data. This can range from highly structured to loosely structured, unstructured, or *binary large object* (BLOB) data.

Highly structured data has a predefined schema, such as data stored in relational databases, which we will discuss in Chapter 4, “Hello, Databases.” In highly structured data, each entity of the same type has the same number and type of attributes, and the domain of allowed values for an attribute can be further constrained. The advantage of highly structured data is its self-described nature.

Loosely structured data has entities, which have attributes/fields. Aside from the field uniquely identifying an entity, however, the attributes are not required to be the same in every entity. This data is more difficult to analyze and process in an automated fashion, putting more of the burden of reasoning about the data on the consumer or application.

Unstructured data does not have any sense or structure. It has no entities or attributes. It can contain useful information, but it must be extracted by the consumer of the data.

BLOB data is useful as a whole, but there is often little benefit in trying to extract value from a piece or attribute of a BLOB. Therefore, the systems that store this data typically treat it as a black box and only need to be able to store and retrieve a BLOB as a whole.

**Volume** *Volume* is the total size of the dataset. There are two main uses for data: developing valuable insight and storage for later use. When getting valuable insights from data, having more data is often preferable to using better models. When keeping data for later use, be it for digital assets or backups, the more data that you can store, the less you need to guess what data to keep and what to throw away. These two uses prompt you to collect as much data as you can store, process, and afford to keep.

Typical metrics that measure the ability of a data store to support volume are maximum storage capacity and cost (such as \$/GB).

## Storage Temperature

**Data temperature** is another useful way of looking at data to determine the right storage for your application. It helps us understand how “lively” the data is: how much is being written or read and how soon it needs to be available.

**Hot** *Hot data* is being worked on actively; that is, new ingests, updates, and transformations are actively contributing to it. Both reads and writes tend to be single-item. Items tend to be small (up to hundreds of kilobytes). Speed of access is essential. Hot data tends to be high-velocity and low-volume.

**Warm** *Warm data* is still being actively accessed, but less frequently than hot data. Often, items can be as small as in hot workloads but are updated and read in sets. Speed of access, while important, is not as crucial as with hot data. Warm data is more balanced across the velocity and volume dimensions.

**Cold** *Cold data* still needs to be accessed occasionally, but updates to this data are rare, so reads can tolerate higher latency. Items tend to be large (tens of hundreds of megabytes or gigabytes). Items are often written and read individually. High durability and low cost are essential. Cold data tends to be high-volume and low-velocity.

**Frozen** *Frozen data* needs to be preserved for business continuity or for archival or regulatory reasons, but it is not being worked on actively. While new data is regularly added to this data store, existing data is never updated. Reads are extremely infrequent (known as “write once, read never”) and can tolerate very high latency. Frozen data tends to be extremely high-volume and extremely low-velocity.

The same data can start as hot and gradually cool down. As it does, the tolerance of read latency increases, as does the total size of the dataset. Later in this chapter, we explore individual AWS services and discuss which services are optimized for the dimensions that we have discussed so far.

## Data Value

Although we would like to extract useful information from all of the data we collect, not all data is equally important to us. Some data has to be preserved at all costs, and other data can be easily regenerated as needed or even lost without significant impact on the business. Depending on the value of data, we are more or less willing to invest in additional durability.



To optimize cost and/or performance further, segment data within each workload by value and temperature, and consider different data storage options for different segments.

**Transient data** *Transient data* is often short-lived. The loss of some subset of transient data does not have significant impact on the system as a whole. Examples include clickstream or Twitter data. We often do not need high durability of this data, because we expect it to be quickly consumed and transformed further, yielding higher-value data. If we lose a tweet or a few clicks, this is unlikely to affect our sentiment analysis or user behavior analysis.

Not all streaming data is transient, however. For example, for an *intrusion detection system* (IDS), every record representing network communication can be valuable because every log record can be valuable for a monitoring/alarming system.

**Reproducible data** *Reproducible data* contains a copy of useful information that is often created to improve performance or simplify consumption, such as adding more structure or altering a structure to match consumption patterns. Although the loss of some or all of this data may affect a system's performance or availability, this will not result in data loss, because the data can be reproduced from other data sources.

Examples include data warehouse data, read replicas of OLTP (online transaction processing) systems, and many types of caches. For this data, we may invest a bit in durability to reduce the impact on system's performance and availability, but only to a point.

**Authoritative data** *Authoritative data* is the source of truth. Losing this data will have significant business impact because it will be difficult, or even impossible, to restore or replace it. For this data, we are willing to invest in additional durability. The greater the value of this data, the more durability we will want.

**Critical/Regulated data** *Critical or regulated data* is data that a business must retain at almost any cost. This data tends to be stored for long periods of time and needs to be protected from accidental and malicious changes—not just data loss or corruption. Therefore, in addition to durability, cost and security are equally important factors.

## One Tool Does Not Fit All

Despite the many applications of a hammer, it cannot replace a screwdriver or a pair of pliers. Likewise, there is no one-size-fits-all solution for data storage. Analyze your data and understand the dimensions that we have discussed. Once you have done that, then you can move on to reviewing the different storage options available on AWS to find the right tool to store and access your files.



For the exam, know the availability, level of durability, and cost factors for each storage option and how they compare.

## Block, Object, and File Storage

There are three types of cloud storage: object, file, and block. Each offers its own unique advantages.

## Block Storage

Some enterprise applications, like databases or *enterprise resource planning systems* (ERP systems), can require dedicated, low-latency storage for each host. This is analogous to *direct-attached storage* (DAS) or a *storage area network* (SAN). Block-based cloud storage solutions like Amazon EBS are provisioned with each Amazon Elastic Compute Cloud (Amazon EC2) instance and offer the ultra-low latency required for high-performance workloads.

## Object Storage

Applications developed on the cloud often take advantage of object storage's vast scalability and metadata characteristics. Object storage solutions like Amazon S3 are ideal for building modern applications from scratch that require scale and flexibility and can also be used to import existing data stores for analytics, backup, or archive.

Cloud object storage makes it possible to store virtually limitless amounts of data in its native format.

## File Storage

Many applications need to access shared files and require a file system. This type of storage is often supported with a *network-attached storage* (NAS) server. File storage solutions like Amazon EFS are ideal for use cases such as large content repositories, development environments, media stores, or user home directories.

## AWS Shared Responsibility Model and Storage

The AWS shared responsibility model is important to understand as it relates to cloud storage. AWS is responsible for securing the storage services. As a developer and customer, you are responsible for securing access to and using encryption on the artifacts you create or objects you store.

AWS makes this model simpler for you by allowing you to inherit certain compliance factors and controls, but you must still ensure that you are securing your data and files on the cloud. It is a best practice always to use the principle of least privilege as part of your responsibility for using AWS Cloud storage. For example, ensure that only those who need access to the file have access and ensure that read and write access are separated and controlled.

## Confidentiality, Integrity, Availability Model

The *confidentiality, integrity, availability model* (CIA model) forms the fundamentals of information security, and you can apply the principles of the CIA model to AWS storage.

*Confidentiality* can be equated to the privacy level of your data. It refers to levels of encryption or access policies for your storage or individual files. With this principle, you will limit access to prevent accidental information disclosure by restricting permissions and enabling encryption.

*Integrity* refers to whether your data is trustworthy and accurate. For example, can you trust that the file you generated has not been changed when it is audited later?



Restrict permission of who can modify data and enable backup and versioning.

*Availability* refers to the availability of a service on AWS for storage, where an authorized party can gain reliable access to the resource.



Restrict permission of who can delete data, enable multi-factor authentication (MFA) for Amazon S3 delete operation, and enable backup and versioning.

Figure 3.3 shows the CIA model.

**FIGURE 3.3** The CIA model



AWS storage services provide many features for maintaining the desired level of confidentiality, integrity, and availability. Each of these features is discussed under its corresponding storage-option section in this chapter.

## AWS Block Storage Services

Let's begin with the storage to which you are most likely already accustomed as a developer; that is, block storage.

## Amazon Elastic Block Store

Amazon EBS presents your data to your Amazon EC2 instance as a disk volume, providing the lowest-latency access to your data from single Amazon EC2 instances.

Amazon EBS provides durable and persistent block storage volumes for use with Amazon EC2 instances. Each Amazon EBS volume is automatically replicated within its Availability Zone to protect your information from component failure, offering high availability and durability. Amazon EBS volumes offer the consistent and low-latency performance needed to run your workloads. With Amazon EBS, you can scale your usage up or down within minutes, while paying only for what you provision.

Typical use cases for Amazon EBS include the following:

- Boot volumes on Amazon EC2 instances
- Relational and NoSQL databases
- Stream and log processing applications
- Data warehousing applications.
- Big data analytics engines (like the Hadoop/HDFS (Hadoop Distributed File System) ecosystem and Amazon EMR clusters)

Amazon EBS is designed to achieve the following:

- Availability of 99.999 percent
- Durability of replication within a single availability zone
- *Annual failure rate (AFR)* of between 0.1 and 0.2 percent

Amazon EBS volumes are 20 times more reliable than typical commodity disk drives, which fail with an AFR of around 4 percent.

## Amazon EBS Volumes

*Amazon EBS volumes* persist independently from the running life of an Amazon EC2 instance. After a volume is attached to an instance, use it like any other physical hard drive.

Amazon EBS volumes are flexible. For current-generation volumes attached to current-generation instance types, you can dynamically increase size, modify provisioned input/output operations per second (IOPS) capacity, and change the volume type on live production volumes without service interruptions.

Amazon EBS provides the following volume types, which differ in performance characteristics and price so that you can tailor your storage performance and cost to the needs of your applications.

**SSD-backed volumes** *Solid-state drive (SSD)-backed volumes* are optimized for transactional workloads involving frequent read/write operations with small I/O size, where the dominant performance attribute is IOPS.

**HDD-backed volumes** *Hard disk drive (HDD)-backed volumes* are optimized for large streaming workloads where throughput (measured in MiB/s) is a better performance measure than IOPS.

## SSD vs. HDD Comparison

Table 3.1 shows a comparison of Amazon EBS HDD-backed and SSD-backed volumes.

**TABLE 3.1** Volume Comparison

|                              | SSD             | HDD              |                      |           |
|------------------------------|-----------------|------------------|----------------------|-----------|
|                              | General Purpose | Provisioned IOPS | Throughput-Optimized | Cold      |
| <b>Max volume size</b>       | 16 TiB          |                  |                      |           |
| <b>Max IOPS/volume</b>       | 10,000          | 32,000           | 500                  | 250       |
| <b>Max throughput/volume</b> | 160 MiB/s       | 500 MiB/s        |                      | 250 MiB/s |

Table 3.2 shows the most common use cases for the different types of Amazon EBS volumes.

**TABLE 3.2** EBS Volume Use Cases

| SSD                                                                                                                                                                                                                                           | HDD                                                                                                                        |                                                                                                                                                                                                                                          |                                                                                                                                                                                                                                            |  |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
| General Purpose                                                                                                                                                                                                                               | Provisioned IOPS                                                                                                           | Throughput-Optimized                                                                                                                                                                                                                     | Cold                                                                                                                                                                                                                                       |  |
| <ul style="list-style-type: none"> <li>■ Recommended for most workloads</li> <li>■ System boot volumes</li> <li>■ Virtual desktops</li> <li>■ Low-latency interactive</li> <li>■ Apps</li> <li>■ Development and test environments</li> </ul> | <ul style="list-style-type: none"> <li>■ I/O-intensive workloads</li> <li>■ Relational DBs</li> <li>■ NoSQL DBs</li> </ul> | <ul style="list-style-type: none"> <li>■ Streaming workloads requiring consistent, fast throughput at a low price</li> <li>■ Big data</li> <li>■ Data warehouses</li> <li>■ Log processing</li> <li>■ Cannot be a boot volume</li> </ul> | <ul style="list-style-type: none"> <li>■ Throughput-oriented storage for large volumes of data that is infrequently accessed</li> <li>■ Scenarios where the lowest storage cost is important</li> <li>■ Cannot be a boot volume</li> </ul> |  |

## Elastic Volumes

*Elastic Volumes* is a feature of Amazon EBS that allows you to increase capacity dynamically, tune performance, and change the type of volume live. This can be done with no downtime or performance impact and with no changes to your application. Create a

volume with the capacity and performance needed when you are ready to deploy your application, knowing that you have the ability to modify your volume configuration in the future and saving hours of planning cycles and preventing overprovisioning.

## Amazon EBS Snapshots

You can protect your data by creating point-in-time snapshots of Amazon EBS volumes, which are backed up to Amazon S3 for long-term durability. The volume does not need to be attached to a running instance to take a *snapshot*.

As you continue to write data to a volume, periodically create a snapshot of the volume to use as a baseline for new volumes. These snapshots can be used to create multiple new Amazon EBS volumes or move volumes across Availability Zones.

When you create a new volume from a snapshot, it is an exact copy of the original volume at the time the snapshot was taken.

If you are taking snapshots at regular intervals, such as once per day, you may be concerned about the cost of the storage. Snapshots are incremental backups, meaning that only the blocks on the volume that have changed after your most recent snapshot are saved, making this a cost-effective way to back up your block data. For example, if you have a volume with 100 GiB of data, but only 5 GiB of data have changed since your last snapshot, only the 5 GiB of modified data is written to Amazon S3.

If you need to delete a snapshot, how do you know which snapshot to delete? Amazon EBS handles this for you. Even though snapshots are saved incrementally, the snapshot deletion process is designed so that you need to retain only the most recent snapshot to restore the volume. Amazon EBS will determine which dependent snapshots can be deleted to ensure that all other snapshots continue working.

## Amazon EBS Optimization

Recall that Amazon EBS volumes are network-attached and not directly attached to the host like instance stores. On instances without support for Amazon EBS-optimized throughput, network traffic can contend with traffic between your instance and your Amazon EBS volumes. On *Amazon EBS-optimized instances*, the two types of traffic are kept separate. Some instance configurations incur an extra cost for using Amazon EBS-optimized, while others are always Amazon EBS-optimized at no extra cost.

## Amazon EBS Encryption

For simplified *data* encryption, create encrypted Amazon EBS volumes with the Amazon EBS encryption feature. All Amazon EBS volume types support encryption, and you can use encrypted Amazon EBS volumes to meet a wide range of data-at-rest encryption requirements for regulated/audited data and applications.

Amazon EBS encryption uses 256-bit *Advanced Encryption Standard* (AES-256) algorithms and an Amazon-managed key infrastructure called *AWS Key Management Service* (AWS KMS). The encryption occurs on the server that hosts the Amazon EC2 instance, providing encryption of data in transit from the Amazon EC2 instance to Amazon EBS storage.

You can encrypt using an AWS KMS-generated key, or you can choose to select a *customer master key* (CMK) that you create separately using AWS KMS.

You can also encrypt your files prior to placing them on the volume. Snapshots of encrypted Amazon EBS volumes are automatically encrypted. Amazon EBS volumes that are restored from encrypted snapshots are also automatically encrypted.

## Amazon EBS Performance

To achieve optimal performance from your Amazon EBS volumes in a variety of scenarios, use the following best practices:

**Use Amazon EBS-optimized instances** The dedicated network throughput that you get when you request Amazon EBS-optimized support will make volume performance more predictable and consistent, and your Amazon EBS volume network traffic will not have to contend with your other instance traffic because they are kept separate.

**Understand how performance is calculated** When you measure the performance of your Amazon EBS volumes, it is important to understand the units of measure involved and how performance is calculated.

**Understand your workload** There is a relationship between the maximum performance of your Amazon EBS volumes, the size and number of I/O operations, and the time it takes for each action to complete. Each of these factors affects the others, and different applications are more sensitive to one factor or another.

On a given volume configuration, certain I/O characteristics drive the performance behavior for your Amazon EBS volumes. *SSD-backed volumes*, *General-Purpose SSD*, and *Provisioned IOPS SSD* deliver consistent performance whether an I/O operation is random or sequential. *HDD-backed volumes*, *Throughput-Optimized HDD*, and *Cold HDD* deliver optimal performance only when I/O operations are large and sequential.

To understand how SSD and HDD volumes will perform in your application, it is important to understand the connection between demand on the volume, the quantity of IOPS available to it, the time it takes for an I/O operation to complete, and the volume's throughput limits.

**Be aware of the performance penalty when initializing volumes from snapshots** New Amazon EBS volumes receive their maximum performance the moment that they are available and do not require initialization (formerly known as *pre-warming*).

Storage blocks on volumes that were restored from snapshots, however, must be initialized (pulled down from Amazon S3 and written to the volume) before you can access the block. This preliminary action takes time and can cause a significant increase in the latency of an I/O operation the first time each block is accessed. Performance is restored after the data is accessed once.

For most applications, amortizing this cost over the lifetime of the volume is acceptable. For some applications, however, this performance hit is not acceptable. If that is the case, avoid a performance hit by accessing each block prior to putting the volume into production. This process is called *initialization*.

**Factors that can degrade HDD performance** When you create a snapshot of a Throughput-Optimized HDD or Cold HDD volume, performance may drop as far as the volume's baseline value while the snapshot is in progress. This behavior is specific only to these volume types.

Other factors that can limit performance include the following:

- Driving more throughput than the instance can support
- The performance penalty encountered when initializing volumes restored from a snapshot
- Excessive amounts of small, random I/O on the volume

**Increase read-ahead for high-throughput, read-heavy workloads** Some workloads are read-heavy and access the block device through the operating system *page cache* (for example, from a file system). In this case, to achieve the maximum throughput, we recommend that you configure the read-ahead setting to 1 MiB. This is a per-block-device setting that should be applied only to your HDD volumes.

**Use RAID 0 to maximize utilization of instance resources** Some instance types can drive more I/O throughput than what you can provision for a single Amazon EBS volume. You can join multiple volumes of certain instance types together in a *RAID 0* configuration to use the available bandwidth for these instances.

**Track performance with Amazon CloudWatch** *Amazon CloudWatch*, a monitoring and management service, provides performance metrics and status checks for your Amazon EBS volumes.

## Amazon EBS Troubleshooting

If you are using an Amazon EBS volume as a boot volume, your instance is no longer accessible, and you cannot use *SSH* or *Remote Desktop Protocol* (RDP) to access that boot volume. There are some steps that you can take, however, to access the volume.

If you have an Amazon EC2 instance based on an *Amazon Machine Image* (AMI), you may just choose to terminate the instance and create a new one.

If you do need access to that Amazon EBS boot volume, perform the following steps to make it accessible:

1. Create a new Amazon EC2 instance with its own boot volume (a micro instance is great for this purpose).
2. Detach the root Amazon EBS volume from the troubled instance.
3. Attach the root Amazon EBS volume from the troubled instance to your new Amazon EC2 instance as a secondary volume.
4. Connect to the new Amazon EC2 instance, and access the files on the secondary volume.

## Instance Store

Amazon EC2 *instance store* is another type of block storage available to your Amazon EC2 instances. It provides *temporary* block-level storage, and the storage is located on disks

that are physically attached to the host computer (unlike Amazon EBS volumes, which are network-attached).



If your data does not need to be resilient to *reboots, restarts, or auto recovery*, then your data may be a candidate for using instance store, but you should exercise caution.

## Instance Store Volumes

*Instance store* should not be used for persistent storage needs. It is a type of ephemeral (short-lived) storage that does not persist if the instance fails or is terminated.

Because instance store is on the host of your Amazon EC2 instance, it will provide the lowest-latency storage to your instance other than RAM. Instance store volumes can be used when incurring large amounts of I/O for your application at the lowest possible latency. You need to ensure that you have another source of truth of your data, however, and that the only copy is not placed on instance store. For data that needs to be durable, we recommend using Amazon EBS volumes instead.

Not all instance types come with available instance store volume(s), and the size and type of volumes vary by instance type. When you launch an instance, the instance store is available at no additional cost, depending on the particular instance type. However, you must enable these volumes when you launch an Amazon EC2 instance, as you cannot add instance store volumes to an Amazon EC2 instance once it has been launched.

After you launch an instance, the instance store volumes are available to the instance, but you cannot access them until they are mounted. Refer to the AWS documentation for Amazon EBS to learn more about mounting these volumes on different operating systems.

Many customers use a combination of instance store and Amazon EBS volumes with their instances. For example, you may choose to place your scratch data, *tempdb*, or other temporary files on instance store while your root volume is on Amazon EBS.



Do not use instance store for any production data.

## Instance Store–Backed Amazon EC2 Instances

With Amazon EC2, you can use both instance store–backed storage volumes and Amazon EBS–backed storage volumes with your instances, meaning you can have your instance boot off instance store; however, you would want this configured so that you are using an AMI and that new instances will be created if one fails. This is not recommended for your primary instances where it would cause an issue for users if the instance fails. This configuration can save money on storage costs instead of using Amazon EBS as your boot volume in cases where your system is configured to be resilient to instances re-launching. It is critical to understand your application and infrastructure needs before choosing to use instance store-backed Amazon EC2 instances, so choose carefully.

*Instance store-backed Amazon EC2 instances* cannot be stopped and cannot take advantage of the auto recovery feature for Amazon EC2 instances.

Some AWS customers build instances on the fly that are completely resilient to reboot, relaunch, or failure and use instance store as their root volumes. This requires important due diligence regarding your application and infrastructure to ensure that this type of scenario would be right for you.

# AWS Object Storage Services

Now we are going to dive into object storage. An *object* is a piece of data like a document, image, or video that is stored with some *metadata* in a flat structure. Object storage provides that data to applications via *application programming interfaces* (APIs) over the internet.

## Amazon Simple Storage Service

Building a web application, which delivers content to users by retrieving data via making API calls over the internet, is not a difficult task with Amazon S3. Amazon Simple Storage Service (Amazon S3) is storage for the internet. It is a simple storage service that offers software developers a highly scalable, reliable, and low-latency data storage infrastructure at low cost. AWS has seen enormous growth with Amazon S3, and AWS currently has customers who store terabytes and exabytes of data.



Amazon S3 is featured in many AWS certifications because it is a core enabling service for many applications and use cases.

To begin developing with Amazon S3, it is important to understand a few basic concepts.

### Buckets

A *bucket* is a container for objects stored in Amazon S3. Every object is contained in a bucket. You can think of a bucket in traditional terminology similar to a drive or volume.

### Limitations

The following are limitations of which you should be aware when using Amazon S3 buckets:

- Do not use buckets as folders, because there is a maximum limit of 100 buckets per account.
- You cannot create a bucket within another bucket.
- A bucket is owned by the AWS account that created it, and bucket ownership is not transferable.
- A bucket must be empty before you can delete it.

- After a bucket is deleted, that name becomes available to reuse, but the name might not be available for you to reuse for various reasons, such as someone else taking the name after you release it when deleting the bucket. If you expect to use same bucket name, do not delete the bucket.



You can only create up to 100 buckets per account. Do not use buckets as folders or design your application in a way that could result in more than 100 buckets as your application or data grows.

### Universal Namespace

A bucket name must be unique across all existing bucket names in Amazon S3 across all of AWS—not just within your account or within your chosen AWS Region. You must comply with *Domain Name System* (DNS) naming conventions when choosing a bucket name.

The rules for DNS-compliant bucket names are as follows:

- Bucket names must be at least 3 and no more than 63 characters long.
- A bucket name must consist of a series of one or more labels, with adjacent labels separated by a single period (.).
- A bucket name must contain lowercase letters, numbers, and hyphens.
- Each label must start and end with a lowercase letter or number.
- Bucket names must not be formatted like *IP addresses* (for example, 192.168.5.4).
- AWS recommends that you do not use periods (.) in bucket names. When using virtual hosted-style buckets with *Secure Sockets Layer* (SSL), the SSL wildcard certificate only matches buckets that do not contain periods. To work around this, use HTTP or write your own certificate verification logic.



Amazon S3 bucket names must be universally unique.

Table 3.3 shows examples of invalid bucket names.

**TABLE 3.3** Invalid Bucket Names

| Bucket Name       | Reason                                          |
|-------------------|-------------------------------------------------|
| .myawsbucket      | The bucket name cannot start with a period (.). |
| myawsbucket.      | The bucket name cannot end with a period (.).   |
| my..examplebucket | There can be only one period between labels.    |

The following code snippet is an example of creating a bucket using Java:

```
private static String bucketName = "**** bucket name ****";
public static void main(String[] args) throws IOException {
 AmazonS3 s3client = new AmazonS3Client(new ProfileCredentialsProvider());
 s3client.setRegion(Region.getRegion(Regions.US_WEST_1));
 if(!(s3client.doesBucketExist(bucketName))){
 // Note that CreateBucketRequest does not specify region. So bucket is
 // created in the region specified in the client.
 s3client.createBucket(new CreateBucketRequest(bucketName));
 }

 // Get location.
 String bucketLocation = s3client.getBucketLocation(new GetBucketLocationRequest
 (bucketName));
 System.out.println("bucket location = " + bucketLocation);
```

## Versioning

*Versioning* is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket, including recovering deleted objects. With versioning, you can easily recover from both unintended user actions and application failures.

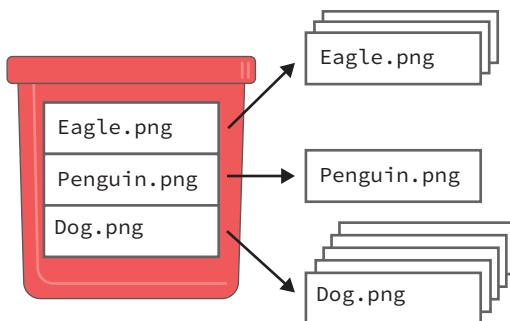
There are several reasons that developers will turn on versioning of files in Amazon S3, including the following:

- Protecting from accidental deletion
- Recovering an earlier version
- Retrieving deleted objects

Versioning is turned off by default. When you turn on versioning, Amazon S3 will create new versions of your object every time you overwrite a particular object key. Every time you update an object with the same key, Amazon S3 will maintain a new version of it.

In Figure 3.4, you can see that we have uploaded the same image multiple times, and all of the previous versions of those files have been maintained.

**FIGURE 3.4** Amazon S3 versioning



As those additional writes apply to a bucket, you can retrieve any of the particular objects that you need using GET on the object key name and the particular version. Amazon S3 versioning *tracks the changes over time*.

Amazon S3 versioning also protects against unintended deletes. If you issue a delete command against an object in a versioned bucket, AWS places a delete marker on top of that object, which means that if you perform a GET on it, you will receive an error as if the object does not exist. However, an administrator, or anyone else with the necessary permissions, could remove the delete marker and access the data.

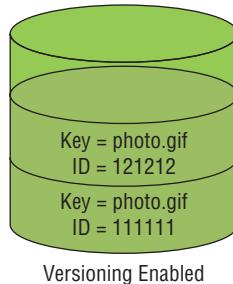
When a delete request is issued against a versioned bucket on a particular object, Amazon S3 still retains the data, but it removes access for users to retrieve that data.

*Versioning-enabled buckets* let you recover objects from accidental deletion or overwrite. Your bucket's versioning configuration can also be MFA Delete–enabled for an additional layer of security. MFA Delete is discussed later in this chapter.

If you overwrite an object, it results in a new object version in the bucket. You can always restore from any previous versions.

In one bucket, for example, you can have two objects with the same key, but different version IDs, such as photo.gif (version 111111) and photo.gif (version 121212). This is illustrated in Figure 3.5.

**FIGURE 3.5** Amazon S3 object version IDs



Later in this chapter, we will cover *lifecycle policies*. You can use versioning in combination with lifecycle policies to implement them if the object is the current or previous version. If you are concerned about building up many versions and using space for a particular object, configure a lifecycle policy that will delete the old version of the object after a certain period of time.



It is easy to set up a lifecycle policy to control the amount of data that's being retained when you use versioning on a bucket.

If you need to discontinue versioning on a bucket, copy all of your objects to a new bucket that has versioning disabled and use that bucket going forward.



Once you enable versioning on a bucket, it can never return to an unversioned state. You can, however, suspend versioning on that bucket.

It is important to be aware of the cost implications of the bucket that is versioning-enabled. When calculating cost for your bucket, you must calculate as though every version is a completely separate object that takes up the same space as the object itself. As you can probably guess, this option might be cost prohibitive for things like large media files or performing many updates on objects.

## Region

Amazon S3 creates buckets in a region that you specify. You can choose any AWS Region that is geographically close to you to optimize latency, minimize costs, or address regulatory requirements.



Objects belonging to a bucket that you create in a specific AWS Region never leave that region unless you explicitly transfer them to another region.

## Operations on Buckets

There are a number of different operations (API calls) that you can perform on Amazon S3 buckets. We will summarize a few of the most basic operations in this section. For more comprehensive information on all of the different operations that you can perform, refer to the Amazon S3 API Reference document available in the AWS Documentation repository. In this section, we show you how to create a bucket, list buckets, and delete a bucket.

### CREATE A BUCKET

This sample Python code shows how to create a bucket:

```
import boto3

s3 = boto3.client('s3')
s3.create_bucket(Bucket='my-bucket')
```

### LIST BUCKETS

This sample Python code demonstrates getting a list of all of the bucket names available:

```
import boto3

Create an S3 client
s3 = boto3.client('s3')

Call S3 to list current buckets
response = s3.list_buckets()

Get a list of all bucket names from the response
buckets = [bucket['Name'] for bucket in response['Buckets']]

Print out the bucket list
print("Bucket List: %s" % buckets)
```

**DELETE A BUCKET**

The following sample Java code shows you how to delete a bucket. Buckets must be empty before you can delete them, unless you use a force parameter.

```
import java.io.IOException;

import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.DeleteObjectRequest;

public class DeleteObjectNonVersionedBucket {

 public static void main(String[] args) throws IOException {
 String clientRegion = "*** Client region ***";
 String bucketName = "*** Bucket name ***";
 String keyName = "*** Key name ***";

 try {
 AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
 .withCredentials(new ProfileCredentialsProvider())
 .withRegion(clientRegion)
 .build();

 s3Client.deleteObject(new DeleteObjectRequest(bucketName, keyName));
 }
 catch(AmazonServiceException e) {
 // The call was transmitted successfully, but Amazon S3 couldn't process
 // it, so it returned an error response.
 e.printStackTrace();
 }
 catch(SdkClientException e) {
 // Amazon S3 couldn't be contacted for a response, or the client
 // couldn't parse the response from Amazon S3.
 e.printStackTrace();
 }
 }
}
```

## AWS COMMAND LINE INTERFACE

The following is a sample AWS Command Line Interface (AWS CLI) command that will delete a bucket and will use the `--force` parameter to remove a nonempty bucket. This command deletes all objects first and then deletes the bucket.

```
$ aws s3 rb s3://bucket-name --force
```

## Objects

You can store an unlimited number of objects within Amazon S3, but an object can only be between 1 byte to 5 TB in size. If you have objects larger than 5 TB, use a file splitter and upload the file in chunks to Amazon S3. Then reassemble them if you download the file parts for later use.

The largest object that can be uploaded in a single PUT is 5 GB. For objects larger than 100 MB, you should consider using multipart upload (discussed later in this chapter). For any objects larger than 5 GB, you must use multipart upload.

### Object Facets

An object consists of the following facets:

**Key** The *key* is the name that you assign to an object, which may include a simulated folder structure. Each key must be unique within a bucket (unless the bucket has versioning turned on).

Amazon S3 URLs can be thought of as a basic data map between “bucket + key + version” and the web service endpoint. For example, in the URL <http://doc.s3.amazonaws.com/2006-03-01/AmazonS3.wsdl>, doc is the name of the bucket and 2006-03-01/AmazonS3.wsdl is the key.

**Version ID** Within a bucket, a key and *version ID* uniquely identify an object. If versioning is turned off, you have only a single version. If versioning is turned on, you may have multiple versions of a stored object.

**Value** The *value* is the actual content that you are storing. An object value can be any sequence of bytes, and objects can range in size from 1 byte up to 5 TB.

**Metadata** *Metadata* is a set of name-value pairs with which you can store information regarding the object. You can assign metadata, referred to as *user-defined metadata*, to your objects in Amazon S3. Amazon S3 also assigns system metadata to these objects, which it uses for managing objects.

**Subresources** Amazon S3 uses the *subresource* mechanism to store additional object-specific information. Because subresources are subordinates to objects, they are always associated with some other entity such as an object or a bucket. The subresources associated with Amazon S3 objects can include the following:

**Access control list (ACL)** A list of grants identifying the grantees and the permissions granted.

**Torrent** Returns the torrent file associated with the specific object.

**Access Control Information** You can control access to the objects you store in Amazon S3. Amazon S3 supports both *resource-based access control*, such as an ACL and *bucket policies*, and *user-based access control*.

## Object Tagging

*Object tagging* enables you to categorize storage. Each tag is a key-value pair. Consider the following tagging examples.

Suppose an object contains *protected health information* (PHI) data. You can tag the object using the following key-value pair:

PHI=True

or

Classification=PHI



While it is acceptable to use tags to label objects containing confidential data (such as personally identifiable information (PII) or PHI), the tags themselves should not contain any confidential information.

Suppose that you store project files in your Amazon S3 bucket. You can tag these objects with a key called Project and a value, as shown here:

Project=Blue

You can add multiple tags to a single object, such as the following:

Project=SalesForecast2018

Classification=confidential

You can tag new objects when you upload them, or you can add them to existing objects. Note the following limitations when working with tagging:

- You can associate 10 tags with an object, and each tag associated with an object must have unique tag keys.
- A tag key can be up to 128 Unicode characters in length, and tag values can be up to 256 Unicode characters in length.

Keys and values are case sensitive.

Developers commonly categorize their files in a folder-like structure in the key name (remember, Amazon S3 has a flat file structure), such as the following:

```
photos/photo1.jpg
project/projectx/document.pdf
project/projecty/document2.pdf
```

This allows you to have only one-dimensional categorization, meaning that everything under a prefix is one category.

With tagging, you now have another dimension. If your *photo1* is in *project x* category, tag the object accordingly. In addition to data classification, tagging offers the following benefits:

- Object tags enable fine-grained access control of permissions. For example, you could grant an AWS Identity and Access Management (IAM) user permissions to read-only objects with specific tags.
- Object tags enable fine-grained object lifecycle management in which you can specify a tag-based filter, in addition to key name prefix, in a lifecycle rule.
- When using Amazon S3 analytics, you can configure filters to group objects together for analysis by object tags, by key name prefix, or by both prefix and tags.
- You can also customize Amazon CloudWatch metrics to display information by specific tag filters. The following sections provide details.

## Cross-Origin Resource Sharing

*Cross-Origin Resource Sharing* (CORS) defines a way for client web applications that are loaded in one domain to interact with resources in a different domain. With CORS support in Amazon S3, you can build client-side web applications with Amazon S3 and selectively allow cross-origin access to your Amazon S3 resources while avoiding the need to use a proxy.

### Cross-Origin Request Scenario

Suppose that you are hosting a website in an Amazon S3 bucket named *website* on Amazon S3. Your users load the website endpoint: <http://website.s3-website-us-east-1.amazonaws.com>.

Your website will use JavaScript on the web pages that are stored in this bucket to be able to make authenticated GET and PUT requests against the same bucket by using the Amazon S3 API endpoint for the bucket: *website.s3.amazonaws.com*.

A browser would normally block JavaScript from allowing those requests, but with CORS, you can configure your bucket to enable cross-origin requests explicitly from *website.s3-website-us-east-1.amazonaws.com*.



Suppose that you host a web font from your Amazon S3 bucket. Browsers require a CORS check (also referred as a *preflight check*) for loading web fonts, so you would configure the bucket that is hosting the web font to allow any origin to make these requests.



There are no coding exercises as part of the exam, but these case studies can help you visualize how to use Amazon S3 and CORS.

## Operations on Objects

There are a number of different operations (API calls) that you can perform on Amazon S3 buckets. We will summarize a few of the most basic operations in this section. For more comprehensive information on all of the different operations that you can perform, refer to the Amazon S3 API Reference document available in the AWS Documentation repository.

### WRITE AN OBJECT

This sample Java code shows how to add an object to a bucket:

```
import boto3

Create an S3 client
s3 = boto3.client('s3')

filename = 'file.txt'
bucket_name = 'my-bucket'

Uploads the given file using a managed uploader, which will split up large
files automatically and upload parts in parallel.
s3.upload_file(filename, bucket_name, filename)
```

### READING OBJECTS

The following Java code example demonstrates getting a stream on the object data of a particular object and processing the response:

```
AmazonS3 s3Client = new AmazonS3Client(new ProfileCredentialsProvider());
S3Object object = s3Client.getObject(
 new GetObjectRequest(bucketName, key));
InputStream objectData = object.getObjectContent();
// Process the objectData stream.
objectData.close();
```

### DELETING OBJECTS

You can delete one or more objects directly from Amazon S3. You have the following options when deleting an object:

**Delete a Single Object** Amazon S3 provides the DELETE API to delete one object in a single HTTP request.

**Delete Multiple Objects** Amazon S3 also provides the Multi-Object Delete API to delete up to 1,000 objects in a single HTTP request.

The following Java sample demonstrates deleting an object by providing the bucket name and key name:

```
AmazonS3 s3client = new AmazonS3Client(new ProfileCredentialsProvider());
s3client.deleteObject(new DeleteObjectRequest(bucketName, keyName));
```

This next Java sample demonstrates deleting a versioned object by providing a bucket name, object key, and a version ID:

```
AmazonS3 s3client = new AmazonS3Client(new ProfileCredentialsProvider());
s3client.deleteObject(new DeleteVersionRequest(bucketName, keyName,
versionId));
```

**List Keys** The following Java code example lists object keys in a bucket:

```
private static String bucketName = "****bucket name****";
AmazonS3 s3client = new AmazonS3Client(new ProfileCredentialsProvider());
System.out.println("Listing objects");
final ListObjectsV2Request req = new ListObjectsV2Request().
withBucketName(bucketName).withMaxKeys(2);
ListObjectsV2Result result;
do {
 result = s3client.listObjectsV2(req);

 for (S3ObjectSummary objectSummary :
 result.getObjectSummaries()) {
 System.out.println(" - " + objectSummary.getKey() + " " +
 "(size = " + objectSummary.getSize() +
 ")");
 }
}
System.out.println("Next Continuation Token : " +
result.getNextContinuationToken());

req.setContinuationToken(result.getNextContinuationToken());
} while(result.isTruncated() == true);
```

## Storage Classes

There are several different storage classes from which to choose when using Amazon S3. Your choice will depend on your level of need for durability, availability, and performance for your application.

### Amazon S3 Standard

Amazon S3 Standard offers high-durability, high-availability, and performance-object storage for frequently accessed data. Because it delivers low latency and high

throughput, Amazon S3 Standard is ideal for a wide variety of use cases, including the following:

- Cloud applications
- Dynamic websites
- Content distribution
- Mobile and gaming applications
- Big data analytics

Amazon S3 Standard is designed to achieve durability of 99.99999999 percent of objects (designed to sustain the loss of data in two facilities) and availability of 99.99 percent over a given year (which is backed by the Amazon S3 Service Level Agreement).

Essentially, the data in Amazon S3 is spread out over multiple facilities within a region. You can lose access to two facilities and still have access to your files.

### **Reduced Redundancy Storage**

*Reduced Redundancy Storage* (RRS) (or Reduced\_Redundancy) is an Amazon S3 storage option that enables customers to store noncritical, reproducible data at lower levels of redundancy than Amazon S3 Standard storage. It provides a highly available solution for distributing or sharing content that is durably stored elsewhere or for objects that can easily be regenerated, such as thumbnails or transcoded media.

The RRS option stores objects on multiple devices across multiple facilities, providing 400 times the durability of a typical disk drive, but it does not replicate objects as many times as Amazon S3 Standard storage.

RRS is designed to achieve availability of 99.99 percent (same as Amazon S3 Standard) and durability of 99.99 percent (designed to sustain the loss of data in a single facility).

### **Amazon S3 Standard-Infrequent Access**

*Amazon S3 Standard-Infrequent Access* (Standard\_IA) is an Amazon S3 storage class for data that is accessed less frequently but requires rapid access when needed. It offers the same high durability, throughput, and low latency of Amazon S3 Standard, but it has a lower per-gigabyte storage price and per-gigabyte retrieval fee.

The ideal use cases for using Standard\_IA include the following:

- Long-term storage
- Backups
- Data stores for disaster recovery

Standard\_IA is set at the object level and can exist in the same bucket as Amazon S3 Standard, allowing you to use lifecycle policies to transition objects automatically between storage classes without any application changes.

Standard\_IA is designed to achieve availability of 99.9 percent (but low retrieval time) and durability of 99.99999999 percent of objects over a given year (same as Amazon S3 Standard).

## Amazon S3 One Zone-Infrequent Access

Amazon S3 One Zone-Infrequent Access (OneZone\_IA) is similar to Amazon S3 Standard-IA. The difference is that the data is stored only in a single Availability Zone instead of a minimum of three Availability Zones. Because of this, storing data in OneZone\_IA costs 20 percent less than storing it in Standard\_IA. Because of this approach, however, any data stored in this storage class will be permanently lost in the event of an Availability Zone destruction.

## Amazon Simple Storage Service Glacier

*Amazon Simple Storage Service Glacier (Amazon S3 Glacier)* is a secure, durable, and extremely low-cost storage service for data archiving that offers the same high durability as Amazon S3. Unlike Amazon S3 Standard's immediate retrieval times, Amazon S3 Glacier's retrieval times run from a few minutes to several hours.

To keep costs low, Amazon S3 Glacier provides three archive access speeds, ranging from minutes to hours. This allows you to choose an option that will meet your *recovery time objective* (RTO) for backups in your disaster recovery plan.

Amazon S3 Glacier can also be used to secure archives that need to be kept due to a compliance policy. For example, you may need to keep certain records for seven years before deletion and only need access during an audit. Amazon S3 Glacier allows redundancy in your files when audits do occur, but at an extremely low cost in exchange for slower access.

### VAULTS

Amazon S3 Glacier uses *vaults* as containers to store archives. You can view a list of your vaults in the *AWS Management Console* and use the *AWS software development kits* (SDKs) to perform a variety of vault operations, such as the following:

- Create vault
- Delete vault
- Lock vault
- List vault metadata
- Retrieve vault inventory
- Tag vaults for filtering
- Configure vault notifications

You can also set access policies for each vault to grant or deny specific activities to users. You can have up to 1,000 vaults per AWS account.

Amazon S3 Glacier provides a management console to create and delete vaults. All other interactions with Amazon S3 Glacier, however, require that you use the AWS CLI or write code.

### VAULT LOCK

Amazon S3 Glacier Vault Lock allows you to deploy and enforce compliance controls easily on individual Amazon S3 Glacier vaults via a lockable policy. You can specify controls such as *Write Once Read Many* (WORM) in a Vault Lock policy and lock the policy from future edits. Once locked, the policy becomes immutable, and Amazon S3 Glacier will enforce the prescribed controls to help achieve your compliance objectives.

Once you initiate a lock, you have 24 hours to validate your lock policy to ensure that it is working as you intended. Until that 24 hours is up, you can abort the lock and make changes. After 24 hours, that Vault Lock is permanent, and you will not be able to change it.

## ARCHIVES

An *archive* is any object, such as a photo, video, or document that you store in a vault. It is a base unit of storage in Amazon S3 Glacier. Each archive has a unique ID and optional description. When you upload an archive, Amazon S3 Glacier returns a response that includes an archive ID. This archive ID is unique in the region in which the archive is stored. You can retrieve an archive using its ID, but not its description.



Amazon S3 Glacier provides a management console to create and delete vaults. However, all other interactions with Amazon S3 Glacier require that you use the AWS CLI or write code.

To upload archives into your vaults, you must either use the AWS CLI or write code to make requests, using either the *REST API* directly or the AWS SDKs.

## MAINTAINING CLIENT-SIDE ARCHIVE METADATA

Except for the optional archive description, Amazon S3 Glacier does not support any additional metadata for the archives. When you upload an archive, Amazon S3 Glacier assigns an ID—an opaque sequence of characters—from which you cannot infer any meaning about the archive. Metadata about the archives can be maintained on the client side. The metadata can include identifying archive information such as the archive name.



If you use Amazon S3, when you upload an object to a bucket, you can assign the object an object key such as `MyDocument.txt` or `SomePhoto.jpg`. In Amazon S3 Glacier, you cannot assign a key name to the archives you upload.

If you maintain client-side archive metadata, note that Amazon S3 Glacier maintains a vault inventory that includes archive IDs and any descriptions that you provided during the archive upload. We recommend that you occasionally download the vault inventory to reconcile any issues in the client-side database that you maintain for the archive metadata. Amazon S3 Glacier takes vault inventory approximately once a day. When you request a vault inventory, Amazon S3 Glacier returns the last inventory it prepared, which is a point-in-time snapshot.

## USING THE AWS SDKS WITH AMAZON S3 GLACIER

AWS provides SDKs for you to develop applications for Amazon S3 Glacier in various programming languages.

The AWS SDKs for Java and .NET offer both high-level and low-level wrapper libraries. The SDK libraries wrap the underlying Amazon S3 Glacier API, simplifying your programming tasks. The low-level wrapper libraries map closely to the underlying REST API supported by Amazon S3 Glacier. To simplify application development further, these SDKs also

offer a higher-level abstraction for some of the operations in the high-level API. For example, when uploading an archive using the low-level API, if you need to provide a checksum of the payload, the high-level API computes the checksum for you.

### ENCRYPTION

All data in Amazon S3 Glacier will be encrypted on the server side using key management and key protection, which Amazon S3 Glacier handles using AES-256 encryption. If you want, you can manage your own keys and encrypt the data prior to uploading.

### RESTORING OBJECTS FROM AMAZON S3 GLACIER

Objects in the Amazon S3 Glacier storage class are not immediately accessible and cannot be retrieved via copy/paste once they have been moved to Amazon S3 Glacier.

Remember that Amazon S3 Glacier charges a retrieval fee for retrieving objects. When you restore an archive, you pay for both the archive and the restored copy. Because there is a storage cost for the copy, restore objects only for the duration that you need them. If you need a permanent copy of the object, create a copy of it in your Amazon S3 bucket.

### ARCHIVE RETRIEVAL OPTIONS

There are several different options for restoring archived objects from Amazon S3 Glacier to Amazon S3, as shown in Table 3.4.

**TABLE 3.4** Amazon S3 Glacier Archive Retrieval Options

| Retrieval Option    | Retrieval Time | Note                                                                                                                                          |
|---------------------|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| Expedited retrieval | 1–5 minutes    |                                                                                                                                               |
| On-Demand           |                | Processed immediately the vast majority of the time. During high demand, may fail to process, and you will be required to repeat the request. |
| Provisioned         |                | Guaranteed to process immediately. After purchasing provisioned capacity, all of your retrievals are processed in this manner.                |
| Standard retrieval  | 3–5 hours      |                                                                                                                                               |
| Bulk retrieval      | 5–12 hours     | Lowest-cost option                                                                                                                            |



Do not use Amazon S3 Glacier for backups if your RTO is shorter than the lowest Amazon S3 Glacier retrieval time for your chosen retrieval option. For example, if your RTO requires data retrieval of two hours in a disaster recovery scenario, then Amazon S3 Glacier standard retrieval will not meet your RTO.

## Storage Class Comparison

Table 3.5 shows a comparison of the Amazon S3 storage classes. This is an important table for the certification exam. Many storage decision questions on the exam center on the level of durability, availability, and cost. The table's comparisons can help you make the right choice for a question, in addition to understanding trade-offs when choosing a data store for an application.

**TABLE 3.5** Amazon S3 Storage Class Comparison

|                                    | Standard       | Standard_IA       | OneZone_IA | Amazon S3 Glacier |
|------------------------------------|----------------|-------------------|------------|-------------------|
| Designed for durability            | 99.999999999%* |                   |            |                   |
| Designed for availability          | 99.99%         | 99.9%             | 99.9%      | N/A               |
| Availability SLAs                  | 99.9%          | 99%               |            | N/A               |
| Availability zones                 | ≥3             |                   | 1          | ≥3                |
| Minimum capacity charge per object | N/A            | 128 KB*           |            | N/A               |
| Minimum storage duration charge    | N/A            | 30 days           |            | 90 days           |
| Retrieval fee                      | N/A            | Per GB retrieved* |            |                   |
| First byte latency                 | milliseconds   |                   |            | Minutes or hours* |
| Storage type                       | Object         |                   |            |                   |
| Lifecycle transitions              | Yes            |                   |            |                   |

\* Because One Zone\_IA stores data in a single Availability Zone, data stored in this storage class will be lost in the event of Availability Zone destruction. Standard\_IA has a minimum object size of 128 KB. Smaller objects will be charged for 128 KB of storage. Amazon S3 Glacier allows you to select from multiple retrieval tiers based upon your needs.

## Data Consistency Model

When deciding whether to choose Amazon S3 or Amazon EBS for your application, one important aspect to consider is the *consistency model* of the storage option. Amazon EBS

provides read-after-write consistency for all operations, whereas Amazon S3 provides read-after-write consistency only for PUTs of new objects.

Amazon S3 offers eventual consistency for overwrite PUTs and DELETEs in all regions, and updates to a single key are atomic. For example, if you PUT an object to update an existing object and immediately attempt to read that object, you may read either the old data or the new data.

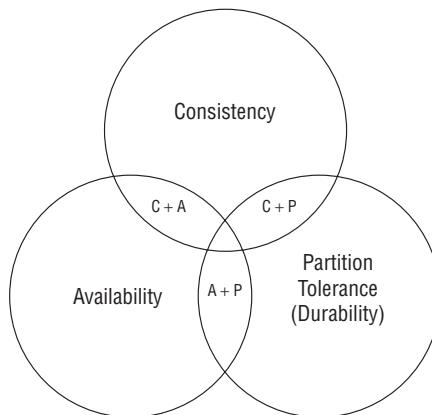
For PUT operations with new objects not yet in Amazon S3, you will experience read-after-write consistency. For PUT updates when you are overwriting an existing file or DELETE operations, you will experience eventual consistency.



Amazon S3 does not currently support object locking. If two PUT requests are simultaneously made to the same key, the request with the latest time stamp wins. If this is an issue, you will be required to build an object locking mechanism into your application.

You may be wondering why Amazon S3 was designed with this style of consistency. The *consistency, availability, and partition tolerance* theorem (CAP theorem) states that you can highly achieve only two out of the three dimensions for a particular storage design. The CAP theorem is shown in Figure 3.6.

**FIGURE 3.6** CAP theorem



Think of partition tolerance in this equation as the storage durability. Amazon S3 was designed for high availability and high durability (multiple copies across multiple facilities), so the design trade-off is the consistency. When you PUT an object, you are not only putting the object into one location but into three, meaning that there is either a slightly increased latency on the read-after-write consistency of a PUT or eventual consistency on the PUT.

update or `DELETE` operations while Amazon S3 reconciles all copies. You do not know, for instance, which facility a file is coming from when you `GET` an object. If you had recently written an object, it may have propagated to only two facilities, so when you try to read the object right after your `PUT`, you may receive the old object or the new object.

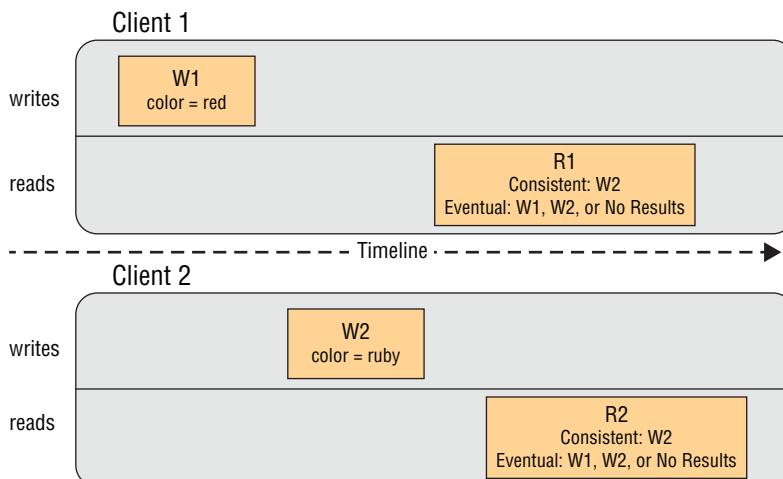
## Concurrent Applications

As a developer, it is critical to consider the way your application works and the consistency needs of your files. If your application requires read-after-write consistency on all operations, then Amazon S3 is not going to be the right choice for that application. If you are working with concurrent applications, it is important to know how your application performs `PUT`, `GET`, and `DELETE` operations *concurrently* to know whether eventual consistency will not be the right choice for your application.

In Figure 3.7, Amazon S3, both W1 (write 1) and W2 (write 2) complete before the start of R1 (read 1) and R2 (read 2). For a consistent read, R1 and R2 both return `color = ruby`. For an eventually consistent read, R1 and R2 might return `color = red`, `color = ruby`, or no results, depending on the amount of time that has elapsed.

**FIGURE 3.7** Consistency example 1

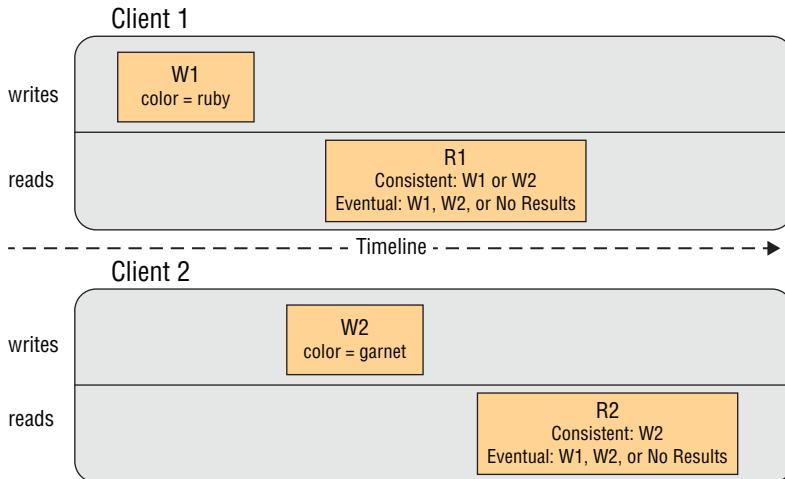
Domain = MyDomain, Item = StandardFez



In Figure 3.8, W2 does not complete before the start of R1. Therefore, R1 might return `color = ruby` or `color = garnet` for either a consistent read or an eventually consistent read. Depending on the amount of time that has elapsed, an eventually consistent read might also return no results.

**FIGURE 3.8** Consistency example 2

Domain = MyDomain, Item = StandardFez

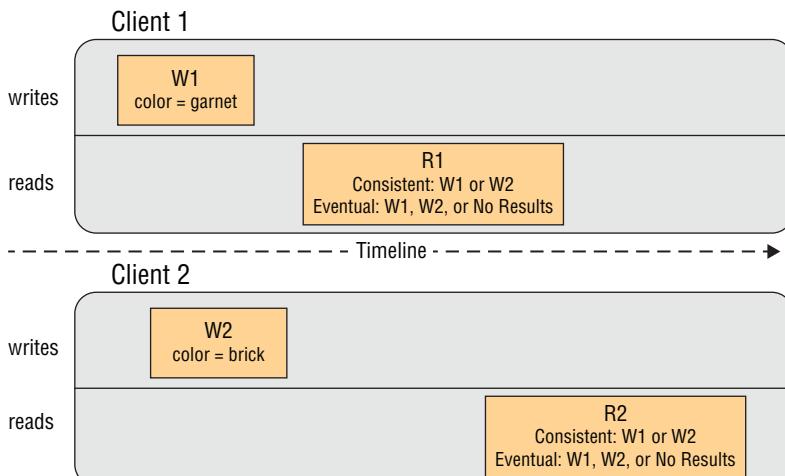


For a consistent read, R2 returns color = garnet. For an eventually consistent read, R2 might return color = ruby, color = garnet, or no results depending on the amount of time that has elapsed.

In Figure 3.9, client 2 performs W2 before Amazon S3 returns a success for W1, so the outcome of the final value is unknown (color = garnet or color = brick). Any subsequent reads (consistent read or eventually consistent) might return either value. Depending on the amount of time that has elapsed, an eventually consistent read might also return no results.

**FIGURE 3.9** Consistency example 3

Domain = MyDomain, Item = StandardFez





If you need a strongly consistent data store, choose a different data store than Amazon S3 or code consistency checks into your application.

## Presigned URLs

A *presigned URL* is a way to grant access to an object. One way that developers use presigned URLs is to allow users to upload or download objects without granting them direct access to Amazon S3 or the account.

For example, if you need to send a document hosted in an Amazon S3 bucket to an external reviewer who is outside of your organization, you do not want to grant them access using IAM to your bucket or objects. Instead, generate a presigned URL to the object and send that to the user to download your file.

Another example is if you need someone external to your organization to upload a file. Maybe a media company is designing the graphics for the website you are developing. You can create a presigned URL for them to upload their artifacts directly to Amazon S3 without granting them access to your Amazon S3 bucket or account.

Anyone with valid security credentials can create a presigned URL. For you to upload an object successfully, however, the presigned URL must be created by someone who has permission to perform the operation upon which the presigned URL is based.

The following Java code example demonstrates generating a presigned URL:

```
AmazonS3 s3Client = new AmazonS3Client(new ProfileCredentialsProvider());

java.util.Date expiration = new java.util.Date();
long msec = expiration.getTime();
msec += 1000 * 60 * 60; // Add 1 hour.
expiration.setTime(msec);

GeneratePresignedUrlRequest generatePresignedUrlRequest = new
GeneratePresignedUrlRequest(bucketName, objectKey);
generatePresignedUrlRequest.setMethod(HttpMethod.PUT);
generatePresignedUrlRequest.setExpiration(expiration);

URL s = s3client.generatePresignedUrl(generatePresignedUrlRequest);

// Use the pre-signed URL to upload an object.
```



Amazon S3 presigned URLs cannot be generated within the AWS Management Console, but they can be generated using the AWS CLI or AWS SDKs.

## Encryption

*Data protection* refers to protecting data while in transit (as it travels to and from Amazon S3) and at rest (while it is stored on Amazon S3 infrastructure). As a best practice, all sensitive data stored in Amazon S3 should be encrypted, both at rest and in transit.

You can protect *data in transit* by using Amazon S3 SSL API endpoints, which ensures that all data sent to and from Amazon S3 is encrypted using the *HTTPS protocol* while in transit.

For *data at rest* in Amazon S3, you can encrypt it using different options of *Server-Side Encryption (SSE)*. Your objects in Amazon S3 are encrypted at the object level as they are written to disk in the data centers and then decrypted for you when you access the objects using AES-256.

You can also use client-side encryption, with which you encrypt the objects before uploading to Amazon S3 and then decrypt them after you have downloaded them. Some customers, for some workloads, will use a combination of both server-side and client-side encryption for extra protection.

### Envelope Encryption Concepts

Before examining the different types of encryption available, we will review *envelope encryption*, which several AWS services use to provide a balance between performance and security.

The following steps describe how envelope encryption works:

1. A data key is generated by the AWS service at the time you request your data to be encrypted, as shown in Figure 3.10.

**FIGURE 3.10** Generating a data key



2. The data key generated in step 1 is used to encrypt your data, as shown in Figure 3.11.

**FIGURE 3.11** Encrypting the data



3. The data key is then encrypted with a key-encrypting key unique to the service storing your data, as shown in Figure 3.12.

**FIGURE 3.12** Encrypted data key



4. The encrypted data key and the encrypted data are then stored by the AWS storage service (such as Amazon S3 or Amazon EBS) on your behalf. This is shown in Figure 3.13.

**FIGURE 3.13** Encrypted data and key storage



When you need access to your plain-text data, this process is reversed. The encrypted data key is decrypted using the key-encrypting key, and the data key is then used to decrypt your data.



The important point to remember regarding envelope encryption is that the key-encrypting keys used to encrypt data keys are stored and managed separately from the data and the data keys.

### Server-Side Encryption (SSE)

You have three, mutually exclusive options for how you choose to manage your encryption keys when using SSE with Amazon S3.

**SSE-S3 (Amazon S3 managed keys)** You can set an API flag or check a box in the AWS Management Console to have data encrypted before it is written to disk in Amazon S3. Each object is encrypted with a unique data key. As an additional safeguard, this key is encrypted with a periodically-rotated master key managed by Amazon S3. AES-256 is used for both object and master keys. This feature is offered at no additional cost beyond what you pay for using Amazon S3.

**SSE-C (Customer-provided keys)** You can use your own encryption key while uploading an object to Amazon S3. This encryption key is used by Amazon S3 to encrypt your data using AES-256. After the object is encrypted, the encryption key you supplied is deleted from the Amazon S3 system that used it to encrypt your data. When you retrieve this object from Amazon S3, you must provide the same encryption key in your request. Amazon S3 verifies that the encryption key matches, decrypts the object, and returns the object to you. This feature is also offered at no additional cost beyond what you pay for using Amazon S3.

**SSE-KMS (AWS KMS managed encryption keys)** You can encrypt your data in Amazon S3 by defining an AWS KMS master key within your account to encrypt the unique object key (referred to as a *data key*) that will ultimately encrypt your object. When you upload your object, a request is sent to AWS KMS to create an object key. AWS KMS generates this object key and encrypts it using the master key that you specified earlier. Then, AWS KMS

returns this encrypted object key along with the plaintext object key to Amazon S3. The Amazon S3 web server encrypts your object using the plaintext object key and stores the now encrypted object (with the encrypted object key) and deletes the plaintext object key from memory.

To retrieve this encrypted object, Amazon S3 sends the encrypted object key to AWS KMS, which then decrypts the object key using the correct master key and returns the decrypted (plaintext) object key to Amazon S3. With the plaintext object key, Amazon S3 decrypts the encrypted object and returns it to you. Unlike SSE-S3 and SSE-C, using SSE-KMS does incur an additional charge. Refer to the AWS KMS pricing page on the AWS website for more information.



For maximum simplicity and ease of use, use SSE with AWS managed keys (SSE-S3 or SSE-KMS). Also, know the difference between SSE-S3, SSE-KMS, and SSE-C for SSE.

## Client-Side Encryption

*Client-side encryption* refers to encrypting your data before sending it to Amazon S3. You have two options for using data encryption keys.

### CLIENT-SIDE MASTER KEY

The first option is to use a client-side master key of your own. When uploading an object, you provide a client-side master key to the Amazon S3 encryption client (for example, `AmazonS3EncryptionClient` when using the AWS SDK for Java). The client uses this master key only to encrypt the data encryption key that it generates randomly. When downloading an object, the client first downloads the encrypted object from Amazon S3 along with the metadata. Using the material description in the metadata, the client first determines which master key to use to decrypt the encrypted data key. Then the client uses that master key to decrypt the data key and uses it to decrypt the object. The client-side master key that you provide can be either a symmetric key or a public/private key pair.

The process works as follows:

1. The Amazon S3 encryption client locally generates a one-time-use symmetric key (also known as a *data encryption key* or *data key*) and uses this data key to encrypt the data of a single Amazon S3 object (for each object, the client generates a separate data key).
2. The client encrypts the data encryption key using the master key that you provide.
3. The client uploads the encrypted data key and its material description as part of the object metadata. The material description helps the client later determine which client-side master key to use for decryption (when you download the object, the client decrypts it).
4. The client then uploads the encrypted data to Amazon S3 and also saves the encrypted data key as object metadata (`x-amz-meta-x-amz-key`) in Amazon S3 by default.

**AWS KMS-MANAGED CUSTOMER MASTER KEY (CMK)**

The second option is to use an AWS KMS managed customer master key (CMK). This process is similar to the process described earlier for using KMS-SSE, except that it is used for data at rest instead of data in transit. There is an Amazon S3 encryption client in the AWS SDK for Java.

**Using an AWS KMS Managed CMK (AWS SDK for Java)**

```
import java.io.ByteArrayInputStream;
import java.util.Arrays;

import junit.framework.Assert;

import org.apache.commons.io.IOUtils;

import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Region;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3EncryptionClient;
import com.amazonaws.services.s3.model.CryptoConfiguration;
import com.amazonaws.services.s3.model.KMSEncryptionMaterialsProvider;
import com.amazonaws.services.s3.model.ObjectMetadata;
import com.amazonaws.services.s3.model.PutObjectRequest;
import com.amazonaws.services.s3.model.S3Object;

public class testKMSkeyUploadObject {

 private static AmazonS3EncryptionClient encryptionClient;

 public static void main(String[] args) throws Exception {
 String bucketName = "***bucket name***";
 String objectKey = "ExampleKMSEncryptedObject";
 String kms_cmk_id = "***AWS KMS customer master key ID***";

 KMSEncryptionMaterialsProvider materialProvider = new
 KMSEncryptionMaterialsProvider(kms_cmk_id);

 encryptionClient = new AmazonS3EncryptionClient(new ProfileCredentials
 Provider(), materialProvider,
 new CryptoConfiguration().withKmsRegion(Regions.US_EAST_1))
 .withRegion(Region.getRegion(Regions.US_EAST_1));

 // Upload object using the encryption client.

 byte[] plaintext = "Hello World, S3 Client-side Encryption Using
 Asymmetric Master Key!"
 .getBytes();
 }
}
```

```
System.out.println("plaintext's length: " + plaintext.length);
encryptionClient.putObject(new PutObjectRequest(bucketName, objectKey,
 new ByteArrayInputStream(plaintext), new ObjectMetadata()));

// Download the object.
S3object downloadedObject = encryptionClient.getObject(bucketName,
 objectKey);
byte[] decrypted = IOUtils.toByteArray(downloadedObject
 .getObjectContent());

// Verify same data.
Assert.assertTrue(Arrays.equals(plaintext, decrypted));
}

}
```



Know the difference between CMK and client-side master keys for client-side encryption.

## Access Control

By default, all Amazon S3 resources—buckets, objects, and related sub-resources (for example, lifecycle configuration and website configuration)—are private. Only the resource owner, an account that created it, can access the resource. The resource owner can optionally grant access permissions to others by writing an access policy.

Amazon S3 offers access policy options broadly categorized as resource-based policies and user policies. Access policies that you attach to your resources (buckets and objects) are referred to as *resource-based policies*. For example, bucket policies and ACLs are resource-based policies. You can also attach access policies to users in your account. These are called *user policies*. You can choose to use resource-based policies, user policies, or some combination of both to manage permissions to your Amazon S3 resources. The following sections provide general guidelines for managing permissions.

### Using Bucket Policies and User Policies

Bucket policy and user policy are two of the access policy options available for you to grant permissions to your Amazon S3 resources. Both use a JSON-based access policy language, as do all AWS services that use policies.

A *bucket policy* is attached only to Amazon S3 buckets, and it specifies what actions are allowed or denied for whichever principals on the bucket to which the bucket policy is attached (for instance, allow user Alice to PUT but not DELETE objects in the bucket).

A *user policy* is attached to IAM users to perform or not perform actions on your AWS resources. For example, you may choose to grant an IAM user in your account access to

one of your buckets and allow the user to add, update, and delete objects. You can grant them access with a user policy.

Now we will discuss the differences between IAM policies and Amazon S3 bucket policies. Both are used for access control, and they are both written in JSON using the AWS access policy language. However, unlike Amazon S3 bucket policies, IAM policies specify what actions are allowed or denied on what AWS resources (such as, allow `ec2:TerminateInstance` on the Amazon EC2 instance with `instance_id=i8b3620ec`). You attach IAM policies to IAM users, groups, or roles, which are then subject to the permissions that you have defined. Instead of attaching policies to the users, groups, or roles, bucket policies are attached to a specific resource, such as an Amazon S3 bucket.

### Managing Access with Access Control Lists

*Access with access control lists* (ACLs) are resource-based access policies that you can use to manage access to your buckets and objects, including granting basic read/write permissions to other accounts.

There are limits to managing permissions using ACLs. For example, you can grant permissions only to other accounts; you cannot grant permissions to users in your account. You cannot grant conditional permissions, nor can you explicitly deny permissions using ACLs.

ACLs are suitable only for specific scenarios (for example, if a bucket owner allows other accounts to upload objects), and permissions to these objects can be managed only using an object ACL by the account that owns the object.



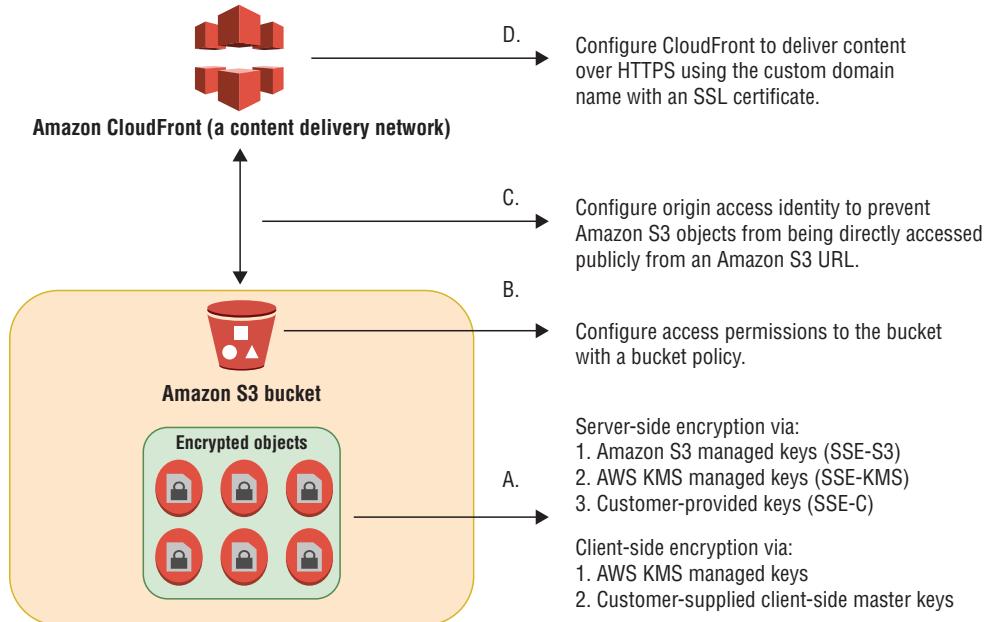
You can only grant access to other accounts using ACLs—not users in your own account.

### Defense in Depth—Amazon S3 Security

Amazon S3 provides comprehensive security and compliance capabilities that meet the most stringent regulatory requirements, and it gives you flexibility in the way that you manage data for cost optimization, access control, and compliance. With this flexibility, however, comes the responsibility of ensuring that your content is secure.

You can use an approach known as *defense in depth* in Amazon S3 to secure your data. This approach uses multiple layers of security to ensure redundancy if one of the multiple layers of security fails.

Figure 3.14 represents defense in depth visually. It contains several Amazon S3 objects (A) in a single Amazon S3 bucket (B). You can encrypt these objects on the server side or the client side, and you can also configure the bucket policy such that objects are accessible only through Amazon CloudFront, which you can accomplish through an origin access identity (C). You can then configure Amazon CloudFront to deliver content only over HTTPS in addition to using your own domain name (D).

**FIGURE 3.14** Defense in depth on Amazon S3

To meet defense in depth requirements on Amazon S3:

- Data must be encrypted at rest and during transit.
- Data must be accessible only by a limited set of public IP addresses.
- Data must not be publicly accessible directly from an Amazon S3 URL.
- A domain name is required to consume the content.

You can apply policies to Amazon S3 buckets so that only users with appropriate permissions are allowed to access the buckets. Anonymous users (with public-read/public-read-write permissions) and authenticated users without the appropriate permissions are prevented from accessing the buckets.

You can also secure access to objects in Amazon S3 buckets. The objects in Amazon S3 buckets can be encrypted at rest and during transit to provide end-to-end security from the source (in this case, Amazon S3) to your users.

## Query String Authentication

You can provide authentication information using *query string parameters*. Using query parameters to authenticate requests is useful when expressing a request entirely in a URL. This method is also referred to as *presigning* a URL.

With presigned URLs, you can grant temporary access to your Amazon S3 resources. For example, you can embed a presigned URL on your website, or alternatively use it in a command line client (such as Curl), to download objects.

The following is an example presigned URL:

```
https://s3.amazonaws.com/examplebucket/test.txt
?X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=<your-access-key-id>/20130721/us-east-1/s3/aws4_request
&X-Amz-Date=20130721T201207Z
&X-Amz-Expires=86400
&X-Amz-SignedHeaders=host
&X-Amz-Signature=<signature-value>
```

In the example URL, note the following:

- The line feeds are added for readability.
- The X-Amz-Credential value in the URL shows the / character only for readability. In practice, it should be encoded as %2F.

## Hosting a Static Website

If your website contains static content and optionally client-side scripts, then you can host your *static website* directly in Amazon S3 without the use of web-hosting servers.

To host a static website, you configure an Amazon S3 bucket for website hosting and upload your website content to the bucket. The website is then available at the AWS Region-specific website endpoint of the bucket in one of the following formats:

```
<bucket-name>.s3-website-<AWS-region>.amazonaws.com
<bucket-name>.s3-website.<AWS-region>.amazonaws.com
```

Instead of accessing the website by using an Amazon S3 website endpoint, use your own domain (for instance, example.com) to serve your content. The following steps allow you to configure your own domain:

1. Register your domain with the registrar of your choice. You can use Amazon Route 53 to register your domain name or any other third-party domain registrar.
2. Create your bucket in Amazon S3 and upload your static website content.
3. Point your domain to your Amazon S3 bucket using either of the following as your DNS provider:
  - Amazon Route 53
  - Your third-party domain name registrar

Amazon S3 does not support server-side scripting or dynamic content. We discuss other AWS options for that throughout this study guide.



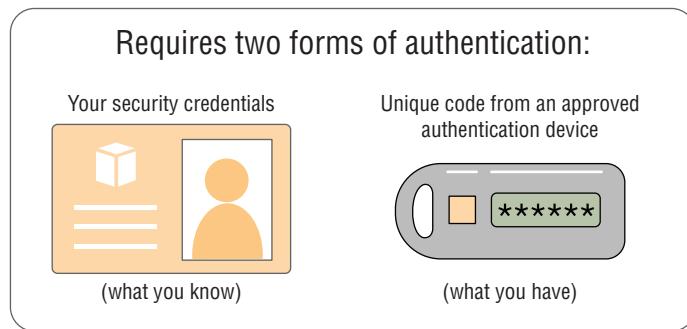
---

Static websites can be hosted in Amazon S3.

## MFA Delete

MFA is another way to control deletes on your objects in Amazon S3. It does so by adding another layer of protection against unintentional or malicious deletes, requiring an authorized request against Amazon S3 to delete the object. MFA also requires a unique code from a token or an authentication device (virtual or hardware). These devices provide a unique code that will then allow you to delete the object. Figure 3.15 shows what would be required for a user to execute a delete operation on an object when MFA is enabled.

**FIGURE 3.15** MFA Delete



## Cross-Region Replication

*Cross-region replication* (CRR) is a bucket-level configuration that enables automatic, asynchronous copying of objects across buckets in different AWS Regions. We refer to these buckets as the *source* bucket and *destination* bucket. These buckets can be owned by different accounts.

To activate this feature, add a replication configuration to your source bucket to direct Amazon S3 to replicate objects according to the configuration. In the replication configuration, provide information including the following:

- The destination bucket
- The objects that need to be replicated
- Optionally, the destination storage class (otherwise the source storage class will be used)

The replicas that are created in the destination bucket will have these same characteristics as the source objects:

- Key names
- Metadata
- Storage class (unless otherwise specified)
- Object ACL

All data is encrypted in transit across AWS Regions using SSL.

You can replicate objects from a source bucket to only one destination bucket. After Amazon S3 replicates an object, the object cannot be replicated again. For example, even after you change the destination bucket in an existing replication configuration, Amazon S3 will not replicate it again.



After Amazon S3 replicates an object using CRR, the object cannot be replicated again (such as to another destination bucket).

Requirements for CRR include the following:

- Versioning is enabled for both the source and destination buckets.
- Source and destination buckets must be in different AWS Regions.
- Amazon S3 must be granted appropriate permissions to replicate files.

## VPC Endpoints

A *virtual private cloud (VPC) endpoint* enables you to connect your VPC privately to Amazon S3 without requiring an internet gateway, *network address translation (NAT)* device, *virtual private network (VPN)* connection, or *AWS Direct Connect* connection. Instances in your VPC do not require public IP addresses to communicate with the resources in the service. Traffic between your VPC and Amazon S3 does not leave the Amazon network.

Amazon S3 uses a gateway type of VPC endpoint. The gateway is a target for a specified route in your route table, used for traffic destined for a supported AWS service. These endpoints are easy to configure, are highly reliable, and provide a secure connection to Amazon S3 that does not require a gateway or NAT instance.

Amazon EC2 instances running in private subnets of a VPC can have controlled access to Amazon S3 buckets, objects, and API functions that are in the same region as the VPC. You can use an Amazon S3 bucket policy to indicate which VPCs and which VPC endpoints have access to your Amazon S3 buckets.

## Using the AWS SDKs, AWS CLI, and AWS Explorers

You can use the AWS SDKs when developing applications with Amazon S3. The AWS SDKs simplify your programming tasks by wrapping the underlying REST API. The AWS Mobile SDKs and the AWS Amplify JavaScript library are also available for building connected mobile and web applications using AWS. In addition to AWS SDKs, AWS explorers are available for Visual Studio and Eclipse for *Java Integrated Development Environment (IDE)*. In this case, the SDKs and AWS explorers are available bundled together as AWS Toolkits. You can also use the AWS CLI to manage Amazon S3 buckets and objects.

AWS has deprecated SOAP support over HTTP, but it is still available over HTTPS. New Amazon S3 features will not be supported over SOAP. We recommend that you use

either the REST API or the AWS SDKs for any new development and migrate any existing SOAP calls when you are able.

## Making Requests

Every interaction with Amazon S3 is either authenticated or anonymous. *Authentication* is the process of verifying the identity of the requester trying to access an AWS product (you are who you say you are, and you are allowed to do what you are asking to do). Authenticated requests must include a signature value that authenticates the request sender, generated in part from the requester's AWS access keys. If you are using the AWS SDK, the libraries compute the signature from the keys that you provide. If you make direct REST API calls in your application, however, you must write code to compute the signature and add it to the request.

## Stateless and Serverless Applications

Amazon S3 provides developers with secure, durable, and highly scalable object storage that can be used to decouple storage for use in *serverless applications*. Developers can also use Amazon S3 for storing and sharing state in *stateless applications*.

Developers on AWS are regularly moving shared file storage to Amazon S3 for stateless applications. This is a common method for decoupling your compute and storage and increasing the ability to scale your application by decoupling that storage. We will discuss stateless and serverless applications throughout this study guide.

## Data Lake

Traditional data storage can no longer provide the agility and flexibility required to handle the volume, velocity, and variety of data used by today's applications. Because of this, many organizations are shifting to a *data lake* architecture.

A *data lake* is an architectural approach that allows you to store massive amounts of data in a central location for consumption by multiple applications. Because data can be stored as is, there is no need to convert it to a predefined schema, and you no longer need to know what questions to ask of your data beforehand.

Amazon S3 is a common component of a data lake solution on the cloud, and it can complement your other storage solutions. If you move to a data lake, you are essentially separating compute and storage, meaning that you are going to build and scale your storage and compute separately. You can take storage that you currently have on premises or in your data center and instead use Amazon S3, which then allows you to scale and build your compute in any desired configuration, regardless of your storage.

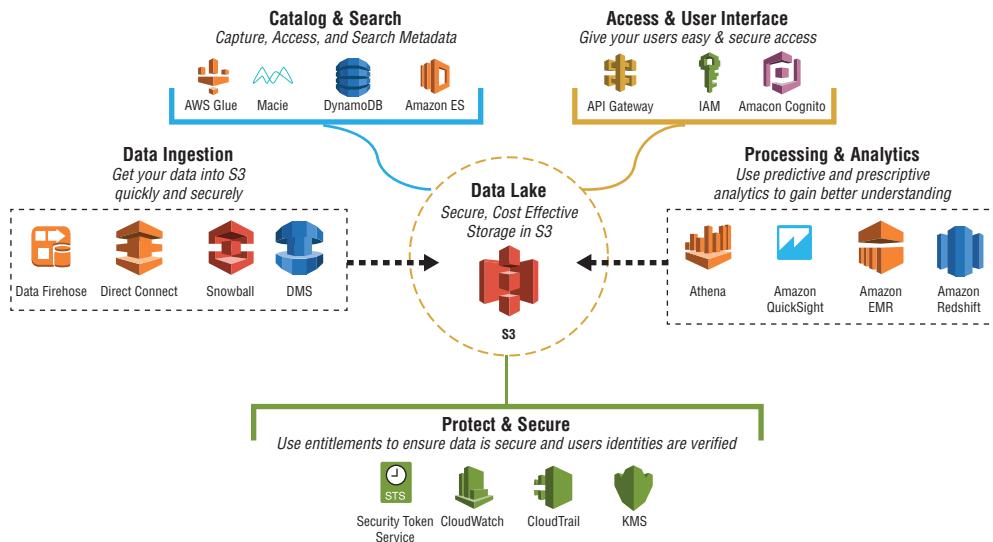
That design pattern is different from most applications available today, where the storage is tied to the compute. When you separate those two features and instead use a data lake, you achieve an agility that allows you to invent new types of applications while you are managing your storage as an independent entity.

In addition, Amazon S3 lets you grow and scale in a virtually unlimited fashion. You do not have to take specific actions to expand your storage capacity—it grows automatically with your data.

In the data lake diagram shown in Figure 3.16, you will see how to use Amazon S3 as a highly available and durable central storage repository. From there, a virtually unlimited

number of services and applications, both on premises and in the cloud, can take advantage of using Amazon S3 as a data lake.

**FIGURE 3.16** Data lakes



Customers often set up a data lake as part of their migration to the cloud so that they can access their data from new applications on the cloud, migrated applications to the cloud, and applications that have not yet been migrated to the cloud.

## Performance

There are a number of actions that Amazon S3 takes by default to help you achieve high levels of performance. Amazon S3 automatically scales to thousands of requests per second per prefix based on your steady state traffic. Amazon S3 will automatically partition your prefixes within hours, adjusting to increases in request rates.

### Consideration for Workloads

To optimize the use of Amazon S3 mixed or GET-intensive workloads, you must become familiar with best practices for performance optimization.

**Mixed request types** If your requests are typically a mix of GET, PUT, DELETE, and GET Bucket (list objects), choosing appropriate key names for your objects ensures better performance by providing low-latency access to the Amazon S3 index.

**GET-intensive workloads** If the bulk of your workload consists of GET requests, you may want to use Amazon CloudFront, a content delivery service (discussed later in this chapter).

## Tips for Object Key Naming

The way that you name your keys in Amazon S3 can affect the data access patterns, which may directly impact the performance of your application.

It is a best practice at AWS to design for performance from the start. Even though you may be developing a new application, that application is likely to grow over time. If you anticipate your application growing to more than approximately 1,000 requests per second (including both PUTs and GETs on your object), you will want to consider using a three- or four-character hash in your key names.

If you anticipate your application receiving fewer than 1,000 requests per second and you don't see a lot of traffic in your storage, then you do not need to implement this best practice. Your application will still benefit from Amazon S3's default performance.



In the past, customers would also add entropy in their key names. Because of recent Amazon S3 performance enhancements, most customers no longer need to worry about introducing entropy in key names.

### Example 1: Random Hash

examplebucket/**232a**-2017-26-05-15-00-00/cust1234234/photo1.jpg

examplebucket/**7b54**-2017-26-05-15-00-00/cust3857422/photo2.jpg

examplebucket/**921c**-2017-26-05-15-00-00/cust1248473/photo2.jpg



A random hash should come before patterns, such as dates and sequential IDs.

Using a *naming hash* can improve the performance of heavy-traffic applications. Object keys are stored in an index in all regions. If you're constantly writing the same key prefix over and over again (for example, a key with the current year), all of your objects will be close to each other within the same partition in the index. When your application experiences an increase in traffic, it will be trying to read from the same section of the index, resulting in decreased performance as Amazon S3 tries to spread out your data to achieve higher levels of throughput.



Always first ensure that your application can accommodate a naming hash.

By putting the hash at the beginning of your key name, you are adding randomness. You could hash the key name and place it at the beginning of your object right after the bucket name. This will ensure that your data will be spread across different partitions and allow you to grow to a higher level of throughput without experiencing a re-indexing slowdown if you go above peak traffic volumes.

**Example 2:** Naming Hash

```
examplebucket/animations/232a-2017-26-05-15-00/cust1234234/animation1.obj
```

```
examplebucket/videos/ba65-2017-26-05-15-00/cust8474937/video2.mpg
```

```
examplebucket/photos/8761-2017-26-05-15-00/cust1248473/photo3.jpg
```

In this second example, imagine that you are storing a lot of animations, videos, and photos in Amazon S3. If you know that you are going to have a lot of traffic to those individual prefixes, you can add your hash after the prefix. That allows you to write prefixes into your lifecycle policies or perform *list API* calls against a particular prefix. You are still getting the performance benefit by adding the hash to your key name, but now you can also use the prefix as necessary.

This example allows you to balance the need to list your objects and organize them against the need to spread your data across different partitions for performance.

### Amazon S3 Transfer Acceleration

*Amazon S3 Transfer Acceleration* is a feature that optimizes throughput when transferring larger objects across larger geographic distances. Amazon S3 Transfer Acceleration uses Amazon CloudFront edge locations to assist you in uploading your objects more quickly in cases where you are closer to an edge location than to the region to which you are transferring your files.

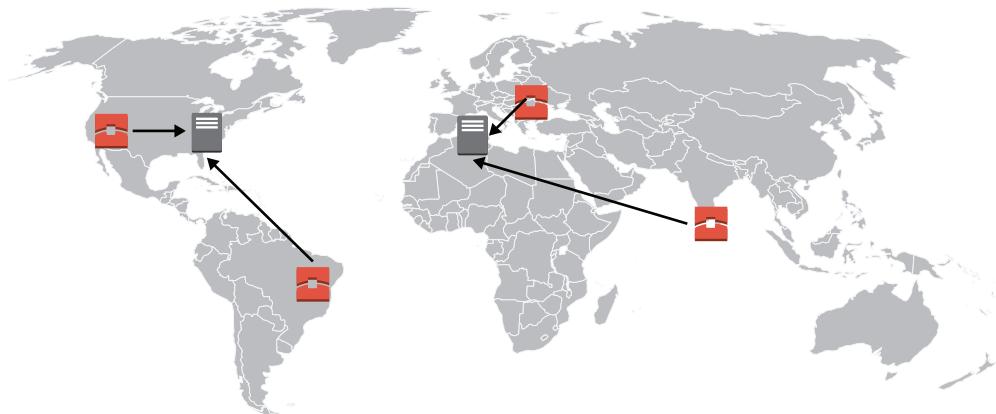
Instead of using the public internet to upload objects from Southeast Asia, across the globe to Northern Virginia, take advantage of the global *Amazon content delivery network* (CDN). AWS has edge locations around the world, and you upload your data to the edge location closest to your location. This way, you are traveling across the AWS network backbone to your destination region, instead of across the public internet. This option might give you a significant performance improvement and better network consistency than the public internet.

To implement Amazon S3 Transfer Acceleration, you do not need to make any changes to your application. It is enabled by performing the following steps:

1. Enable Transfer Acceleration on a bucket that conforms to DNS naming requirements and does not contain periods (.).
2. Transfer data to and from the acceleration-enabled bucket by using one of the s3-accelerate endpoint domain names.

There is a small fee for using Transfer Acceleration. If your speed using Transfer Acceleration is no faster than it would have been going over the public internet, however, there is no additional charge.

The further you are from a particular region, the more benefit you will derive from transferring your files more quickly by uploading to a closer edge location. Figure 3.17 shows how accessing an edge location can reduce the latency for your users, as opposed to accessing content from a region that is farther away.

**FIGURE 3.17** Using an AWS edge location

### Multipart Uploads

When uploading a large object to Amazon S3 in a single-threaded manner, it can take a significant amount of time to complete. The multipart upload API enables you to upload large objects in parts to speed up your upload by doing so in parallel.

To use multipart upload, you first break the object into smaller parts, parallelize the upload, and then submit a manifest file telling Amazon S3 that all parts of the object have been uploaded. Amazon S3 will then assemble all of those individual pieces to a single Amazon S3 object.

Multipart upload can be used for objects ranging from 5 MB to 5 TB in size.

### Range GETs

*Range GETs* are similar to multipart uploads, but in reverse. If you are downloading a large object and tracking the offsets, use range GETs to download the object as multiple parts instead of a single part. You can then download those parts in parallel and potentially see an improvement in performance.

### Amazon CloudFront

Using a CDN like Amazon CloudFront, you may achieve lower latency and higher-throughput performance. You also will not experience as many requests to Amazon S3 because your content will be cached at the edge location. Your users will also experience the performance improvement of having cached storage through Amazon CloudFront versus going back to Amazon S3 for each new GET on an object.

### TCP Window Scaling

Transmission Control Protocol (TCP) window scaling allows you to improve network throughput performance between your operating system, application layer, and Amazon S3 by supporting window sizes larger than 64 KB. Although it can improve performance,

it can be challenging to set up correctly, so refer to the AWS Documentation repository for details.

### TCP Selective Acknowledgment

*TCP selective acknowledgment* is designed to improve recovery time after a large number of packet losses. It is supported by most newer operating systems, but it might have to be enabled. Refer to the Amazon S3 Developer Guide for more information.

## Pricing

With Amazon S3, you pay only for what you use. There is no minimum fee, and there is no charge for data transfer into Amazon S3.

You pay for the following:

- The storage that you use
- The API calls that you make (PUT, COPY, POST, LIST, GET)
- Data transfer out of Amazon S3

Data transfer out pricing is tiered, so the more you use, the lower your cost per gigabyte. Refer to the AWS website for the latest pricing.



Amazon S3 pricing differs from the pricing of Amazon EBS volumes in that if you create an Amazon EBS volume and store nothing on it, you are still paying for the storage space of the volume that you have allocated. With Amazon S3, you pay for the storage space that is being used—not allocated.

## Object Lifecycle Management

To manage your objects so that they are stored cost effectively throughout their lifecycle, use a *lifecycle configuration*. A lifecycle configuration is a set of rules that defines actions that Amazon S3 applies to a group of objects.

There are two types of actions:

**Transition actions** *Transition actions* define when objects transition to another storage class. For example, you might choose to transition objects to the STANDARD\_IA storage class 30 days after you created them or archive objects to the GLACIER storage class one year after creating them.

**Expiration actions** *Expiration actions* define when objects expire. Amazon S3 deletes expired objects on your behalf.

## When Should You Use Lifecycle Configuration?

You should use lifecycle configuration rules for objects that have a well-defined lifecycle. The following are some examples:

- If you upload periodic logs to a bucket, your application might need them for a week or a month. After that, you may delete them.
- Some documents are frequently accessed for a limited period of time. After that, they are infrequently accessed. At some point, you might not need real-time access to them, but your organization or regulations might require you to archive them for a specific period. After that, you may delete them.
- You can upload some data to Amazon S3 primarily for archival purposes. For example, archiving digital media, financial, and healthcare records; raw genomics sequence data, long-term database backups; and data that must be retained for regulatory compliance.

With lifecycle configuration rules, you can tell Amazon S3 to transition objects to less expensive storage classes or archive or delete them.

## Configuring a Lifecycle

A *lifecycle configuration* (an XML file) comprises a set of rules with predefined actions that you need Amazon S3 to perform on objects during their lifetime. Amazon S3 provides a set of API operations for managing lifecycle configuration on a bucket, and it is stored by Amazon S3 as a *lifecycle subresource* that is attached to your bucket.

You can also configure the lifecycle by using the Amazon S3 console, the AWS SDKs, or the REST API.

The following lifecycle configuration specifies a rule that applies to objects with key name prefix `logs/`. The rule specifies the following actions:

- Two transition actions
  - Transition objects to the STANDARD\_IA storage class 30 days after creation
  - Transition objects to the GLACIER storage class 90 days after creation
- One expiration action that directs Amazon S3 to delete objects a year after creation

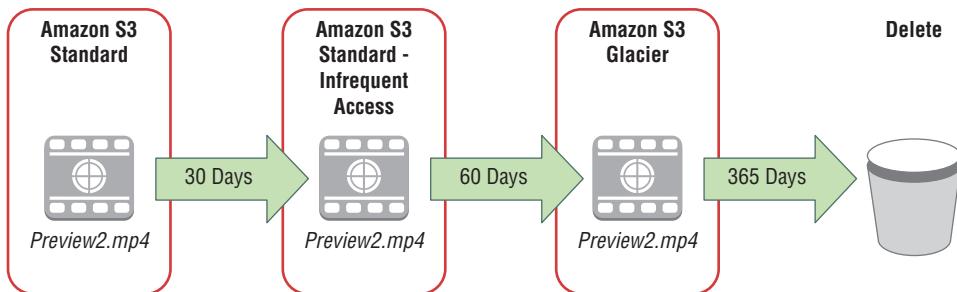
```
<LifecycleConfiguration>
 <Rule>
 <ID>example-id</ID>
 <Filter>
 <Prefix>logs/</Prefix>
 </Filter>
 <Status>Enabled</Status>
 <Transition>
 <Days>30</Days>
 <StorageClass>STANDARD_IA</StorageClass>
 </Transition>
 <Expiration>365</Expiration>
 </Rule>
</LifecycleConfiguration>
```

```
<Transition>
 <Days>90</Days>
 <StorageClass>GLACIER</StorageClass>
</Transition>
<Expiration>
 <Days>365</Days>
</Expiration>
</Rule>
</LifecycleConfiguration>
```

Figure 3.18 shows a set of Amazon S3 lifecycle policies in place. These policies move files automatically from one storage class to another as they age out at certain points in time.

**FIGURE 3.18** Amazon S3 lifecycle policies

*Amazon S3 lifecycle policies* allow you to delete or move objects based on age.



## AWS File Storage Services

AWS offers Amazon Elastic File System (Amazon EFS) for file storage to enable you to share access to files that reside on the cloud.

### Amazon Elastic File System

*Amazon Elastic File System* (Amazon EFS) provides scalable file storage and a standard file system interface for use with Amazon EC2. You can create an Amazon EFS file system, configure your instances to mount the file system, and then use an Amazon EFS file system as a common data source for workloads and application running on multiple instances.

Amazon EFS can be mounted to multiple Amazon EC2 instances simultaneously, where it can continue to expand up to petabytes while providing low latency and high throughput.

Consider using Amazon EFS instead of Amazon S3 or Amazon EBS if you have an application (Amazon EC2 or on premises) or a use case that requires a file system and any of the following:

- Multi-attach
- GB/s throughput
- Multi-AZ availability/durability
- Automatic scaling (growing/shrinking of storage)

Customers use Amazon EFS for the following use cases today:

- Web serving
- Database backups
- Container storage
- Home directories
- Content management
- Analytics
- Media and entertainment workflows
- Workflow management
- Shared state management



Amazon EFS is not supported on Windows instances.

## Creating your Amazon EFS File System

### File System

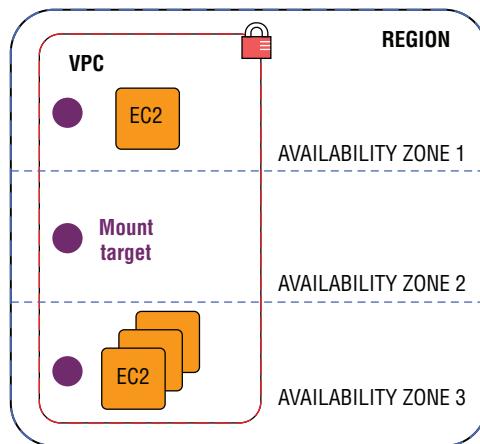
The *Amazon EFS file system* is the primary resource in Amazon EFS, and it is where you store your files and directories. You can create up to 125 file systems per account.

### Mount Target

To access your file system from within a VPC, create mount targets in the VPC. A *mount target* is a Network File System (NFS) endpoint within your VPC that includes an IP address and a DNS name, both of which you use in your mount command. A mount target is highly available, and it is illustrated in Figure 3.19.

## Accessing an Amazon EFS File System

There are several different ways that you can access an Amazon EFS file system, including using Amazon EC2 and AWS Direct Connect.

**FIGURE 3.19** Mount target

### Using Amazon Elastic Compute Cloud

To access a file system from an *Amazon Elastic Compute Cloud* (Amazon EC2) instance, you must mount the file system by using the standard Linux `mount` command, as shown in Figure 3.20. The file system will then appear as a local set of directories and files. An NFS v4.1 client is standard on Amazon Linux AMI distributions.

**FIGURE 3.20** Mounting the file system

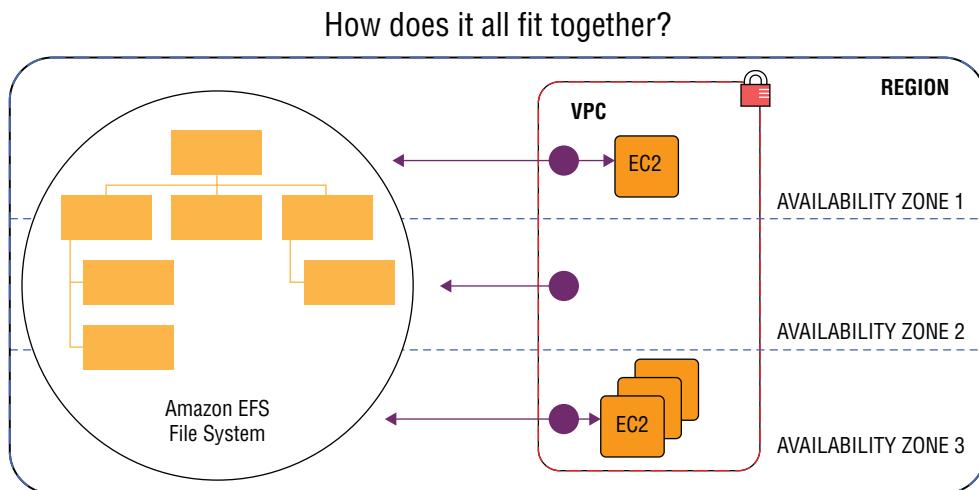
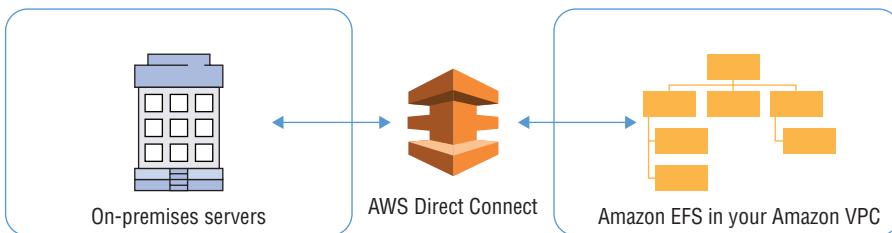
```
mount -t nfs4 -o nfsvers=4.1
 [file system DNS name]:/
 /[user's target directory]
```

In your command, specify the file system type (`nfs4`), the version (4.1), the file system DNS name or IP address, and the user's target directory.

A file system belongs to a region, and your Amazon EFS file system spans all Availability Zones in that region. Once you have mounted your file system, data can be accessed from any Availability Zone in the region within your VPC while maintaining full consistency. Figure 3.21 shows how you communicate with Amazon EC2 instances within a VPC.

### Using AWS Direct Connect

You can also mount your on-premises servers to Amazon EFS in your Amazon VPC using AWS Direct Connect. With *AWS Direct Connect*, you can mount your on-premises servers to Amazon EFS using the same mount command used to mount in Amazon EC2. Figure 3.22 shows how to use AWS Direct Connect with Amazon EFS.

**FIGURE 3.21** Using Amazon EFS**FIGURE 3.22** Using AWS Direct Connect with Amazon EFS

Customers can use Amazon EFS combined with AWS Direct Connect for migration, bursting, or backup and disaster recovery.

## Syncing Files Using AWS DataSync

Now that you have a functioning Amazon EFS file system, you can use *AWS DataSync* to synchronize files from an existing file system to Amazon EFS. AWS DataSync can synchronize your file data and also file system metadata such as ownership, time stamps, and access permissions.

To do this, download and deploy a sync agent from the Amazon EFS console as either a *virtual machine* (VM) image or an AMI.

Next, create a sync task and configure your source and destination file systems. Then start your task to begin syncing the files and monitor the progress of the file sync using Amazon CloudWatch.

## Performance

Amazon EFS is designed for a wide spectrum of performance needs, including the following:

- High throughput and parallel I/O
- Low latency and serial I/O

To support those two sets of workloads, Amazon EFS offers two different performance modes, as described here:

**General purpose (default)** General-purpose mode is the default mode, and it is used for latency-sensitive applications and general-purpose workloads, offering the lowest latencies for file operations. While there is a trade-off of limiting operations to 7,000 per second, general-purpose mode is the best choice for most workloads.

**Max I/O** If you are running large-scale and data-heavy applications, then choose the max I/O performance option, which provides you with a virtually unlimited ability to scale out throughput and IOPS, but with a trade-off of slightly higher latencies. Use max I/O when you have 10 or more instances accessing your file system concurrently, as shown in Table 3.6.

**TABLE 3.6** I/O Performance Options

Mode	What's It For?	Advantages	Trade-Offs	When to Use
General purpose (default)	Latency-sensitive applications and general-purpose workloads	Lowest latencies for file operations	Limit of 7,000 ops/sec	Best choice for most workloads
Max I/O	Large-scale and data-heavy applications	Virtually unlimited ability to scale out throughput/ IOPS	Slightly higher latencies	Consider if 10 (or more) instances are accessing your file system concurrently

If you are not sure which mode is best for your usage pattern, use the PercentIOLimit Amazon CloudWatch metric to determine whether you are constrained by general-purpose mode. If you are regularly hitting the 7,000 IOPS limit in general-purpose mode, then you will likely benefit from max I/O performance mode.

As discussed with the CAP theorem earlier in this study guide, there are differences in both performance and trade-off decisions when you're designing systems that use Amazon EFS and Amazon EBS. The distributed architecture of Amazon EFS results in a small increase in latency for each operation, as the data that you are storing gets pushed across multiple servers in multiple Availability Zones. Amazon EBS can provide lower latency

than Amazon EFS, but at the cost of some durability. With Amazon EBS, you provision the size of the device, and if you reach its maximum limit, you must increase its size or add more volumes, whereas Amazon EFS scales automatically. Table 3.7 shows the various performance and other characteristics for Amazon EFS as related to Amazon EBS Provisioned IOPS.

**TABLE 3.7** Amazon EBS Performance Relative to Amazon EFS

		Amazon EFS	Amazon EBS Provisioned IOPS
<b>Performance</b>	Per-operation latency	Low, consistent	Lowest, consistent
	Throughput scale	Multiple GBs per second	Single GB per second
<b>Characteristics</b>	Data availability/durability	Stored redundantly across multiple Availability Zones	Stored redundantly in a single Availability Zone
	Access	1 to 1000s of EC2 instances, from multiple Availability Zones, concurrently	Single Amazon EC2 instance in a single Availability Zone
	Use cases	Big Data and analytics, media processing workflows, content management, web serving, home directories	Boot volumes, transactional and NoSQL databases, data warehousing, ETL

## Security

You can implement security in multiple layers with Amazon EFS by controlling the following:

- Network traffic to and from file systems (mount targets) using the following:
  - VPC security groups
  - Network ACLs
- File and directory access by using POSIX permissions
- Administrative access (API access) to file systems by using IAM. Amazon EFS supports:
  - Action-level permissions
  - Resource-level permissions



Familiarize yourself with the Amazon EFS product, details, and FAQ pages. Some exam questions may be answered by components from those pages.

# Storage Comparisons

This section provides valuable charts that can serve as a quick reference if you are tasked with choosing a storage system for a particular project or application.

## Use Case Comparison

Table 3.8 will help you understand the main properties and use cases for each of the cloud storage products on AWS.

**TABLE 3.8** AWS Cloud Storage Products

If You Need:	Consider Using:
Persistent local storage for Amazon EC2, relational and NoSQL databases, data warehousing, enterprise applications, big data processing, or backup and recovery	Amazon EBS
A file system interface and file system access semantics to make data available to one or more Amazon EC2 instances for content serving, enterprise applications, media processing workflows, big data storage, or backup and recovery	Amazon EFS
A scalable, durable platform to make data accessible from any internet location for user-generated content, active archive, serverless computing, Big Data storage, or backup and recovery	Amazon S3
Highly affordable, long-term storage that can replace tape for archive and regulatory compliance	Amazon S3 Glacier
A hybrid storage cloud augmenting your on-premises environment with AWS cloud storage for bursting, tiering, or migration	AWS Storage Gateway
A portfolio of services to help simplify and accelerate moving data of all types and sizes into and out of the AWS Cloud	AWS Cloud Data Migration Services

## Storage Temperature Comparison

Table 3.9 shows a comparison of instance store, Amazon EBS, Amazon S3, and Amazon S3 Glacier.



Understanding Table 3.9 will help you make decisions about latency, size, durability, and cost during the exam.

**TABLE 3.9** Storage Comparison

	Instance Store	Amazon EBS	Amazon S3	Amazon S3 Glacier
<b>Average latency</b>	ms		ms, sec, min (~ size)	hrs
<b>Data volume</b>	4 GB to 48 TB	1 GiB to 1 TiB	No limit	
<b>Item size</b>	Block storage		5 TB max	40 TB max
<b>Request rate</b>	Very high		Low to very high (no limit)	Very low (no limit)
<b>Cost/GB per month</b>	Amazon EC2 instance cost	¢¢	¢	
<b>Durability</b>	Low	High	Very high	Very high
<b>Temperature</b>	Hot <-----> Cold			

## Comparison of Amazon EBS and Instance Store

Before considering Amazon EC2 instance store as a storage option, make sure that your data does *not* meet any of these criteria:

- Must persist through instance stops, terminations, or hardware failures
- Needs to be encrypted at the full volume level
- Needs to be backed up with Amazon EBS snapshots
- Needs to be removed from instances and reattached to another

If your data meets any of the previous four criteria, use an Amazon EBS volume. Otherwise, compare instance store and Amazon EBS for storage.

Because instance store is directly attached to the host computer, it will have lower latency than an Amazon EBS volume attached to the Amazon EC2 instance. Instance store is provided at no additional cost beyond the price of the Amazon EC2 instance you choose (if the instance has instance store[s] available), whereas Amazon EBS volumes incur an additional cost.

## Comparison of Amazon S3, Amazon EBS, and Amazon EFS

Table 3.10 is a useful in helping you to compare performance and storage characteristics for Amazon's highest-performing file, object, and block cloud storage offerings. This comparison will also be helpful when choosing the right data store for the applications that you are developing. It is also important for the exam.

**TABLE 3.10** Storage Service Comparison (EFS, S3, and EBS)

	File Amazon EFS	Object Amazon S3	Block Amazon EBS
<b>Performance</b>	Per-operation latency	Low, consistent	Low, for mixed request types, and integration with CloudFront
	Throughput scale	Multiple GB per second	Single GB per second
<b>Characteristics</b>	Data Availability/ Durability	Stored redundantly across multiple Availability Zones	Stored redundantly in a single Availability Zone
	Access	One to thousands of Amazon EC2 instances or on-premises servers, from multiple Availability Zones, concurrently	One to millions of connections over the web
	Use Cases	Web serving and content management, enterprise applications, media and entertainment, home directories, database backups, developer tools, container storage, Big Data analytics	Boot volumes, transactional and NoSQL databases, data warehousing, ETL

# Cloud Data Migration

Data is the cornerstone of successful cloud application deployments. Your evaluation and planning process may highlight the physical limitations inherent to migrating data from on-premises locations into the cloud. To assist you with that process, AWS offers a suite of tools to help you move data via networks, roads, and technology partners in and out of the cloud through offline, online, or streaming models.

The daunting realities of data transport apply to most projects: Knowing how to move to the cloud with minimal disruption, cost, and time, and knowing what is the most efficient way to move your data.

To determine the best-case scenario for efficiently moving your data, use this formula:

Number of Days =  $(\text{Total Bytes}) / (\text{Megabits per second} * 125 * 1000 * \text{Network Utilization} * 60 \text{ seconds} * 60 \text{ minutes} * 24 \text{ hours})$

For example, if you have a T1 connection (1.544 Mbps) and 1 TB ( $1024 \times 1024 \times 1024 \times 1024$  bytes) to move in or out of AWS, the theoretical minimum time that it would take to load over your network connection at 80 percent network utilization is 82 days.

Instead of using up bandwidth and taking a long time to migrate, many AWS customers are choosing one of the *data migration* options that are discussed next.



Multiple-choice questions that ask you to choose two or three true answers require that all of your answers be correct. There is no partial credit for getting a fraction correct. Pay extra attention to those questions when doing your review.

## AWS Storage Gateway

*AWS Storage Gateway* is a hybrid cloud storage service that enables your on-premises applications to use AWS cloud storage seamlessly. You can use this service for the following:

- Backup and archiving
- Disaster recovery
- Cloud bursting
- Storage tiering
- Migration

Your applications connect to the service through a gateway appliance using standard storage protocols, such as NFS and *internet Small Computer System Interface* (iSCSI). The gateway connects to AWS storage services, such as Amazon S3, Amazon S3 Glacier, and Amazon EBS, providing storage for files, volumes, and virtual tapes in AWS.

## File Gateway

A *file gateway* supports a file interface into Amazon S3, and it combines a cloud service with a virtual software appliance that is deployed into your on-premises environment as a VM. You can think of file gateway as an NFS mount on Amazon S3, allowing you to access your data directly in Amazon S3 from on premises as a file share.

## Volume Gateway

A *volume gateway* provides cloud-based storage volumes that you can mount as iSCSI devices from your on-premises application servers. A volume gateway supports cached mode and stored volume mode configurations.

Note that the volume gateway represents the family of gateways that support block-based volumes, previously referred to as *gateway-cached volumes* and *gateway-stored volumes*.

### Cached Mode

In the *cached volume mode*, your data is stored in Amazon S3, and a cache of the frequently accessed data is maintained locally by the gateway. This enables you to achieve cost savings on primary storage and minimize the need to scale your storage on premises while retaining low-latency access to your most used data.

### Stored Volume Mode

In the *stored volume mode*, data is stored on your local storage with volumes backed up asynchronously as Amazon EBS snapshots stored in Amazon S3. This provides durable off-site backups.

## Tape Gateway

A *tape gateway* can be used for backup to migrate off of physical tapes and onto a cost-effective and durable archive backup such as Amazon S3 Glacier. For a tape gateway, you store and archive your data on virtual tapes in AWS. A tape gateway eliminates some of the challenges associated with owning and operating an on-premises physical tape infrastructure. It can also be used for migrating data off of tapes, which are nearing end of life, into a more durable type of storage that still acts like tape.

## AWS Import/Export

*AWS Import/Export* accelerates moving large amounts of data into and out of the AWS Cloud using portable storage devices for transport. It transfers your data directly onto and off of storage devices using Amazon's high-speed internal network and bypassing the internet.

For significant datasets, AWS Import/Export is often faster than internet transfer and more cost-effective than upgrading your connectivity. You load your data onto your

devices and then create a job in the AWS Management Console to schedule shipping of your devices.

You are responsible for providing your own storage devices and the shipping charges to AWS.

It supports (in a limited number of regions) the following:

- Importing and exporting of data in Amazon S3 buckets
- Importing data into Amazon EBS snapshots

You cannot export directly from Amazon S3 Glacier. You must first restore your objects to Amazon S3 before exporting using AWS Import/Export.

## AWS Snowball

*AWS Snowball* is a petabyte-scale data transport solution that uses physical storage appliances, bypassing the internet, to transfer large amounts of data into and out of Amazon S3.

AWS Snowball addresses common challenges with large-scale data transfers, including the following:

- High network costs
- Long transfer times
- Security concerns

Figure 3.23 shows a physical AWS Snowball device.

**FIGURE 3.23** AWS Snowball



When you transfer your data with AWS Snowball, you do not need to write any code or purchase any hardware. To transfer data using AWS Snowball, perform the following steps:

1. Create a job in the AWS Management Console. The AWS Snowball appliance is shipped to you automatically.
2. When the appliance arrives, attach it to your local network.
3. Download and run the AWS Snowball client to establish a connection.
4. Use the client to select the file directories that you need to transfer to the appliance. The client will then encrypt and transfer the files to the appliance at high speed.
5. Once the transfer is complete and the appliance is ready to be returned, the E Ink shipping label automatically updates and you can track the job status via Amazon Simple Notification Service (Amazon SNS), text messages, or directly in the console.

Table 3.11 shows some common AWS Snowball use cases.

**TABLE 3.11** AWS Snowball Use Cases

Use Case	Description
Cloud migration	If you have large quantities of data that you need to migrate into AWS, AWS Snowball is often much faster and more cost-effective than transferring that data over the internet.
Disaster recovery	In the event that you need to retrieve a large quantity of data stored in Amazon S3 quickly, AWS Snowball appliances can help retrieve the data much quicker than high-speed internet.
Data center decommission	There are many steps involved in decommissioning a data center to make sure that valuable data is not lost. Snowball can help ensure that your data is securely and cost-effectively transferred to AWS during this process.
Content distribution	Use Snowball appliances if you regularly receive or need to share large amounts of data with clients, customers, or business associates. Snowball appliances can be sent directly from AWS to client or customer locations.

## AWS Snowball Edge

*AWS Snowball Edge* is a 100-TB data transfer service with on-board storage and compute power for select AWS capabilities. In addition to transferring data to AWS, AWS Snowball Edge can undertake local processing and edge computing workloads. Figure 3.24 shows a physical AWS Snowball Edge device.

**FIGURE 3.24** AWS Snowball Edge

Features of AWS Snowball Edge include the following:

- An endpoint on the device that is compatible with Amazon S3
- A file interface with NFS support
- A cluster mode where multiple AWS Snowball Edge devices can act as a single, scalable storage pool with increased durability
- The ability to run AWS Lambda powered by AWS IoT Greengrass functions as data is copied to the device
- Encryption taking place on the appliance itself

The transport of data is done by shipping the data in the appliances through a regional carrier. The appliance differs from the standard AWS Snowball because it can bring the power of the AWS Cloud to your local environment, with local storage and compute functionality.

There are three types of jobs that can be performed with Snowball Edge appliances:

- Import jobs into Amazon S3
- Export jobs from Amazon S3
- Local compute and storage-only jobs

Use AWS Snowball Edge when you need the following:

- Local storage and compute in an environment that might or might not have an internet connection
- To transfer large amounts of data into and out of Amazon S3, bypassing the internet

Table 3.12 shows the different use cases for the different AWS Snowball devices.

**TABLE 3.12** AWS Snowball Device Use Cases

Use Case	AWS Snowball	AWS Snowball Edge
Import data into Amazon S3	✓	✓
Copy data directly from HDFS	✓	
Export from Amazon S3	✓	✓
Durable local storage		✓
Use in a cluster of devices		✓
Use with AWS IoT Greengrass		✓
Transfer files through NFS with a GUI		✓

## AWS Snowmobile

*AWS Snowmobile* is an exabyte-scale data transfer service used to move extremely large amounts of data from on premises to AWS. You can transfer up to 100 PB per AWS Snowmobile, a 45-foot long ruggedized shipping container pulled by a semi-trailer truck.

AWS Snowmobile makes it easy to move massive volumes of data to the cloud, including video libraries, image repositories, or even a complete data center migration. In 2017, one AWS customer moved 8,700 tapes with 54 million files to Amazon S3 using AWS Snowmobile. Figure 3.25 shows an AWS Snowmobile shipping container being pulled by a semi-trailer truck.

**FIGURE 3.25** AWS Snowmobile



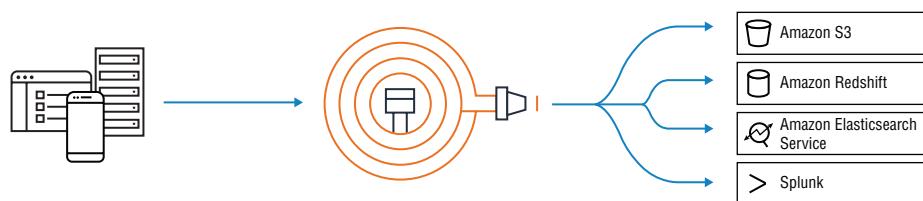
How do you choose between AWS Snowmobile and AWS Snowball? To migrate large datasets of 10 PB or more in a single location, you should use AWS Snowmobile. For datasets that are less than 10 PB or distributed in multiple locations, you should use AWS Snowball.

## Amazon Kinesis Data Firehose

*Amazon Kinesis Data Firehose* lets you prepare and load real-time data streams into data stores and analytics tools. Although it has much broader uses for loading data continuously for data streaming and analytics, it can be used as a one-time tool for data migration into the cloud.

Amazon Kinesis Data Firehose can capture, transform, and load streaming data into Amazon S3 and Amazon Redshift, which will be discussed further in Chapter 4, “Hello, Databases.” With Amazon Kinesis Data Firehose, you can avoid writing applications or managing resources. When you configure your data producers to send data to Amazon Kinesis Data Firehose, as shown in Figure 3.26, it automatically delivers the data to the destination that you specified. This is an efficient option to transform and deliver data from on premises to the cloud.

**FIGURE 3.26** Amazon Kinesis Data Firehose



Input	Amazon Kinesis Data Firehose	Data stores
Capture and send data to Kinesis Data Firehose	Prepares and loads the data continuously to the destinations you choose	Durably store the data for analytics

Destinations include the following:

- Amazon S3
- Amazon Redshift
- Amazon Elasticsearch Service
- Splunk

## Key Concepts

As you get started with Amazon Kinesis Data Firehose, you will benefit from understanding the concepts described next.

## Kinesis Data Delivery Stream

You use Amazon Kinesis Data Firehose by creating an Amazon Kinesis *data delivery stream* and then sending data to it.

### Record

A *record* is the data that your producer sends to a Kinesis data delivery stream, with a maximum size of 1,000 KB.

### Data Producer

*Data producers* send records to Amazon Kinesis data delivery streams. For example, your web server could be configured as a data producer that sends log data to an Amazon Kinesis delivery stream.

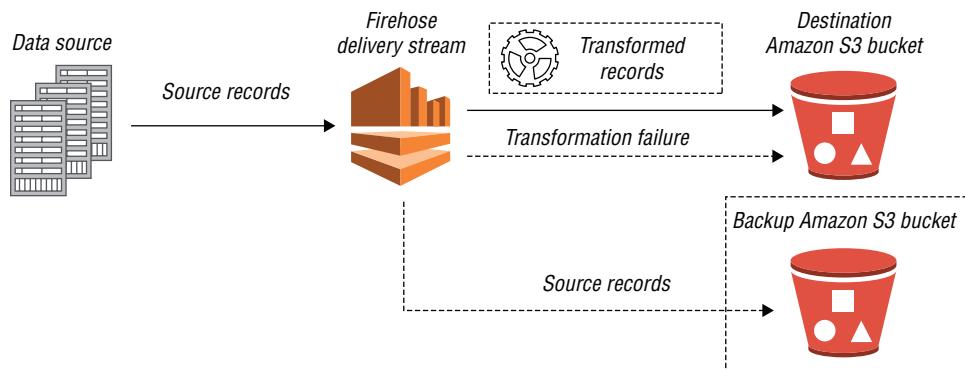
### Buffer Size and Buffer Interval

Amazon Kinesis Data Firehose buffers incoming data to a certain size or for a certain period of time before delivering it to destinations. Buffer size is in megabytes, and buffer interval is in seconds.

## Data Flow

You can stream data to your Amazon S3 bucket, as shown in Figure 3.27. If data transformation is enabled, you can optionally back up source data to another Amazon S3 bucket.

**FIGURE 3.27** Streaming to Amazon S3



## AWS Direct Connect

Using *AWS Direct Connect*, you can establish private connectivity between AWS and your data center, office, or colocation environment, which in many cases can reduce your network costs, increase bandwidth throughput, and provide a more consistent network experience than internet-based connections.

These benefits can then be applied to storage migration. Transferring large datasets over the internet can be time-consuming and expensive. When you use the cloud, you may find that transferring large datasets can be slow because your business-critical network traffic is contending for bandwidth with your other internet usage. To decrease the amount of time required to transfer your data, increase the bandwidth to your internet service provider. Be aware that this frequently requires a costly contract renewal and minimum commitment.

More details on using AWS Direct Connect will be provided in Chapter 4.

## VPN Connection

You can connect your Amazon VPC to remote networks by using a VPN connection. Table 3.13 shows some of the connectivity options available to you.

**TABLE 3.13** Amazon VPC Connectivity Options

VPN Connectivity Option	Description
AWS managed VPN	Create an IP Security (IPsec) VPN connection between your VPC and your remote network. On the AWS side of the VPN connection, a virtual private gateway provides two VPN endpoints (tunnels) for automatic failover. You configure your customer gateway on the remote side of the VPN connection.
AWS VPN CloudHub	If you have more than one remote network (for example, multiple branch offices), create multiple AWS managed VPN connections via your virtual private gateway to enable communication between these networks.
Third-party software VPN appliance	Create a VPN connection to your remote network by using an Amazon EC2 instance in your VPC that's running a third-party software VPN appliance. AWS does not provide or maintain third-party software VPN appliances; however, you can choose from a range of products provided by partners and open-source communities.

You can also use AWS Direct Connect to create a dedicated private connection from a remote network to your VPC. You can combine this connection with an AWS managed VPN connection to create an IPsec-encrypted connection.

You will learn more about VPN connections in subsequent chapters.

# Summary

AWS cloud computing provides a reliable, scalable, and secure place for your data. Cloud storage is a critical component of cloud computing, holding the information used by applications. Big Data analytics, data warehouses, Internet of Things, databases, and backup and archive applications all rely on some form of data storage architecture. Cloud storage is typically more reliable, scalable, and secure than traditional on-premises storage systems.

AWS offers a complete range of cloud storage services to support both application and archival compliance requirements. You may choose from object, file, and block storage services and cloud data migration options to start designing the foundation of your cloud IT environment.

Amazon EBS provides highly available, consistent, low-latency, persistent local block storage for Amazon EC2. It helps you to tune applications with the right storage capacity, performance, and cost.

Amazon EFS provides a simple, scalable file system interface and file system access semantics to make data available to one or more EC2 instances as block storage. Amazon EFS grows and shrinks capacity automatically, and it provides high throughput with consistent low latencies. Amazon EFS is designed for high availability and durability, and it provides performance for a broad spectrum of workloads and applications.

Amazon S3 is a form of object storage that provides a scalable, durable platform to make data accessible from any internet location, and it allows you to store and access any type of data over the internet. Amazon S3 is secure, 99.99999999 percent durable, and scales past tens of trillions of objects.

Amazon S3 Glacier provides extremely low-cost and highly durable object storage for long-term backup and archiving of any type of data. Amazon S3 Glacier is a solution for customers who want low-cost storage for infrequently accessed data. It can replace tape, and assist with compliance in highly regulated organizations.

Amazon offers a full portfolio of cloud data migration services to help simplify and accelerate moving data of all types and sizes into and out of the AWS Cloud. These include AWS Storage Gateway, AWS Import/Export Disk, AWS Snowball, AWS Snowball Edge, AWS Snowmobile, Amazon Kinesis Data Firehose, AWS Direct Connect, and a VPN connection.

Understanding when to use the right tool for your data storage and data migration options is a key component of the exam, including data dimension, block versus object versus file storage, data structure, and storage temperature. Be ready to compare and contrast the durability, availability, latency, means of access, and cost of different storage options for a given use case.

# Exam Essentials

**Know the different data dimensions.** Consider the different data dimensions when choosing which storage option and storage class will be most appropriate for your data. This includes velocity, variety, volume, storage temperature (hot, warm, cold, frozen), data value, transient, reproducible, authoritative, and critical/regulated data.

**Know the difference between block, object, and file storage.** Block storage is commonly dedicated, low-latency storage for each host and is provisioned with each instance. Object storage is developed for the cloud, has vast scalability, is accessed over the Web, and is not directly attached to an instance. File storage enables accessing shared files as a file system.

**Know the AWS shared responsibility model and how it applies to storage.** AWS is responsible for securing the storage services. You are responsible for securing access to the artifacts that you create or objects that you store.

**Know what Amazon EBS is and for what it is commonly used.** Amazon EBS provides persistent block storage volumes for use with Amazon EC2 instances. It is designed for application workloads that benefit from fine tuning for performance, cost, and capacity. Typical use cases include Big Data analytics engines, relational and NoSQL databases, stream and log processing applications, and data warehousing applications. Amazon EBS volumes also serve as root volumes for Amazon EC2 instances.

**Know what Amazon EC2 instance store is and what it is commonly used for.** An instance store provides temporary block-level storage for your instance. It is located on disks that are physically attached to the host computer. Instance store is ideal for temporary storage of information that changes frequently, such as buffers, caches, scratch data, and other temporary content, or for data that is replicated across a fleet of instances, such as a load-balanced pool of web servers. In some cases, you can use instance-store backed volumes. Do not use instance store (also known as ephemeral storage) for either production data or data that must be kept durable.

**Know what Amazon S3 is and what it is commonly used for.** Amazon S3 is object storage built to store and retrieve any amount of data from anywhere. It is secure, durable, and highly scalable cloud storage using a simple web services interface. Amazon S3 is commonly used for backup and archiving, content storage and distribution, Big Data analytics, static website hosting, cloud-native application hosting, and disaster recovery, and as a data lake.

**Know the basic concepts of Amazon S3.** Amazon S3 stores data as objects within resources called *buckets*. You can store as many objects as desired within a bucket, and write, read, and delete objects in your bucket. Objects contain data and metadata and are identified by a user-defined key in a flat file structure. Interfaces to Amazon S3 include a native REST interface, SDKs for many languages, the AWS CLI, and the AWS Management Console.

**Know how to create a bucket, how to upload, download, and delete objects, how to make objects public, and how to open an object URL.**

**Understand how security works in Amazon S3.** By default, new buckets are private and nothing is publicly accessible. When you add an object to a bucket, it is private by default.

**Know how much data you can store in Amazon S3.** The total volume of data and number of objects that you can store in Amazon S3 are unlimited. Individual Amazon S3 objects can range in size from a minimum of 0 bytes to a maximum of 5 TB. The largest object

that can be uploaded in a single PUT is 5 GB. For objects larger than 100 MB, consider using the Multipart Upload capability.

**Know the Amazon S3 service limit for buckets per account.** One hundred buckets are allowed per account.

**Understand the durability and availability of Amazon S3.** Amazon S3 standard storage is designed for 11 nines of durability and four nines of availability of objects over a given year. Other storage classes differ. Reduced Redundancy Storage (RRS) storage class is less durable than Standard, and it is intended for noncritical, reproducible data.

**Know the data consistency model of Amazon S3.** Amazon S3 is eventually consistent, but it offers read-after-write consistency (read after write consistency for PUT of new objects and eventual consistency for overwrite PUT and DELETE).

**Know the Amazon S3 storage classes and use cases for each.** Standard is used to store general-purpose data that needs high durability, high performance, and low latency access. Standard\_IA is used for data that is less frequently accessed but that needs the same performance and availability when accessed. OneZone\_IA is similar to Standard\_IA, but it is stored only in a single Availability Zone, costing 20 percent less. However, data stored with OneZone\_IA will be permanently lost in the event of an Availability Zone destruction. Reduced\_Redundancy offers lower durability at lower cost for easily-reproducible data. Amazon S3 Glacier is used to store rarely accessed archival data at an extremely low cost, when three- to five-hour retrieval time is acceptable under the standard retrieval option. There are other retrieval options for higher and lower cost at shorter and longer retrieval times, including expedited retrieval (on-demand or provisioned, 1–5 minutes) and bulk retrieval (5–12 hours).

Every object within a bucket can be designated to a different storage class.

**Know how to enable static web hosting on Amazon S3.** The steps to enable static web hosting on Amazon S3 require you to do the following:

- Create a bucket with the website hostname.
- Upload your static content and make it public.
- Enable static website hosting on the bucket.
- Indicate the index and error page objects.

**Know how to encrypt your data on Amazon S3.** For server-side encryption, use SSE-SE (Amazon S3 Managed Keys), SSE-C (Customer-Provided Keys), and SSE-KMS (KMS-Managed Keys). For client-side encryption, choose from a client-side master key or an AWS KMS managed customer master key.

**Know how to protect your data on Amazon S3.** Know the different options for protecting your data in flight and in transit. Encrypt data in flight using HTTPS and at rest using server-side or client-side encryption. Enable versioning to keep multiple versions of an object in a bucket. Enable MFA Delete to protect against accidental deletion. Use ACLs,

Amazon S3 bucket policies, and IAM policies for access control. Use presigned URLs for time-limited download access. Use cross-region replication to replicate data to another region automatically.

**Know how to use lifecycle configuration rules.** Lifecycle rules can be used to manage your objects so that they are stored cost-effectively throughout their lifecycle. There are two types of actions. Transition actions define when an object transitions to another storage class. Expiration actions define when objects expire and will be deleted on your behalf.

**Know what Amazon EFS is and what it is commonly used for.** Amazon EFS provides simple, scalable, elastic file storage for use with AWS services and on-premises resources. Amazon EFS is easy to use and offers a simple interface that allows you to create and configure file systems quickly and easily. Amazon EFS is built to scale elastically on demand without disrupting applications, growing and shrinking automatically as you add and remove files, so your applications have the storage that they need, when they need it. Amazon EFS is designed for high availability and durability. Amazon EFS can be mounted to multiple Amazon EC2 instances at the same time.

**Know the basics of Amazon S3 Glacier as a stand-alone service.** Data is stored in encrypted archives that can be as large as 40 TB. Archives typically contain TAR and ZIP files. Vaults are containers for archives, and vaults can be locked for compliance.

**Know which storage option to choose based on storage temperature.** For hot to warm storage, use Amazon EC2 instance store, Amazon EBS, or Amazon S3. For cold storage, choose Amazon S3 Glacier.

**Know which storage option to choose based on latency.** Amazon EC2 instance store and Amazon EBS are designed for millisecond latency. Amazon S3 depends on size, anywhere from milliseconds to seconds to minutes. Amazon S3 Glacier is minutes to hours depending on retrieval option.

**Know which storage option to choose based on data volume.** Amazon EC2 instance store can be from 4 GB to 48 TB. Amazon EBS can be from 1 GiB to 16 TiB. Amazon S3 and Amazon S3 Glacier have no limit.

**Know which storage option to choose based on item size.** Amazon EC2 instance store and Amazon EBS depend on the size of the block storage and operating system limits. Amazon S3 has a 5 TB max size per object, but objects may be split. Amazon S3 Glacier has a 40 TB maximum.

**Know when you should use Amazon EBS, Amazon EFS, Amazon S3, Amazon S3 Glacier, or AWS Storage Gateway for your data.** For persistent local storage for Amazon EC2, use Amazon EBS. For a file system interface and file system access semantics to make data available to one or more Amazon EC2 instances, use Amazon EFS. For a scalable, durable platform to make data accessible from any internet location, use Amazon S3. For highly affordable, long-term cold storage, use Amazon S3 Glacier. For a hybrid storage cloud augmenting your on-premises environment with Amazon cloud storage, use AWS Storage Gateway.

**Know when to choose Amazon EBS or Amazon EC2 instance store.** Amazon EBS is most often the default option. However, Amazon EC2 instance store may be an option if your data does not meet any of the following criteria:

- Must persist through instance stops, terminations, or hardware failures
- Needs to be encrypted at the full volume level
- Needs to be backed up with EBS snapshots
- Needs to be removed from one instance and reattached to another

**Know the different cloud data migration options.** There are a number of options for migrating your data to the AWS Cloud, or having a hybrid data solution between AWS and your data center or on premises. These include (but are not limited to) AWS Storage Gateway, AWS Import/Export, AWS Snowball, AWS Snowball Edge, AWS Snowmobile, Amazon Kinesis Data Firehose, AWS Direct Connect, and AWS VPN connections. Know when to choose one over the other based on time, cost, or volume.

**Know what AWS Storage Gateway is and how it is used for cloud data migration.** AWS Storage Gateway is a hybrid cloud storage service that enables your on-premises applications to use AWS cloud storage seamlessly. Use this for data migration by means of a gateway that connects to AWS storage services, such as Amazon S3, Amazon S3 Glacier, and Amazon EBS.

**Know what AWS Import/Export Disk is and how it is used for cloud data migration.** AWS Import/Export Disk accelerates moving large amounts of data into and out of the AWS Cloud using portable storage devices for transport. It transfers your data directly onto and off of storage devices using Amazon's high-speed internal network and bypassing the internet. For significant data sets, it is often much faster than transferring the data via the internet. You provide the hardware.

**Know what AWS Snowball is and how it is used for cloud data migration.** Snowball is a petabyte-scale data transport solution that uses devices designed to be secure to transfer large amounts of data into and out of the AWS Cloud. Using Snowball addresses common challenges with large-scale data transfers including high network costs, long transfer times, and security concerns. You can transfer data at as little as one-fifth the cost of transferring data via high-speed internet. AWS provides the hardware.

**Know what AWS Snowball Edge is and how it is used for cloud data migration.** AWS Snowball Edge is a 100-TB data transfer device with on-board storage and compute capabilities. Use it to move large amounts of data into and out of AWS, as a temporary storage tier for large local datasets, or to support local workloads in remote or offline locations. AWS Snowball Edge is a fast and inexpensive way to transfer large amounts of data when migrating to AWS.

**Know what AWS Snowmobile is and how it is used for cloud data migration.** AWS Snowmobile is an exabyte-scale data transfer service used to move extremely large amounts of data to AWS. You can transfer up to 100 PB per Snowmobile, a 45-foot long ruggedized shipping container pulled by a semi-trailer truck. Snowmobile makes it easy to move massive volumes of data to the cloud, even a complete data center migration.

**Know what Amazon Kinesis Data Firehose is and how it is used for cloud data migration.** Amazon Kinesis Data Firehose is the easiest way to load streaming data reliably into data stores and analytics tools. It can capture, transform, and load streaming data into Amazon S3, Amazon Redshift, Amazon Elasticsearch Service, and Splunk. Kinesis Data Firehose can be used to transform and migrate data from on premises into the cloud.

**Know what AWS Direct Connect is and how it is used for cloud data migration.** Use AWS Direct Connect to establish private connectivity between AWS and your data center, office, or colocation environment, which in many cases can reduce your network costs, increase bandwidth throughput, and provide a more consistent network experience than internet-based connections.

**Know what a VPN connection is and how it is used for cloud data migration.** Connect your Amazon VPC to remote networks by using a VPN connection to increase privacy while migrating your data.

**Know which tool to use for migrating storage to the AWS Cloud based on data size, timeline, and cost.** There are two ways to migrate data: online and offline.

**Online** Use AWS Direct Connect to connect your data center privately and directly to an AWS Region. Use AWS Snowball to transport petabytes of data physically in batches to the cloud. Use Snowball Edge to build hybrid storage that preserves existing on-premises investment and adds AWS services. Use AWS Snowmobile to migrate exabytes of data in batches to the cloud. Use Amazon S3 Transfer Acceleration to work with Amazon S3 over long geographic distances.

**Offline** Use AWS Storage Gateway to integrate existing on-premises resources with the cloud. Use AWS Snowball Edge to transport petabytes of data physically in an appliance with on-board storage and compute capabilities. Use Amazon Kinesis Data Firehose to collect and ingest multiple streaming data sources or perform ETL on data while migrating to the AWS Cloud.

## Resources to Review

Cloud Storage with AWS:

<https://aws.amazon.com/products/storage/>

AWS Storage Optimization (Whitepaper):

<https://docs.aws.amazon.com/aws-technical-content/latest/cost-optimization-storage-optimization/cost-optimization-storage-optimization.pdf>

AWS Storage Services Overview (Whitepaper):

<https://aws.amazon.com/whitepapers/storage-options-aws-cloud/>

Overview of AWS Security—Storage Services (Whitepaper):

[https://d1.awsstatic.com/whitepapers/Security/Security\\_Storage\\_Services\\_Whitepaper.pdf](https://d1.awsstatic.com/whitepapers/Security/Security_Storage_Services_Whitepaper.pdf)

Writing IAM Policies—How to Grant Access to an Amazon S3 Bucket  
(AWS Security Blog):

<https://aws.amazon.com/blogs/security/writing-iam-policies-how-to-grant-access-to-an-amazon-s3-bucket/>

Leveraging the Breadth of Storage Services and the Ecosystem at AWS—Unlock the Full Potential of Public Cloud IaaS:

<https://d0.awsstatic.com/analyst-reports/US41693416.pdf>

Cloud Data Migration Services:

<https://aws.amazon.com/cloud-data-migration/>

AWS Migration (Whitepaper):

<https://d1.awsstatic.com/whitepapers/Migration/aws-migration-whitepaper.pdf>

AWS Storage Gateway (Whitepaper):

<https://d1.awsstatic.com/whitepapers/aws-storage-gateway-file-gateway-for-hybrid-architectures.pdf>

Hosting Static Websites on AWS (Whitepaper):

<https://d1.awsstatic.com/whitepapers/Building%20Static%20Websites%20on%20AWS.pdf>

Encrypting Data at Rest (Whitepaper):

[https://d0.awsstatic.com/whitepapers/AWS\\_Securing\\_Data\\_at\\_Rest\\_with\\_Encryption.pdf](https://d0.awsstatic.com/whitepapers/AWS_Securing_Data_at_Rest_with_Encryption.pdf)

Building Big Data Storage Solutions (Data Lakes) for Maximum Flexibility (Whitepaper):

<https://docs.aws.amazon.com/aws-technical-content/latest/building-data-lakes/building-data-lakes-on-aws.pdf>

What Is Cloud Object Storage?

<https://aws.amazon.com/what-is-cloud-object-storage/>

Amazon Simple Storage Service—Getting Started Guide:

<http://docs.aws.amazon.com/AmazonS3/latest/gsg/>

Amazon Simple Storage Service—Developer Guide:

<https://docs.aws.amazon.com/AmazonS3/latest/dev>Welcome.html>

VPC Endpoints:

<https://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/vpc-endpoints.html>

Amazon S3 Glacier—Developer Guide:

<https://docs.aws.amazon.com/amazonglacier/latest/dev/introduction.html>

Amazon Elastic File System—User Guide:

<https://docs.aws.amazon.com/efs/latest/ug/>

Deep Dive on Elastic File System—2017 AWS Online Tech Talks (Video):

<https://youtu.be/NhI0g8vI5M0>

AWS re:Invent 2017: Deep Dive on Amazon Elastic File System (Amazon EFS) (STG307) (Video):

<https://www.youtube.com/watch?v=VffbHp34UzQ>

Amazon Elastic Block Store—Linux:

<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AmazonEBS.html>

Amazon Elastic Block Store—Windows:

<https://docs.aws.amazon.com/AWSEC2/latest/WindowsGuide/AmazonEBS.html>

Amazon EC2 Instance Storage:

<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/InstanceStorage.html>

AWS Storage Gateway—User Guide:

<https://docs.aws.amazon.com/storagegateway/latest/userguide/WhatIsStorageGateway.html>

AWS Import/Export Disk:

<https://aws.amazon.com/snowball/disk/>

AWS Snowball—User Guide:

<http://docs.aws.amazon.com/snowball/latest/ug/whatissnowball.html>

AWS Snowball Edge—Developer Guide:

<http://docs.aws.amazon.com/snowball/latest/developer-guide/whatisedge.html>

AWS Snowmobile:

<https://aws.amazon.com/snowmobile/>

Amazon Kinesis Data Firehose—Developer Guide:

<https://docs.aws.amazon.com/firehose/latest/dev/what-is-this-service.html>

AWS Direct Connect—User Guide:

<http://docs.aws.amazon.com/directconnect/latest/UserGuide/>

VPN Connections:

<https://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/vpn-connections.html>

How to Use Bucket Policies and Apply Defense-in-Depth to Help Secure Your Amazon S3 Data:

<https://aws.amazon.com/blogs/security/how-to-use-bucket-policies-and-apply-defense-in-depth-to-help-secure-your-amazon-s3-data/>

IAM Policies and Bucket Policies and ACLs! Oh, My! (Controlling Access to S3 Resources):

<https://aws.amazon.com/blogs/security/iam-policies-and-bucket-policies-and-acls-oh-my-controlling-access-to-s3-resources/>

AWS re:Invent Storage State of the Union (Video):

<https://www.youtube.com/watch?v=U-flt95opTw>

AWS re:Invent Best Practices for Amazon S3 (STG302) (Video):

<https://www.youtube.com/watch?v=UKuL1K3oWuo>

What Is a Data Lake?

<https://aws.amazon.com/big-data/data-lake-on-aws/>

Amazon S3 Service Level Agreement:

<https://aws.amazon.com/s3/sla/>

Protecting Data Using Server-Side Encryption with Amazon S3-Managed Encryption Keys (SSE-S3):

<https://docs.aws.amazon.com/AmazonS3/latest/dev/serv-side-encryption.html>

Picking the Right Data Store for Your Workload:

<https://aws.amazon.com/blogs/startups/picking-the-right-data-store-for-your-workload/>

Demystifying Storage on AWS (Video):

<https://www.youtube.com/watch?v=6UWmN2RbsnY>

## Exercises

For assistance in completing the following exercises, refer to the Amazon Simple Storage Service Developer Guide:

<https://docs.aws.amazon.com/AmazonS3/latest/dev>Welcome.html>

We assume that you have performed the Exercises in Chapter 1 and Chapter 2 to set up your development environment in AWS Cloud9, or have done so on your own system with the AWS SDK.

For instructions on creating and testing a working sample, see Testing the Amazon S3 Java Code Examples here:

<https://docs.aws.amazon.com/AmazonS3/latest/dev/UsingTheMPJavaAPI.html#TestingJavaSamples>

**EXERCISE 3.1****Create an Amazon Simple Storage Service (Amazon S3) Bucket**

In this exercise, you will create an Amazon S3 bucket using the AWS SDK for Java. You will use this bucket in the exercises that follow.

For assistance in completing this exercise, copying this code, or for code in other languages, see the following documentation:

<https://docs.aws.amazon.com/AmazonS3/latest/dev/create-bucket-get-location-example.html>

1. Enter the following code in your preferred development environment for Java:

```
import java.io.IOException;

import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.CreateBucketRequest;
import com.amazonaws.services.s3.model.GetBucketLocationRequest;

public class CreateBucket {

 public static void main(String[] args) throws IOException {
 String clientRegion = "*** Client region ***";
 String bucketName = "*** Bucket name ***";

 try {
 AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
 .withCredentials(new ProfileCredentialsProvider())
 .withRegion(clientRegion)
 .build();

 if (!s3Client.doesBucketExistV2(bucketName)) {
 // Because the CreateBucketRequest object doesn't specify a
 // region, the
 // bucket is created in the region specified in the client.
 s3Client.createBucket(new CreateBucketRequest(bucketName));
 }
 } catch (AmazonServiceException e) {
 System.out.println("Caught an AmazonServiceException, which " +
 "means your request made it to Amazon S3, " +
 "but was rejected with an error response for some reason." +
 "Status code: " + e.getStatusCode() + ", " +
 "Error code: " + e.getErrorCode());
 } catch (SdkClientException e) {
 System.out.println("Caught an SdkClientException, which means " +
 "something happened in the client while " +
 "communicating with Amazon S3." +
 "Message: " + e.getMessage());
 }
 }
}
```

---

(continued)

**EXERCISE 3.1 (continued)**

```
// Verify that the bucket was created by retrieving it and
// checking its location.
String bucketLocation = s3Client.getBucketLocation(new
GetBucketLocationRequest(bucketName));
System.out.println("Bucket location: " + bucketLocation);
}
}
catch(AmazonServiceException e) {
 // The call was transmitted successfully, but Amazon S3 couldn't
 // process
 // it and returned an error response.
 e.printStackTrace();
}
catch(SdkClientException e) {
 // Amazon S3 couldn't be contacted for a response, or the client
 // couldn't parse the response from Amazon S3.
 e.printStackTrace();
}
}
}
```

2. Replace the static variable values for `clientRegion` and `bucketName`. Note that bucket names must be unique across all of AWS. Make a note of these two values; you will use the same region and bucket name for the exercises that follow in this chapter.
3. Execute the code. Your bucket gets created with the name you specified in the region you specified. A successful result shows the following output:

```
Bucket Location: [bucketLocation]
```

**EXERCISE 3.2****Upload an Object to a Bucket**

Now that you have a bucket, you can add objects to it. In this example, you will create two objects. The first object has a text string as data, and the second object is a file. This example creates the first object by specifying the bucket name, object key, and text data directly in a call to `AmazonS3Client.putObject()`. The example creates a second object by using a `PutObjectRequest` that specifies the bucket name, object key, and file path. The `PutObjectRequest` also specifies the `ContentType` header and title metadata.

---

For assistance in completing this exercise, copying this code, or for code in other languages, see the following documentation:

<https://docs.aws.amazon.com/AmazonS3/latest/dev/UploadObjSingleOpJava.html>

1. Enter the following code in your preferred development environment for Java:

```
import java.io.File;
import java.io.IOException;

import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.ObjectMetadata;
import com.amazonaws.services.s3.model.PutObjectRequest;

public class UploadObject {

 public static void main(String[] args) throws IOException {
 String clientRegion = "*** Client region ***";
 String bucketName = "*** Bucket name ***";
 String stringObjKeyName = "*** String object key name ***";
 String fileObjKeyName = "*** File object key name ***";
 String fileName = "*** Path to file to upload ***";

 try {
 AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
 .withRegion(clientRegion)
 .withCredentials(new ProfileCredentialsProvider())
 .build();

 // Upload a text string as a new object.
 s3Client.putObject(bucketName, stringObjKeyName, "Uploaded
String Object");

 // Upload a file as a new object with ContentType and title
 specified.
 }
 }
}
```

---

(continued)

**EXERCISE 3.2 (continued)**

```
 PutObjectRequest request = new PutObjectRequest(bucketName,
 fileObjKeyName, new File(fileName));
 ObjectMetadata metadata = new ObjectMetadata();
 metadata.setContentType("plain/text");
 metadata.addUserMetadata("x-amz-meta-title", "someTitle");
 request.setMetadata(metadata);
 s3Client.putObject(request);
 }
 catch(AmazonServiceException e) {
 // The call was transmitted successfully, but Amazon S3 couldn't
 // process
 // it, so it returned an error response.
 e.printStackTrace();
 }
 catch(SdkClientException e) {
 // Amazon S3 couldn't be contacted for a response, or the client
 // couldn't parse the response from Amazon S3.
 e.printStackTrace();
 }
}
}
```

2. Replace the static variable values for clientRegion and bucketName used in the previous exercise.
3. Replace the value for stringObjKeyName with the name of the key that you intend to create in your Amazon S3 bucket, which will upload a text string as a new object.
4. Replace the Uploaded String Object text with the text being placed inside the object that you are generating.
5. Replace the someTitle text in the code with your own metadata title for the object that you are uploading.
6. Create a local file on your machine and then replace the value for fileName with the full path and filename of the file that you created.
7. Replace the fileObjKeyName with the key name that you want for the file that you will be uploading. A file can be uploaded with a different name than the filename that's used locally.
8. Execute the code. Your bucket gets created with the name that you specified in the region that you specified. A successful result without errors will create two objects in your bucket.

**EXERCISE 3.3****Emptying and Deleting a Bucket**

Now that you have finished with the Amazon S3 exercises, you will want to clean up your environment by deleting all the files and the bucket you created. It is easy to delete an empty bucket. However, in some situations, you may need to delete or empty a bucket that contains objects. In this exercise, we show you how to delete objects and then delete the bucket.

For assistance in completing this exercise, copying this code, or for code in other languages, see the following documentation:

<https://docs.aws.amazon.com/AmazonS3/latest/dev/delete-or-empty-bucket.html>

<https://docs.aws.amazon.com/AmazonS3/latest/dev/delete-or-empty-bucket.html#delete-bucket-sdk-java>

1. Enter the following code in your preferred development environment for Java:

```
import java.util.Iterator;

import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.ListVersionsRequest;
import com.amazonaws.services.s3.model.ObjectListing;
import com.amazonaws.services.s3.model.S3ObjectSummary;
import com.amazonaws.services.s3.model.S3VersionSummary;
import com.amazonaws.services.s3.model.VersionListing;

public class DeleteBucket {

 public static void main(String[] args) {
 String clientRegion = "*** Client region ***";
 String bucketName = "*** Bucket name ***";

 try {
 AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
 .withCredentials(new ProfileCredentialsProvider())
 .withRegion(clientRegion)
 .build();
 }
 }
}
```

---

(continued)

**EXERCISE 3.3 (continued)**

```
// Delete all objects from the bucket. This is sufficient
// for unversioned buckets. For versioned buckets, when you
attempt to delete objects, Amazon S3 inserts
// delete markers for all objects, but doesn't delete the object
versions.
// To delete objects from versioned buckets, delete all of the
object versions before deleting
// the bucket (see below for an example).
ObjectListing objectListing = s3Client.listObjects(bucketName);
while (true) {
 Iterator<S3ObjectSummary> objIter = objectListing.
 getObjectSummaries().iterator();
 while (objIter.hasNext()) {
 s3Client.deleteObject(bucketName, objIter.next().
 getKey());
 }

 // If the bucket contains many objects, the listObjects()
 // call
 // might not return all of the objects in the first listing.
 // Check to
 // see whether the listing was truncated. If so, retrieve
 // the next page of objects
 // and delete them.
 if (objectListing.isTruncated()) {
 objectListing = s3Client.listNextBatchOfObjects
 (objectListing);
 } else {
 break;
 }
}

// Delete all object versions (required for versioned buckets).
VersionListing versionList = s3Client.listVersions(new
ListVersionsRequest().withBucketName(bucketName));
while (true) {
 Iterator<S3VersionSummary> versionIter = versionList.
 getVersionSummaries().iterator();
 while (versionIter.hasNext()) {
 S3VersionSummary vs = versionIter.next();
 s3Client.deleteVersion(bucketName, vs.getKey(),
 vs.getVersionId());
 }
}
```

```
 if (versionList.isTruncated()) {
 versionList = s3Client.listNextBatchOfVersions
 (versionList);
 } else {
 break;
 }
 }

// After all objects and object versions are deleted, delete the
bucket.
s3Client.deleteBucket(bucketName);
}

catch(AmazonServiceException e) {
 // The call was transmitted successfully, but Amazon S3 couldn't
process
 // it, so it returned an error response.
 e.printStackTrace();
}

catch(SdkClientException e) {
 // Amazon S3 couldn't be contacted for a response, or the client
couldn't
 // parse the response from Amazon S3.
 e.printStackTrace();
}
}
```

- 
2. Replace the static variable values for clientRegion and bucketName with the values that you used in the previous steps.
  3. Execute the code.
  4. When execution is complete without errors, both of your objects and your bucket will have been deleted.

## Review Questions

1. You are developing an application that will run across dozens of instances. It uses some components from a legacy application that requires some configuration files to be copied from a central location and be held on a volume local to each of the instances. You plan to modify your application with a new component in the future that will hold this configuration in Amazon DynamoDB. However, in the interim, which storage option should you use that will provide the lowest cost and the lowest latency for your application to access the configuration files?
  - A. Amazon S3
  - B. Amazon EBS
  - C. Amazon EFS
  - D. Amazon EC2 instance store
2. In what ways does Amazon Simple Storage Service (Amazon S3) object storage differ from block and file storage? (Select TWO.)
  - A. Amazon S3 stores data in fixed size blocks.
  - B. Objects are identified by a numbered address.
  - C. Object can be any size.
  - D. Objects contain both data and metadata.
  - E. Objects are stored in buckets.
3. You are restoring an Amazon Elastic Block Store (Amazon EBS) volume from a snapshot. How long will it take before the data is available?
  - A. It depends on the provisioned size of the volume.
  - B. The data will be available immediately.
  - C. It depends on the amount of data stored on the volume.
  - D. It depends on whether the attached instance is an Amazon EBS-optimized instance.
4. What are some of the key characteristics of Amazon Simple Storage Service (Amazon S3)? (Select THREE.)
  - A. All objects have a URL.
  - B. Amazon S3 can store unlimited amounts of data.
  - C. Buckets can be mounted to the file system of multiple Amazon EC2 instances.
  - D. Amazon S3 uses a Representational State Transfer (REST) application program interface (API).
  - E. You must pre-allocate the storage in a bucket.

5. Amazon S3 Glacier is well-suited to data that is which of the following? (Select TWO.)
  - A. Infrequently or rarely accessed
  - B. Must be immediately available when needed
  - C. Is available after a three- to five-hour restore period
  - D. Is frequently erased within 30 days
6. You have valuable media files hosted on AWS and want them to be served only to authenticated users of your web application. You are concerned that your content could be stolen and distributed for free. How can you protect your content?
  - A. Use static web hosting.
  - B. Generate presigned URLs for content in the web application.
  - C. Use AWS Identity and Access Management (IAM) policies to restrict access.
  - D. Use logging to track your content.
7. Which of the following are features of Amazon Elastic Block Store (Amazon EBS)? (Select TWO.)
  - A. Data stored on Amazon EBS is automatically replicated within an Availability Zone.
  - B. Amazon EBS data is automatically backed up to tape.
  - C. Amazon EBS volumes can be encrypted transparently to workloads on the attached instance.
  - D. Data on an Amazon EBS volume is lost when the attached instance is stopped.
8. Which option should you choose for Amazon EFS when tens, hundreds, or thousands of Amazon EC2 instances will be accessing the file system concurrently?
  - A. General-Purpose performance mode
  - B. RAID 0
  - C. Max I/O performance mode
  - D. Change to a larger instance
9. Which of the following must be performed to host a static website in an Amazon Simple Storage Service (Amazon S3) bucket? (Select THREE.)
  - A. Configure the bucket for static hosting, and specify an index and error document.
  - B. Create a bucket with the same name as the website.
  - C. Enable File Transfer Protocol (FTP) on the bucket.
  - D. Make the objects in the bucket world-readable.
  - E. Enable HTTP on the bucket.

- 10.** You have a workload that requires 1 TB of durable block storage at 1,500 IOPS during normal use. Every night there is an extract, transform, load (ETL) task that requires 3,000 IOPS for 15 minutes. What is the most appropriate volume type for this workload?
- A.** Use a Provisioned IOPS SSD volume at 3,000 IOPS.
  - B.** Use an instance store.
  - C.** Use a general-purpose SSD volume.
  - D.** Use a magnetic volume.
- 11.** Which statements about Amazon S3 Glacier are true? (Select THREE.)
- A.** It stores data in objects that live in buckets.
  - B.** Archives are identified by user-specified key names.
  - C.** Archives take 3–5 hours to restore.
  - D.** Vaults can be locked.
  - E.** It can be used as a standalone service and as an Amazon S3 storage class.
- 12.** You are developing an application that will be running on several hundred Amazon EC2 instances. The application on each instance will be required to reach out through a file system protocol concurrently to a file system holding the files. Which storage option should you choose?
- A.** Amazon EFS
  - B.** Amazon EBS
  - C.** Amazon EC2 instance store
  - D.** Amazon S3
- 13.** You need to take a snapshot of an Amazon Elastic Block Store (Amazon EBS) volume. How long will the volume be unavailable?
- A.** It depends on the provisioned size of the volume.
  - B.** The volume will be available immediately.
  - C.** It depends on the amount of data stored on the volume.
  - D.** It depends on whether the attached instance is an Amazon EBS-optimized instance.
- 14.** Amazon Simple Storage Service (S3) bucket policies can restrict access to an Amazon S3 bucket and objects by which of the following? (Select THREE.)
- A.** Company name
  - B.** IP address range
  - C.** AWS account
  - D.** Country of origin
  - E.** Objects with a specific prefix

- 15.** Which of the following are *not* appropriate use cases for Amazon Simple Storage Service (Amazon S3)? (Select TWO.)
- A.** Storing static web content or hosting a static website
  - B.** Storing a file system mounted to an Amazon Elastic Compute Cloud (Amazon EC2) instance
  - C.** Storing backups for a relational database
  - D.** Primary storage for a database
  - E.** Storing logs for analytics
- 16.** Which features enable you to manage access to Amazon Simple Storage Service (Amazon S3) buckets or objects? (Select THREE.)
- A.** Enable static website hosting on the bucket.
  - B.** Create a presigned URL for an object.
  - C.** Use an Amazon S3 Access Control List (ACL) on a bucket or object.
  - D.** Use a lifecycle policy.
  - E.** Use an Amazon S3 bucket policy.
- 17.** Your application stores critical data in Amazon Simple Storage Service (Amazon S3), which must be protected against inadvertent or intentional deletion. How can this data be protected? (Select TWO.)
- A.** Use cross-region replication to copy data to another bucket automatically.
  - B.** Set a vault lock.
  - C.** Enable versioning on the bucket.
  - D.** Use a lifecycle policy to migrate data to Amazon S3 Glacier.
  - E.** Enable MFA Delete on the bucket.
- 18.** You have a set of users that have been granted access to your Amazon S3 bucket. For compliance purposes, you need to keep track of all files accessed in that bucket. To have a record of who accessed your Amazon Simple Storage Service (Amazon S3) data and from where, what should you do?
- A.** Enable versioning on the bucket.
  - B.** Enable website hosting on the bucket.
  - C.** Enable server access logging on the bucket.
  - D.** Create an AWS Identity and Access Management (IAM) bucket policy.
  - E.** Enable Amazon CloudWatch logs.
- 19.** What are some reasons to enable cross-region replication on an Amazon Simple Storage Service (Amazon S3) bucket? (Select THREE.)
- A.** Your compliance requirements dictate that you store data at an even further distance than Availability Zones, which are tens of miles apart.
  - B.** Minimize latency when your customers are in two geographic regions.
  - C.** You need a backup of your data in case of accidental deletion.
  - D.** You have compute clusters in two different AWS Regions that analyze the same set of objects.
  - E.** Your data requires at least five nines of durability.

- 20.** Your company requires that all data sent to external storage be encrypted before being sent. You will be sending company data to Amazon S3. Which Amazon Simple Storage Service (Amazon S3) encryption solution will meet this requirement?
- A. Server-Side Encryption with AWS managed keys (SSE-S3)
  - B. Server-Side Encryption with customer-provided keys (SSE-C)
  - C. Client-side encryption with customer-managed keys
  - D. Server-side encryption with AWS Key Management Service (AWS KMS) keys (SSE-KMS)
- 21.** How is data stored in Amazon Simple Storage Service (Amazon S3) for high durability?
- A. Data is automatically replicated to other regions.
  - B. Data is automatically replicated within a region.
  - C. Data is replicated only if versioning is enabled on the bucket.
  - D. Data is automatically backed up on tape and restored if needed.

# Chapter

# 4



# Hello, Databases

---

**THE AWS CERTIFIED DEVELOPER –  
ASSOCIATE EXAM TOPICS COVERED IN  
THIS CHAPTER MAY INCLUDE, BUT ARE  
NOT LIMITED TO, THE FOLLOWING:**

**Domain 2: Security**

- ✓ 2.1 Make authenticated calls to AWS services.
- ✓ 2.2 Implement encryption using AWS services.

**Domain 3: Development with AWS Services**

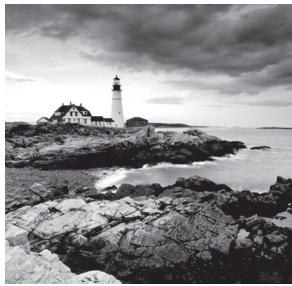
- ✓ 3.2 Translate functional requirements into application design.
- ✓ 3.3 Write code that interacts with AWS services by using APIs, SDKs, and AWS CLI.

**Domain 4: Refactoring**

- ✓ 4.1 Optimize application to best use AWS services and features.

**Domain 5: Monitoring and Troubleshooting**

- ✓ 5.2 Perform root cause analysis on faults found in testing or production.



# Introduction to Databases

In addition to the storage options discussed in the previous chapters, AWS offers a broad range of databases purposely built for your specific application use cases. You can also set up your own database platform on the Amazon Elastic Compute Cloud (Amazon EC2). You can easily migrate your existing databases with the AWS Database Migration Service (AWS DMS) in a cost-effective manner.

AWS Cloud offerings include the following databases:

**Managed relational databases**—For transactional applications

**Nonrelational databases**—For internet-scale applications

**Data warehouse databases**—For analytics

**In-memory data store databases**—For caching and real-time workloads

**Time-series databases**—For efficiently collecting, synthesizing, and deriving insights from time-series data

**Ledger databases**—For when you need a centralized, trusted authority to maintain a scalable, complete, and cryptographically verifiable record of transactions

**Graph databases**—For building applications with highly connected data

Depending on your use case, you can choose from an AWS database that closely aligns with your needs. Table 4.1 describes each database service that AWS offers and indicates the database type.

**TABLE 4.1** AWS Database Service Mapping to Database Type

Product	Type	Description
Amazon Aurora	Relational database	A MySQL- and PostgreSQL-compatible relational database built for the cloud that combines the performance and availability of traditional enterprise databases with the simplicity and cost-effectiveness of open source databases.

Product	Type	Description
Amazon Relational Database Service (Amazon RDS)	Relational database	A managed relational database for MySQL, PostgreSQL, Oracle, SQL Server, and MariaDB. Easy to set up, operate, and scale a relational database in the cloud quickly.
Amazon DynamoDB	NoSQL database	A serverless, managed NoSQL database that delivers consistent single-digit millisecond latency at any scale. Pay only for the throughput and storage you use.
Amazon Redshift	Data warehouse	A fast, fully managed, petabyte-scale data warehouse at one-tenth the cost of traditional solutions. Simple and cost-effective solution to analyze data by using standard SQL and your existing business intelligence (BI) tools.
Amazon ElastiCache	In-memory data store	To deploy, operate, and scale an in-memory data store based on Memcached or Redis in the cloud.
Amazon Neptune	Graph database	A fast, reliable, fully managed graph database to store and manage highly connected datasets.
Amazon Document DB (with MongoDB compatibility)	Nonrelational database	A fast, scalable, highly available, and fully managed document database service that supports MongoDB workloads.
Amazon Timestream	Time series database	A fast, scalable, fully managed time series database service for IoT and operational applications that makes it easy to store and analyze trillions of events per day at one-tenth the cost of relational databases.
Amazon Quantum Ledger Database (Amazon QLDB)	Ledger database	A fully managed ledger database that provides a transparent, immutable, and cryptographically verifiable transaction log owned by a central trusted authority.
AWS Database Migration Service (AWS DMS)	Database migration	Help migrate your databases to AWS easily and inexpensively with minimal downtime.

Now that you know the purpose of these database services and what they can do, review the type of applications that can be used for each AWS database service. Refer to the application type mappings shown in Table 4.2.

**TABLE 4.2** Application Mapping to AWS Database Service

Applications	Product
Transactional applications, such as ERP, CRM, and ecommerce to log transactions and store structured data.	Aurora or Amazon RDS
Internet-scale applications, such as hospitality, dating, and ride sharing, to serve content and store structured and unstructured data.	DynamoDB or Amazon DocumentDB
Analytic applications for operational reporting and querying terabyte- to exabyte-scale data.	Amazon Redshift
Real-time application use cases that require submillisecond latency such as gaming leaderboards, chat, messaging, streaming, and Internet of Things (IoT).	ElastiCache
Applications with use cases that require navigation of highly connected data such as social news feeds, recommendations, and fraud detection.	Neptune
Applications that collect data at millions of inserts per second in a time-series fashion, for example clickstream data and IoT devices.	Timestream
Applications that require an accurate history of their application data; for example, tracking the history of credits and debits in banking transactions or verifying the audit trails created in relational databases.	Amazon QLDB

## Relational Databases

Many developers have had to interact with relational databases in their applications. This section describes first what a relational database is. Then, it covers how you can run a relational database in the AWS Cloud with Amazon RDS or on Amazon EC2.

A *relational database* is a collection of data items with predefined relationships between them. These items are organized as a set of tables with columns and rows. Tables store information about the *objects* to be represented in the database. Each *column* in a table

holds certain data, and a *field* stores the actual value of an attribute. The *rows* in the table represent a collection of related values of one object or entity. Each row in a table contains a unique identifier called a *primary key*, and rows among multiple tables can be linked by using *foreign keys*. You can access data in many different ways without reorganizing the database tables.

## Characteristics of Relational Databases

Relational databases include four important characteristics: Structured Query Language, data integrity, transactions, and atomic, consistent, isolated, and durable compliance.

### Structured Query Language

*Structured query language (SQL)* is the primary interface that you use to communicate with relational databases. The standard American National Standards Institute (ANSI) SQL is supported by all popular relational database engines. Some of these engines have extensions to ANSI SQL to support functionality that is specific to that engine. You use SQL to add, update, or delete data rows; to retrieve subsets of data for transaction processing and analytics applications; and to manage all aspects of the database.

### Data Integrity

*Data integrity* is the overall completeness, accuracy, and consistency of data. Relational databases use a set of constraints to enforce data integrity in the database. These include primary keys, foreign keys, NOT NULL constraints, unique constraint, default constraints, and check constraints. These integrity constraints help enforce business rules in the tables to ensure the accuracy and reliability of the data. In addition, most relational databases enable you to embed custom code triggers that execute based on an action on the database.

### Transactions

A database *transaction* is one or more SQL statements that execute as a sequence of operations to form a single logical unit of work. Transactions provide an all-or-nothing proposition, meaning that the entire transaction must complete as a single unit and be written to the database, or none of the individual components of the transaction will continue. In relational database terminology, a transaction results in a COMMIT or a ROLLBACK. Each transaction is treated in a coherent and reliable way, independent of other transactions.

### ACID Compliance

All database transactions must be *atomic, consistent, isolated, and durable (ACID)*—compliant or be atomic, consistent, isolated, and durable to ensure data integrity.

**Atomicity** *Atomicity* requires that the transaction as a whole executes successfully, or if a part of the transaction fails, then the entire transaction is invalid.

**Consistency** *Consistency* mandates that the data written to the database as part of the transaction must adhere to all defined rules and restrictions, including constraints, cascades, and triggers.

**Isolation** *Isolation* is critical to achieving concurrency control, and it makes sure that each transaction is independent unto itself.

**Durability** *Durability* requires that all of the changes made to the database be permanent when a transaction is successfully completed.

## Managed vs. Unmanaged Databases

Managed database services on AWS, such as Amazon RDS, enable you to offload the administrative burdens of operating and scaling distributed databases to AWS so that you don't have to worry about the following tasks:

- Hardware provisioning
- Setup and configuration
- Throughput capacity planning
- Replication
- Software patching
- Cluster scaling

AWS provides a number of database alternatives for developers. As a managed database, Amazon RDS enables you to run a fully featured relational database while off-loading database administration. By contrast, you can run unmanaged databases on Amazon EC2, which gives you more flexibility on the types of databases that you can deploy and configure; however, you are responsible for the administration of the unmanaged databases.

## Amazon Relational Database Service

With *Amazon Relational Database Service (Amazon RDS)*, you can set up, operate, and scale a relational database in the AWS Cloud. It provides cost-efficient, resizable capacity for open-standard relational database engines. Amazon RDS is easy to administer, and you do not need to install the database software. Amazon RDS manages time-consuming database administration tasks, which frees you up to focus on your applications and business. For example, Amazon RDS automatically patches the database software and backs up your database. The Amazon RDS managed relational database service works with the popular database engines depicted in Figure 4.1.

**FIGURE 4.1** Amazon RDS database engines

Amazon RDS assumes many of the difficult or tedious management tasks of a relational database:

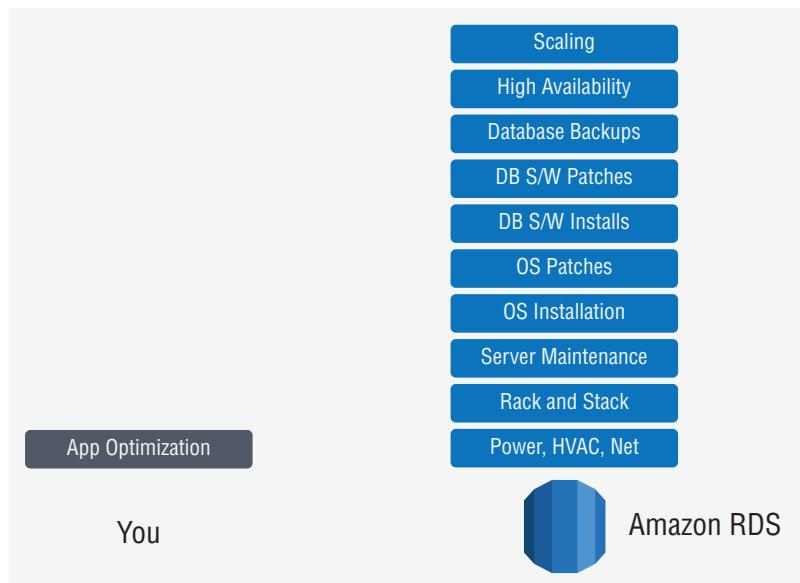
#### Procurement, configuration, and backup tasks

- When you buy a server, you get a central processing unit (CPU), memory, storage, and input/output operations per second (IOPS) all bundled together. With Amazon RDS, these are split apart so that you can scale them independently and allocate your resources as you need them.
- Amazon RDS manages backups, software patches, automatic failure detection, and recovery.
- You can configure automated backups or manually create your own backup snapshot and use these backups to restore a database. The Amazon RDS restore process works reliably and efficiently.
- You can use familiar database products: MySQL, MariaDB, PostgreSQL, Oracle, Microsoft SQL Server, and the MySQL- and PostgreSQL-compatible Amazon Aurora DB engine.

#### Security and availability

- You can enable the encryption option for your Amazon RDS DB instance.
- You can get high availability with a primary instance and a synchronous secondary instance that you can fail over to when problems occur. You can also use MySQL, MariaDB, or PostgreSQL read replicas to increase read scaling.
- In addition to the security in your database package, you can use AWS Identity and Access Management (IAM) to define users, and permissions help control who can access your Amazon RDS databases. You can also help protect your databases by storing them in a virtual private cloud (VPC).
- To deliver a managed service experience, Amazon RDS does not provide shell access to DB instances, and it restricts access to certain system procedures and tables that require advanced permissions.

When you host databases on Amazon RDS, AWS is responsible for the items in Figure 4.2.

**FIGURE 4.2** Amazon RDS host responsibilities

## Relational Database Engines on Amazon RDS

Amazon RDS provides six familiar database engines: Amazon Aurora, Oracle, Microsoft SQL Server, PostgreSQL, MySQL, and MariaDB. Because Amazon RDS is a managed service, you gain a number of benefits and features built right into the Amazon RDS service. These features include, but are not limited to, the following:

- Automatic software patching
- Easy vertical scaling
- Easy storage scaling
- Read replicas
- Automatic backups
- Database snapshots
- Multi-AZ deployments
- Encryption
- IAM DB authentication
- Monitoring and metrics with Amazon CloudWatch

To create an Amazon RDS instance, you can run the following command from the AWS CLI:

```
aws rds create-db-instance \
--db-instance-class db.t2.micro \
--allocated-storage 30 \
--db-instance-identifier my-cool-rds-db --engine mysql \
--master-username masteruser --master-user-password masterpassword1!
```

Depending on the configurations chosen, the database can take several minutes before it is active and ready for use. You can monitor the Amazon RDS Databases console to view the status. When the status states Available, it is ready to be used, as shown in Figure 4.3.

**FIGURE 4.3** Amazon RDS Databases console

DB identifier	Role	Engine	Region & AZ	Size	Status
my-cool-rds-db	Instance	MySQL	us-east-1f	db.t2.micro	Available

## Automatic Software Patching

Periodically, Amazon RDS performs maintenance on Amazon RDS resources. Maintenance mostly involves patching the Amazon RDS database underlying operating system (OS) or database engine version. Because this is a managed service, Amazon RDS handles the patching for you.

When you create an Amazon RDS database instance, you can define a maintenance window. A *maintenance window* is where you can define a period of time when you want to apply any updates or downtime to your database instance. You also can enable the automatic minor version upgrade feature, which automatically applies any new minor versions of the database as they are released (see Figure 4.4).

**FIGURE 4.4** Maintenance window

Maintenance

Auto minor version upgrade [Info](#)

Enable auto minor version upgrade  
Enabling auto minor version upgrade will automatically upgrade to new minor versions as they are released. The automatic upgrades occur during the maintenance window for the database.

Maintenance window [Info](#)  
Select the period you want pending modifications or maintenance applied to the database by Amazon RDS.

Select window

No preference

Start day: Monday

Start time: 00 : 00 UTC

Duration: 0.5 hours

You can select a maintenance window by using the AWS Management Console, AWS CLI, or Amazon RDS API. After selecting the maintenance window, the Amazon RDS instance is upgraded (if upgrades are available) during that time window. You can also modify the maintenance window by running the following AWS CLI command:

```
aws rds modify-db-instance --db-instance-identifier your-db-instance-identifier
--preferred-maintenance-window Mon:07:00-Mon:07:30
```

### Vertical Scaling

If your database needs to handle a bigger load, you can vertically scale your Amazon RDS instance. At the time of this writing, there are 40 available DB instance classes, which enable you to choose the number of virtual CPUs and memory available. This gives you flexibility over the performance and cost of your Amazon RDS database. To scale the Amazon RDS instance, you can use the console, AWS CLI, or AWS SDK.

If you are in a Single-AZ configuration for your Amazon RDS instance, the database is unavailable during the scaling operation. However, if you are in a Multi-AZ configuration, the standby database is upgraded first and then a failover occurs to the newly configured database. You can also apply the change during the next maintenance window. This way, your upgrade can occur during your normal outage windows.

To scale the Amazon RDS database by using the AWS CLI, run the following command:

```
aws rds modify-db-instance --db-instance-identifier your-db-instance-identifier
--db-instance-class db.t2.medium
```

### Easy Storage Scaling

Storage is a critical component for any database. Amazon RDS has the following three storage types:

**General Purpose SSD (gp2)** This storage type is for cost-effective storage that is ideal for a broad range of workloads. Gp2 volumes deliver single-digit millisecond latencies and the ability to burst to 3,000 IOPS for extended periods of time. The volume's size determines the performance of gp2 volumes.

**Provisioned IOPS (io1)** This storage type is for input/output-intensive workloads that require low input/output (I/O) latency and consistent I/O throughput.

**Magnetic Storage** This storage type is designed for backward compatibility, and AWS recommends that you use General Purpose SSD or Provisioned IOPS for any new Amazon RDS workloads.

To scale your storage, you must modify the Amazon RDS DB instance by executing the following AWS CLI command:

```
aws rds modify-db-instance --db-instance-identifier your-db-instance-identifier
--allocated-storage 50 --storage-type io1 --iops 3000
```

This command modifies your storage to 50 GB in size, with a Provisioned IOPS storage drive and a dedicated IOPS of 3000. While modifying the Amazon RDS DB instance, consider the potential downtime.

### Read Replicas (Horizontal Scaling)

There are two ways to scale your database tier with Amazon RDS: vertical scaling and horizontal scaling. Vertical scaling takes the primary database and increases the amount of memory and vCPUs allocated for the primary database. Alternatively, use horizontal scaling (add another server) to your database tier to improve the performance of applications that are read-heavy as opposed to write-heavy.

Read replicas create read-only copies of your master database, which allow you to offload any reads (or SQL SELECT statements) to the read replica. The replication from the master database to the read replica is asynchronous. As a result, the data queried from the read replica is not the latest data. If your application requires strongly consistent reads, consider an alternative option.

At the time of this writing, Amazon RDS MySQL, PostgreSQL, and MariaDB support up to five read replicas, and Amazon Aurora supports up to 15 read replicas. Microsoft SQL Server and Oracle do not support read replicas.

To create a read replica by using AWS CLI, run the following command:

```
aws rds create-db-instance-read-replica --db-instance-identifier your-db-instance-identifier --source-db-instance-identifier your-source-db
```

## Backing Up Data with Amazon RDS

Amazon RDS has two different ways of backing up data of your database instance: automated backups and database snapshots (DB snapshots).

### Automated Backups (Point-in-Time)

With Amazon RDS, automated backups offer a point-in-time recovery of your database. When enabled, Amazon RDS performs a full daily snapshot of your data that is taken during your preferred backup window. After the initial backup is taken (each day), then Amazon RDS captures transaction logs as changes are made to the database.

After you initiate a point-in-time recovery, to restore your database instance, the transaction logs are applied to the most appropriate daily backup. You can perform a restore up to the specific second, as long as it's within your retention period. The default retention period is seven days, but it can be a maximum of up to 35 days.

To perform a restore, you must choose the Latest Restorable Time, which is typically within the last 5 minutes. For example, suppose that the current date is February 14 at 10 p.m., and you would like to do a point-in-time restore of February 14 at 9 p.m. This restore would succeed because the Latest Restorable Time is a maximum of February 14 at 9:55 p.m. (which is the last 5-minute window). However, a point-in-time restore of February 14 at 9:58 p.m. would fail, because it is within the 5-minute window.

Automated backups are kept until the source database is deleted. After the source Amazon RDS instance is removed, the automated backups are also removed.

### Database Snapshots (Manual)

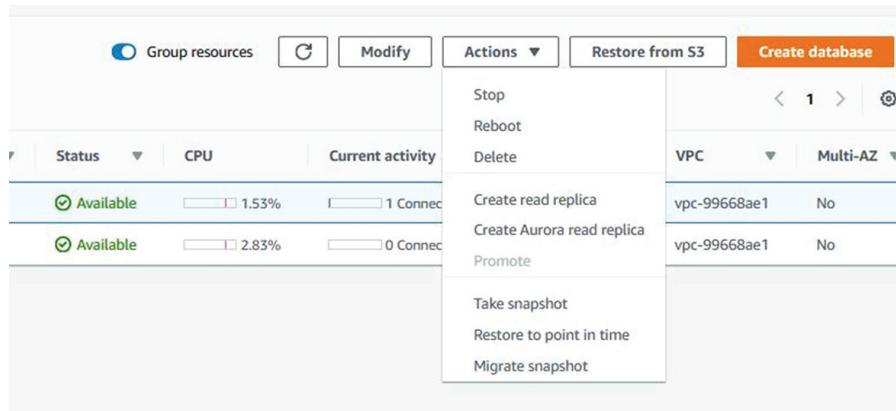
Unlike automated backups, database snapshots with Amazon RDS are user-initiated and enable you to back up your database instance in a known state at any time. You can also restore to that specific snapshot at any time.

Similar to the other Amazon RDS features, you can create the snapshots through the AWS Management Console, with the `CreateDBSnapshot` API, or with the AWS CLI.

With DB snapshots, the backups are kept until you explicitly delete them; therefore, before removing any Amazon RDS instance, take a final snapshot before removing it. Regardless of the backup taken, storage I/O may be briefly suspended while the backup process initializes (typically a few seconds), and you may experience a brief period of elevated latency. A way to avoid these types of suspensions is to deploy in a Multi-AZ configuration. With such a deployment, the backup is taken from the standby instead of the primary database.

To create a snapshot of the database, from the Amazon RDS Databases console, under Actions, select the Take snapshot option (see Figure 4.5). After a snapshot is taken, you can view all of your snaps from the Snapshots console.

**FIGURE 4.5** Taking an Amazon RDS snapshot



### Multi-AZ Deployments

By using Amazon RDS, you can run in a Multi-AZ configuration. In a Multi-AZ configuration, you have a primary and a standby DB instance. Updates to the primary database replicate synchronously to the standby replica in a different Availability Zone. The primary benefit of Multi-AZ is realized during certain types of planned maintenance, or in the unlikely event of a DB instance failure or an Availability Zone failure. Amazon RDS automatically fails over to the standby so that you can resume your workload as soon as the standby is promoted to the primary. This means that you can reduce your downtime in the event of a failure.

Because Amazon RDS is a managed service, Amazon RDS handles the failover to the standby. When there is a DB instance failure, Amazon RDS automatically promotes the standby to the primary—you will not interact with the standby directly. In other words, you will receive one endpoint connection for the Amazon RDS cluster, and Amazon RDS handles the failover.

Amazon RDS Multi-AZ configuration provides the following benefits:

- Automatic failover; no administration required
- Increased durability in the unlikely event of component failure
- Increased availability in the unlikely event of an Availability Zone failure
- Increased availability for planned maintenance (automated backups; I/O activity is no longer suspended)

To create an Amazon RDS instance in a Multi-AZ configuration, you must specify a subnet group that has two different Availability Zones specified. You can specify a Multi-AZ configuration by using AWS CLI by adding the `--multi-az` flag to the AWS CLI command, as follows:

```
aws rds create-db-instance \
--db-instance-class db.t2.micro \
--allocated-storage 30 \
--db-instance-identifier multi-az-rds-db --engine mysql \
--master-username masteruser \
--master-user-password masterpassword1! \
--multi-az
```

## Encryption

For encryption at rest, Amazon RDS uses the AWS Key Management Service (AWS KMS) for AES-256 encryption. You can use a default master key or specify your own for the Amazon RDS DB instance. Encryption is one of the few options that must be configured when the DB instance is created. You cannot modify an Amazon RDS database to enable encryption. You can, however, create a DB snapshot and then restore to an encrypted DB instance or cluster.

Amazon RDS supports using the Transparent Data Encryption (TDE) for Oracle and SQL Server. For more information on TDE with Oracle and Microsoft SQL Server, see the following:

- Microsoft SQL Server Transparent Data Encryption Support at:  
<https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Appendix.SQLServer.Options.TDE.html>
- Options for Oracle DB Instances:  
<https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Appendix.Oracle.Options.html#Appendix.Oracle.Options.AdvSecurity>

At the time of this writing, the following Amazon RDS DB instance types are not supported for encryption at rest:

- Db.m1.small
- Db.m1.medium
- Db.m1.large
- Db.m1.xlarge
- Db.m2.xlarge
- Db.m2.2xlarge
- Db.m2.4xlarge
- Db.t2.micro

For encryption in transit, Amazon RDS generates an SSL certificate for each database instance that can be used to connect your application and the Amazon RDS instance. However, encryption is a compute-intensive operation that increases the latency of your database connection. For more information, see the documentation for the specific database engine.

## IAM DB Authentication

You can authenticate to your DB instance by using IAM. By using IAM, you can manage access to your database resources centrally instead of storing the user credentials in each database. The IAM feature also encrypts network traffic to and from the database by using SSL.

IAM DB authentication is supported only for MySQL and PostgreSQL. At the time of this writing, the following MySQL versions are supported:

- MySQL 5.6.34 or later
- MySQL 5.7.16 or later

There's no support for the following:

- IAM DB Authentication for MySQL 5.5 or MySQL 8.0
- db.t2.small and db.m1.small instances

The following PostgreSQL versions are supported:

- PostgreSQL versions 10.6 or later
- PostgreSQL 9.6.11 or later
- PostgreSQL 9.5.15 or later

To enable IAM DB authentication for your Amazon RDS instance, run the following command:

```
aws rds modify-db-instance --db-instance-identifier my-rds-db --enable-iam-database-authentication --apply-immediately
```

Because downtime is associated with this action, you can enable this feature during the next maintenance window. You can do so by changing the last parameter to `--no-apply-immediately`.

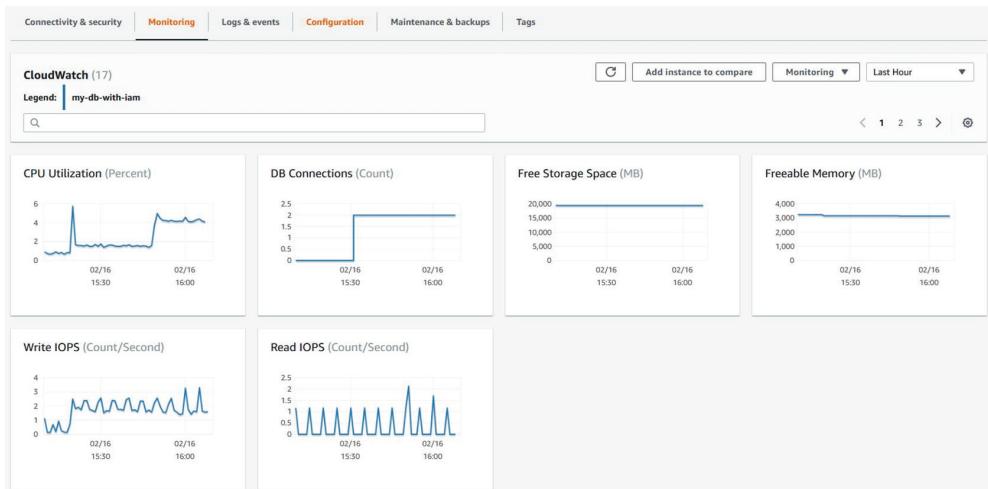
## Monitoring with Amazon CloudWatch

Use Amazon CloudWatch to monitor your database tier. You can create alarms to notify database administrators when there is a failure.

By default, CloudWatch provides some built-in metrics for Amazon RDS with a granularity of 5 minutes (600 seconds). If you want to gather metrics in a smaller window of granularity, such as 1 second, enable enhanced monitoring, which is similar to how you enable these features in Amazon EC2.

To view all the Amazon RDS metrics that are provided through CloudWatch, select the Monitoring tab from the Amazon RDS console (see Figure 4.6).

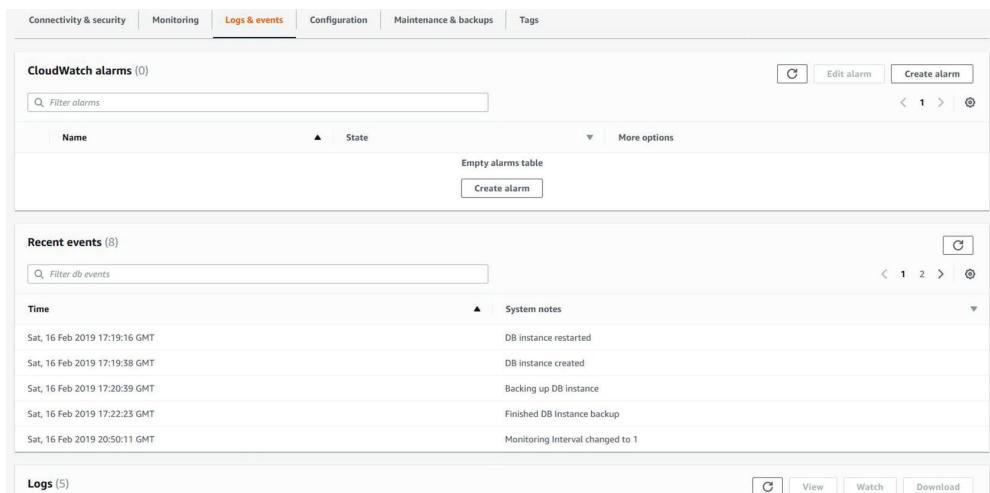
**FIGURE 4.6** Amazon RDS with CloudWatch metrics



Amazon RDS integrates with CloudWatch to send it the following database logs:

- Audit log
- Error log
- General log
- Slow query log

From the Amazon RDS console, select the Logs & events tab to view and download the specified logs, as shown in Figure 4.7.

**FIGURE 4.7** Amazon RDS with CloudWatch Logs

For more information on CloudWatch and its capabilities across other AWS services, see Chapter 15, “Monitoring and Troubleshooting.”

## Amazon Aurora

*Amazon Aurora* is a MySQL- and PostgreSQL-compatible relational database engine that combines the speed and availability of high-end commercial databases with the simplicity and cost-effectiveness of open source databases.

Aurora is part of the managed database service Amazon RDS.

### Amazon Aurora DB Clusters

Aurora is a drop-in replacement for MySQL and PostgreSQL relational databases. It is built for the cloud, and it combines the performance and availability of high-end commercial databases with the simplicity and cost-effectiveness of open source databases. You can use the code, tools, and applications that you use today with your existing MySQL and PostgreSQL databases with Aurora.

The integration of Aurora with Amazon RDS means that time-consuming administration tasks, such as hardware provisioning, database setup, patching, and backups, are automated.

Aurora features a distributed, fault-tolerant, self-healing storage system that automatically scales up to 64 TiB per database instance. (In comparison, other Amazon RDS options allow for a maximum of 32 TiB.) Aurora delivers high performance and availability with up to 15 low-latency read replicas, point-in-time recovery, continuous backup to Amazon Simple Storage Service (Amazon S3), and replication across three Availability Zones. When you create an Aurora instance, you create a DB cluster. A *DB cluster* consists of one or more DB instances and a cluster volume that manages the data for those

instances. An Aurora *cluster volume* is a virtual database storage volume that spans multiple Availability Zones, and each Availability Zone has a copy of the DB cluster data.

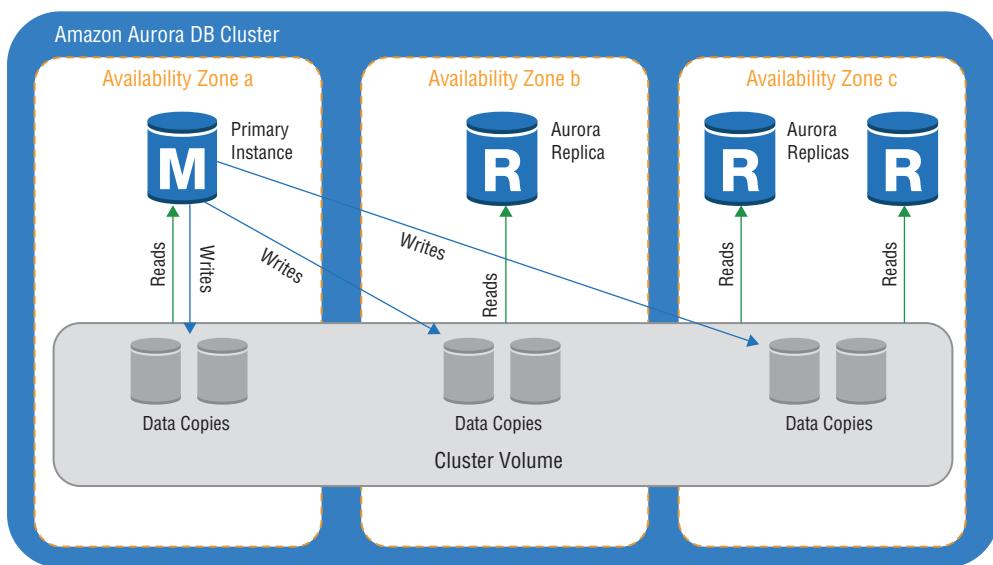
An Aurora DB cluster has two types of DB instances:

**Primary Instance** Supports read and write operations and performs all of the data modifications to the cluster volume. Each Aurora DB cluster has one primary instance.

**Amazon Aurora Replica** Supports read-only operations. Each Aurora DB cluster can have up to 15 *Amazon Aurora Replicas* in addition to the primary instance. Multiple Aurora Replicas distribute the read workload, and if you locate Aurora Replicas in separate Availability Zones, you can also increase database availability.

Figure 4.8 illustrates the relationship between the cluster volume, the primary instance, and Aurora Replicas in an Aurora DB cluster.

**FIGURE 4.8** Amazon Aurora DB cluster



As you can see from Figure 4.8, this architecture is vastly different from the other Amazon RDS databases. Aurora is engineered and architected for the cloud. The primary difference is that there is a separate storage layer, called the *cluster volume*, which is spread across multiple Availability Zones in a single AWS Region. This means that the durability of your data is increased.

Additionally, Aurora has one primary instance that writes across the cluster volume. This means that Aurora replicas can be spun up quickly, because they don't have to copy and store their own storage layer; they connect to it. Because the cluster volume is separated in this architecture, the cluster volume can grow automatically as your data increases. This is in contrast to how other Amazon RDS databases are built, whereby you must define the allocated storage in advance.

### Amazon Aurora Global Databases

With Aurora, you can also create a multiregional deployment for your database tier. In this configuration, the primary AWS Region is where your data is written (you may also do reads from the primary AWS Region). Any application performing writes must write to the primary AWS Region where the cluster is operating.

The secondary AWS Region is used for *reading* data only. Aurora replicates the data to the secondary AWS Region with typical latency of less than a second. Furthermore, you can use the secondary AWS Region for disaster recovery purposes. You can promote the secondary cluster and make it available as the primary typically in less than a minute. At the time of this writing, Aurora global databases are available in the following AWS Regions only:

- US East (N. Virginia)
- US East (Ohio)
- US West (Oregon)
- EU (Ireland)

Additionally, at the time of this writing, Aurora global databases are available only for MySQL 5.6.

### Amazon Aurora Serverless

Aurora Serverless is an on-demand, automatic scaling configuration for Aurora. (It is available only for MySQL at the time of this writing.) With Aurora Serverless, the database will *automatically* start up, shut down, and scale capacity up or down based on your application's needs. This means that, as a developer, you can run your database in the AWS Cloud and not worry about managing any database instances.

## Best Practices for Running Databases on AWS

The following are best practices for working with Amazon RDS:

**Follow Amazon RDS basic operational guidelines.** The Amazon RDS Service Level Agreement requires that you follow these guidelines:

- Monitor your memory, CPU, and storage usage. Amazon CloudWatch can notify you when usage patterns change or when you approach the capacity of your deployment so that you can maintain system performance and availability.
- Scale up your DB instance when you approach storage capacity limits. Have some buffer in storage and memory to accommodate unforeseen increases in demand from your applications.
- Enable automatic backups, and set the backup window to occur during the daily low in write IOPS.

- If your database workload requires more I/O than you have provisioned, recovery after a failover or database failure will be slow. To increase the I/O capacity of a DB instance, do any or all of the following:
  - Migrate to a DB instance class with high I/O capacity.
  - Convert from standard storage either to General Purpose or Provisioned IOPS storage, depending on how much of an increase you need. If you convert to Provisioned IOPS storage, make sure that you also use a DB instance class that is optimized for Provisioned IOPS.
  - If you are already using Provisioned IOPS storage, provision additional throughput capacity.
- If your client application is caching the Domain Name Service (DNS) data of your DB instances, set a time-to-live (TTL) value of less than 30 seconds. Because the underlying IP address of a DB instance can change after a failover, caching the DNS data for an extended time can lead to connection failures if your application tries to connect to an IP address that no longer is in service.
- Test failover for your DB instance to understand how long the process takes for your use case and to ensure that the application that accesses your DB instance can automatically connect to the new DB instance after failover.

**Allocate sufficient RAM to the DB instance.** An Amazon RDS performance best practice is to allocate enough RAM so that your working set resides almost completely in memory. Check the ReadIOPS metric by using CloudWatch while the DB instance is under load to view the working set. The value of ReadIOPS should be small and stable. Scale up the DB instance class until ReadIOPS no longer drops dramatically after a scaling operation or when ReadIOPS is reduced to a small amount.

**Implement Amazon RDS security.** Use IAM accounts to control access to Amazon RDS API actions, especially actions that create, modify, or delete Amazon RDS resources, such as DB instances, security groups, option groups, or parameter groups, and actions that perform common administrative actions, such as backing up and restoring DB instances, or configuring Provisioned IOPS storage.

- Assign an individual IAM account to each person who manages Amazon RDS resources. Do not use an AWS account user to manage Amazon RDS resources; create an IAM user for everyone, including yourself.
- Grant each user the minimum set of permissions required to perform his or her duties.
- Use IAM groups to manage permissions effectively for multiple users.
- Rotate your IAM credentials regularly.

Use the AWS Management Console, the AWS CLI, or the Amazon RDS API to change the password for your master user. If you use another tool, such as a SQL client, to change the master user password, it might result in permissions being revoked for the user unintentionally.

**Use enhanced monitoring to identify OS issues.** Amazon RDS provides metrics in real time for the OS on which your DB instance runs. You can view the metrics for your DB

instance by using the console or consume the Enhanced Monitoring JSON output from CloudWatch Logs in a monitoring system of your choice. Enhanced Monitoring is available for the following database engines:

- MariaDB
- Microsoft SQL Server
- MySQL version 5.5 or later
- Oracle
- PostgreSQL

Enhanced Monitoring is available for all DB instance classes except for db.m1.small. Enhanced Monitoring is available in all regions except for AWS GovCloud (US).

**Use metrics to identify performance issues.** To identify performance issues caused by insufficient resources and other common bottlenecks, you can monitor the metrics available for your Amazon RDS DB instance.

Monitor performance metrics regularly to see the average, maximum, and minimum values for a variety of time ranges. If you do so, you can identify when performance is degraded. You can also set CloudWatch alarms for particular metric thresholds.

To troubleshoot performance issues, it's important to understand the baseline performance of the system. When you set up a new DB instance and get it running with a typical workload, you should capture the average, maximum, and minimum values of all the performance metrics at a number of different intervals (for example, 1 hour, 24 hours, 1 week, or 2 weeks) to get an idea of what is normal. It helps to get comparisons for both peak and off-peak hours of operation. You can then use this information to identify when performance is dropping below standard levels.

**Tune queries.** One of the best ways to improve DB instance performance is to tune your most commonly used and most resource-intensive queries to make them less expensive to run.

A common aspect of query tuning is creating effective indexes. You can use the Database Tuning Advisor to get potential index improvements for your DB instance.

**Use DB parameter groups.** AWS recommends that you apply changes to the DB parameter group on a test DB instance before you apply parameter group changes to your production DB instances. Improperly setting DB engine parameters in a DB parameter group can have unintended adverse effects, including degraded performance and system instability. Always exercise caution when modifying DB engine parameters, and back up your DB instance before modifying a DB parameter group.

**Use read replicas.** Use read replicas to relieve pressure on your master node with additional read capacity. You can bring your data closer to applications in different regions and promote a read replica to a master for faster recovery in the event of a disaster.



---

You can use the AWS Database Migration Service (AWS DMS) to migrate or replicate your existing databases easily to Amazon RDS.

# Nonrelational Databases

Nonrelational databases are commonly used for internet-scale applications that do not require any complex queries.

## NoSQL Database

*NoSQL databases* are nonrelational databases optimized for scalable performance and schema-less data models. NoSQL databases are also widely recognized for their ease of development, low latency, and resilience.

NoSQL database systems use a variety of models for data management, such as in-memory key-value stores, graph data models, and document stores. These types of databases are optimized for applications that require large data volume, low latency, and flexible data models, which are achieved by relaxing some of the data consistency restrictions of traditional relational databases.

### When to Use a NoSQL Database

NoSQL databases are a great fit for many big data, mobile, and web applications that require greater scale and higher responsiveness than traditional relational databases. Because of simpler data structures and horizontal scaling, NoSQL databases typically respond faster and are easier to scale than relational databases.

### Comparison of SQL and NoSQL Databases

Many developers are familiar with SQL databases but might be new to working with NoSQL databases. Relational database management systems (RDBMS) and nonrelational (NoSQL) databases have different strengths and weaknesses. In a RDBMS, data can be queried flexibly, but queries are relatively expensive and do not scale well in high-traffic situations. In a NoSQL database, you can query data efficiently in a limited number of ways. Table 4.3 shows a comparison of different characteristics of SQL and NoSQL databases.

**TABLE 4.3** SQL vs. NoSQL Database Characteristics

Type	SQL	NoSQL
Data Storage	Rows and columns	Key-value, document, wide-column, graph
Schemas	Fixed	Dynamic
Querying	Using SQL	Focused on collection of documents
Scalability	Vertical	Horizontal
Transactions	Supported	Support varies
Consistency	Strong	Eventual and strong

The storage format for SQL versus NoSQL databases also differs. As shown in Figure 4.9, SQL databases are often stored in a row and column format, whereas NoSQL databases, such as Amazon DynamoDB, have a key-value format that could be in a JSON format, as shown in this example.

**FIGURE 4.9** SQL versus NoSQL format comparison

SQL			
ISBN	Title	Edition	Format
1119138558	AWS Certified Solutions Architect Study Guide	1	Paperback

NoSQL
{ ISBN: 9182932465265, Title: "AWS Certified Solutions Architect Study Guide", Edition: "1", Format: "Paperback" }

## NoSQL Database Types

There are four types of NoSQL databases: columnar, document, graph, and in-memory key-value. Generally, these databases differ in how the data is stored, accessed, and structured, and they are optimized for different use cases and applications.

**Columnar databases** Columnar databases are optimized for reading and writing columns of data as opposed to rows of data. Column-oriented storage for database tables is an important factor in analytic query performance because it drastically reduces the overall disk I/O requirements and reduces the amount of data that you must load from disk.

**Document databases** Document databases are designed to store semi-structured data as documents, typically in JSON or XML format. Unlike traditional relational databases, the schema for each NoSQL document can vary, giving you more flexibility in organizing and storing application data and reducing storage required for optional values.

**Graph databases** Graph databases store vertices and directed links called *edges*. Graph databases can be built on both SQL and NoSQL databases. Vertices and edges can each have properties associated with them.

**In-memory key-value stores** In-memory key-value stores are NoSQL databases optimized for read-heavy application workloads (such as social networking, gaming, media sharing, and Q&A portals) or compute-intensive workloads (such as a recommendation engine). In-memory caching improves application performance by storing critical pieces of data in memory for low-latency access.

## Amazon DynamoDB

Amazon DynamoDB is a fast and flexible NoSQL database service for all applications that need consistent, single-digit millisecond latency at any scale. It is a fully managed cloud

database, and it supports both document and key-value store models. Its flexible data model, reliable performance, and automatic scaling of throughput capacity make it a great fit for the following:

- Mobile
- Gaming
- Adtech
- Internet of Things (IoT)
- Applications that do not require complex queries

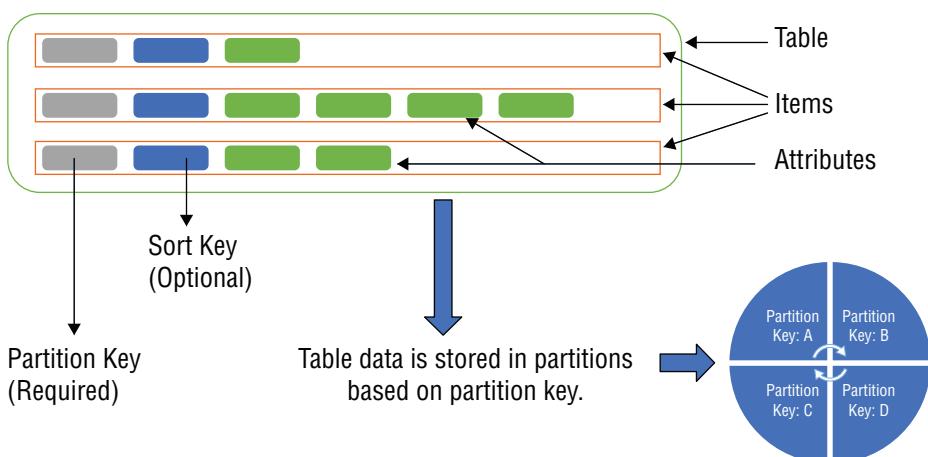
With DynamoDB, you can create database tables that can store and retrieve any amount of data and serve any level of request traffic. You can scale up or scale down your table throughput capacity without downtime or performance degradation. DynamoDB automatically spreads the data and traffic for your tables over a sufficient number of servers to handle your throughput and storage requirements while maintaining consistent and fast performance. All of your data is stored on solid-state drives (SSDs) and automatically replicated across multiple Availability Zones in an AWS Region, providing built-in high availability and data durability. You can use global tables to keep DynamoDB tables in sync across AWS Regions.

## Core Components of Amazon DynamoDB

In DynamoDB, tables, items, and attributes are the core components with which you work. A *table* is a collection of items, and each item is a collection of *attributes*. DynamoDB uses partition keys to identify uniquely each item in a table. Secondary indexes can be used to provide more querying flexibility. You can use DynamoDB Streams to capture data modification events in DynamoDB tables.

Figure 4.10 shows the DynamoDB data model, including a table, items, attributes, a required partition key, an optional sort key, and an example of data being stored in partitions.

**FIGURE 4.10** Amazon DynamoDB tables and partitions



## Tables

Similar to other database systems, DynamoDB stores data in tables. A table is a collection of items. For example, a table called *People* could be used to store personal contact information about friends, family, or anyone else of interest.

## Items

An item in DynamoDB is similar in many ways to rows, records, or tuples in other database systems. Each DynamoDB table contains zero or more items. An *item* is a collection of attributes that is uniquely identifiable for each record in that table. For a *People* table, each item represents a person. There is no limit to the number of items that you can store in a table.

## Attributes

Each item is composed of one or more attributes. Attributes in DynamoDB are similar in many ways to fields or columns in other database systems. An attribute is a fundamental data element, something that does not need to be broken down any further. You can think of an attribute as similar to columns in a relational database. For example, an item in a *People* table contains attributes called *PersonID*, *Last Name*, *First Name*, and so on.

Figure 4.11 shows a table named *People* with items and attributes. Each block represents an item, and within those blocks you have attributes that define the overall item:

- Each item in the table has a unique identifier, a primary key, or a partition key that distinguishes the item from all of the others in the table. The primary key consists of one attribute (*PersonID*).
- Other than the primary key, the *People* table is schemaless, which means that you do not have to define the attributes or their data types beforehand. Each item can have its own distinct attributes. This is where the contrast begins to show between NoSQL and SQL. In SQL, you would have to define a schema for each person, and every person would need to have the same data points or attributes. As you can see in Figure 4.11, with NoSQL and DynamoDB, each person can have different attributes.
- Most of the attributes are scalar, so they can have only one value. Strings and numbers are common examples of scalars.
- Some of the items have a nested attribute (*Address*). DynamoDB supports nested attributes up to 32 levels deep.

**FIGURE 4.11** Amazon DynamoDB table with items and attributes

## Primary Key

When you create a table, at a minimum, you are required to specify the table name and primary key of the table. The primary key uniquely identifies each item in the table. No two items can have the same key within a table.

DynamoDB supports two different kinds of primary keys: partition key and partition key and sort key.

**Partition key (hash key)** A simple primary key, composed of one attribute, is known as the *partition key*. DynamoDB uses the partition key's value as an input to an internal hash function. The output from the hash function determines the partition (physical storage internal to DynamoDB) in which the item is stored.

In a table that has only a partition key, no two items can have the same partition key value. For example, in the *People* table, with a simple primary key of PersonID, you cannot have two items with PersonID of 000-07-1075.

The partition key of an item is also known as its *hash attribute*. The term *hash attribute* derives from the use of an internal hash function in DynamoDB that evenly distributes data items across partitions based on their partition key values.

Each primary key attribute must be a scalar (meaning that it can hold only a single value). The only data types allowed for primary key attributes are string, number, or binary. There are no such restrictions for other, nonkey attributes.

**Partition key and sort key (range attribute)** A *composite primary key* is composed of two attributes: *partition key* and the *sort key*.

The *sort key* of an item is also known as its *range attribute*. The term *range attribute* derives from the way that DynamoDB stores items with the same partition key physically close together, in sorted order, by the sort key value.

The *partition key* acts the same as the sort key, but in addition to also using a sort key, the items with the same partition key are stored together, in sorted order, by sort key value.

In a table that has a partition key and a sort key, it's possible for two items to have the same partition key value, but those two items must have different sort key values. You cannot have two items in the table that have identical partition key and sort key values.

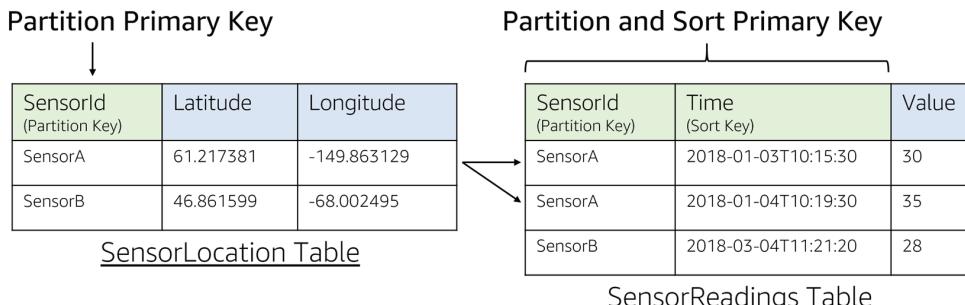
For example, if you have a *Music* table with a composite primary key (*Artist* and *SongTitle*), you can access any item in the *Music* table directly if you provide the *Artist* and *SongTitle* values for that item.

A composite primary key gives you additional flexibility when querying data. For example, if you provide only the value for *Artist*, DynamoDB retrieves all of the songs by that artist. To retrieve only a subset of songs by a particular artist, you can provide a value for *Artist* with a range of values for *SongTitle*.

As a developer, the attribute you choose for your application has important implications. If there is little differentiation among partition keys, all of your data is stored together in the same physical location.

Figure 4.12 shows an example of these two types of keys. In the *SensorLocation* table, the primary key is the *SensorId* attribute. This means that every item (or row) in this table has a unique *SensorId*, meaning that each sensor has exactly one location or latitude and longitude value.

**FIGURE 4.12** Amazon DynamoDB primary keys



Conversely, the *SensorReadings* table has a partition key and a sort key. The *SensorId* attribute is the partition key and the *Time* attribute is the sort key, which combined make it a *composite key*. For each *SensorId*, there may be multiple items corresponding to sensor readings at different times. The combination of *SensorId* and *Time* uniquely identifies items in the table. This design enables you to query the table for all readings related to a particular sensor.

## Secondary Indexes

If you want to perform queries on attributes that are not part of the table's primary key, you can create a *secondary index*. By using a secondary index, you can query the data in the table by using an alternate key, in addition to querying against the primary key. DynamoDB does not require that you use indexes, but doing so may give you more flexibility when querying your data depending on your application and table design.

After you create a secondary index on a table, you can then read data from the index in much the same way as you do from the table. DynamoDB automatically creates indexes based on the primary key of a table and automatically updates all indexes whenever a table changes.

A secondary index contains the following:

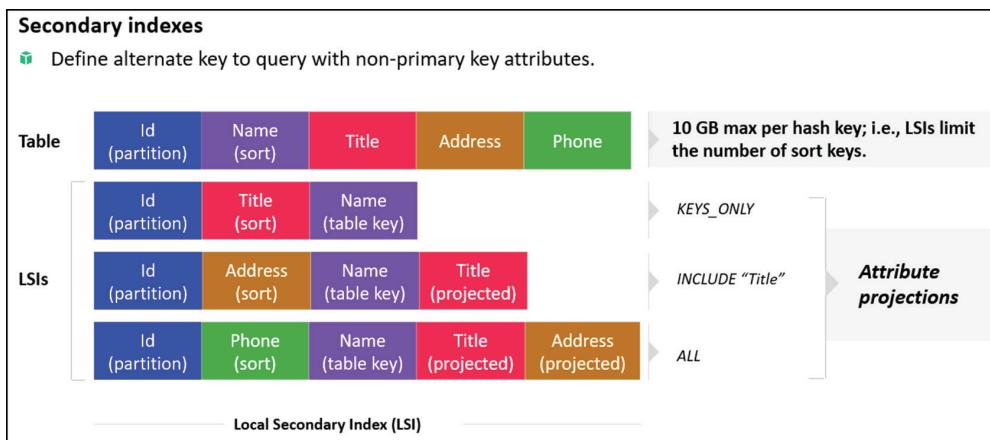
- Primary key attributes
- Alternate key attributes
- (Optional) A subset of other attributes from the base table (*projected attributes*)

DynamoDB provides fast access to the items in a table by specifying primary key values. However, many applications might benefit from having one or more secondary (or alternate) keys available. This allows efficient access to data with attributes other than the primary key.

DynamoDB supports two types of secondary indexes: *local secondary indexes* and *global secondary indexes*. You can define up to five global secondary indexes and five local secondary indexes per table.

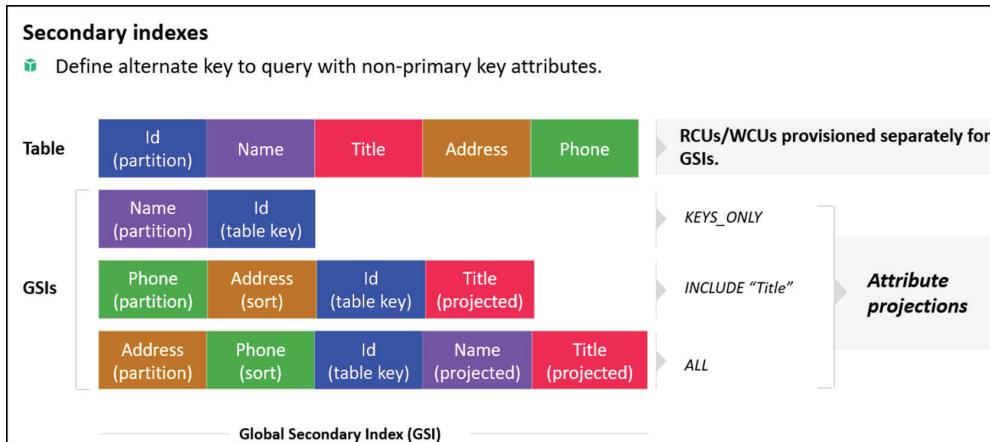
### Local Secondary Index

A *local secondary index* is an index that has the same partition key as the base table, but a different sort key (see Figure 4.13). It is “local” in the sense that every partition of a local secondary index is scoped to a base table partition that has the same partition key value. You can construct only one while creating the table, but you cannot add, remove, or modify it later.

**FIGURE 4.13** Local secondary index

## Global Secondary Index

A *global secondary index* is an index with a partition key and a sort key that can be different from those on the base table (see Figure 4.14). It is considered “global” because queries on the index can span all of the data in the base table across all partitions. You can create one during table creation, and you can add, remove, or modify it later.

**FIGURE 4.14** Global secondary index

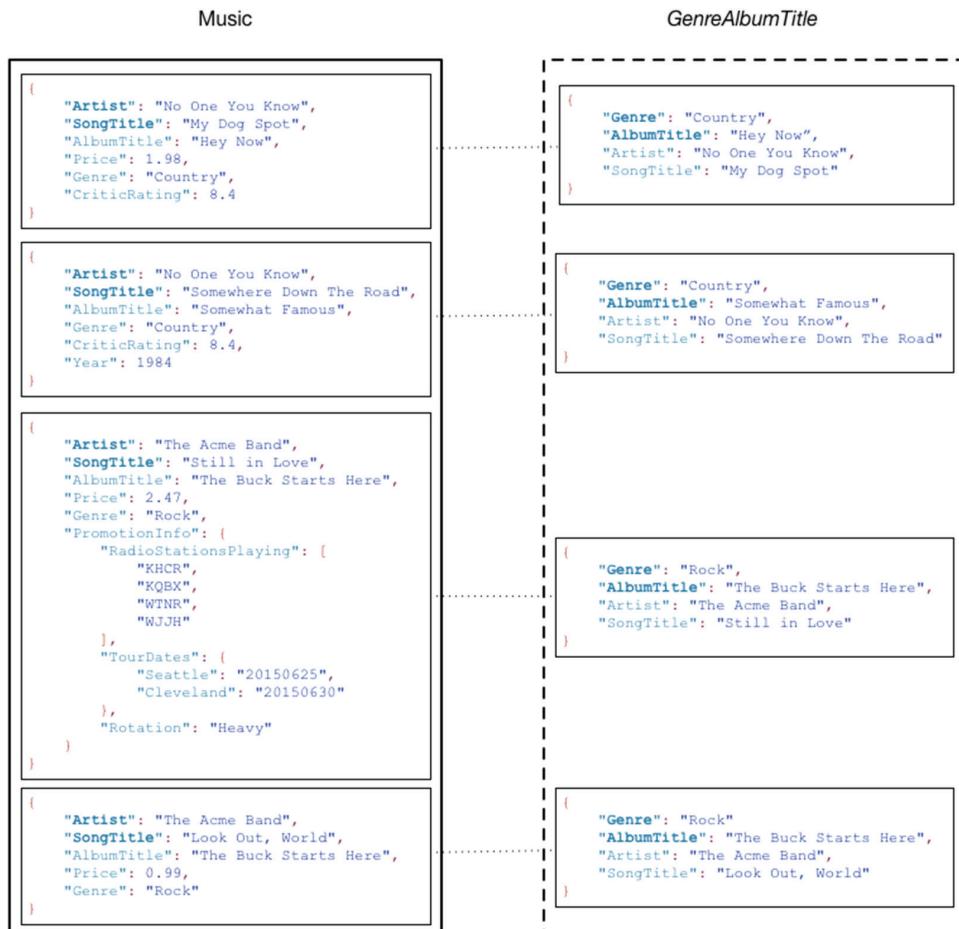


You can create a global secondary index, not a local secondary index, after table creation.

For example, by using a *Music* table, you can query data items by *Artist* (partition key) or by *Artist* and *SongTitle* (partition key and sort key). Suppose that you also wanted to query the data by *Genre* and *Album Title*. To do this, you could create a global secondary index on *Genre* and *AlbumTitle* and then query the index in much the same way as you'd query the *Music* table.

Figure 4.15 shows the example *Music* table with a new index called *GenreAlbumTitle*. In the index, *Genre* is the partition key, and *AlbumTitle* is the sort key.

**FIGURE 4.15** Amazon DynamoDB table and secondary index



Note the following about the `GenreAlbumTitle` index:

- Every index belongs to a table, which is called the *base table* for the index. In the preceding example, `Music` is the base table for the `GenreAlbumTitle` index.
- DynamoDB maintains indexes automatically. When you add, update, or delete an item in the base table, DynamoDB adds, updates, or deletes the corresponding item in any indexes that belong to that table.
- When you create an index, you specify which attributes will be copied, or *projected*, from the base table to the index. At a minimum, DynamoDB projects the key attributes from the base table into the index. This is the case with `GenreAlbumTitle`, wherein only the key attributes from the `Music` table are projected into the index.

You can query the `GenreAlbumTitle` index to find all albums of a particular genre (for example, all *Hard Rock* albums). You can also query the index to find all albums within a particular genre that have certain album titles (for example, all *Heavy Metal* albums with titles that start with the letter *M*).

### Comparison of Local Secondary Indexes and Global Secondary Indexes

To determine which type of index to use, consider your application's requirements. Table 4.4 shows the main differences between a global secondary index and a local secondary index.

**TABLE 4.4** Comparison of Local and Global Secondary Indexes

Characteristic	Global Secondary Index	Local Secondary Index
<b>Query Scope</b>	Entire table, across all partitions.	Single partition, as specified by the partition key value in the query.
<b>Key Attributes</b>	<ul style="list-style-type: none"> <li>▪ Partition key, or partition and sort key.</li> <li>▪ Can be any scalar attribute in the table.</li> </ul>	<ul style="list-style-type: none"> <li>▪ Partition and sort key.</li> <li>▪ Partition key of index must be the same attribute as base table.</li> </ul>
<b>Projected Attributes</b>	Only projected attributes can be queried.	Can query attributes that are not projected. Attributes are retrieved from the base table.
<b>Read Consistency</b>	Eventual consistency only.	Eventual consistency or strong consistency.

Characteristic	Global Secondary Index	Local Secondary Index
<b>Provisioned Throughput</b>	<ul style="list-style-type: none"><li>▪ Separate throughput settings from base table.</li><li>▪ Consumes separate capacity units.</li></ul>	<ul style="list-style-type: none"><li>▪ Same throughput settings as base table.</li><li>▪ Consumes base table capacity units.</li></ul>
<b>Lifecycle Considerations</b>	Can be created or deleted at any time.	<ul style="list-style-type: none"><li>▪ Must be created when the table is created.</li><li>▪ Can be deleted only when the table is deleted.</li></ul>

## Amazon DynamoDB Streams

*Amazon DynamoDB Streams* is an optional feature that captures data modification events in DynamoDB tables. The data about these events appears in the stream in near real time and in the order that the events occurred.

Each event is represented by a *stream record*. If you enable a stream on a table, DynamoDB Streams writes a stream record whenever one of the following events occurs:

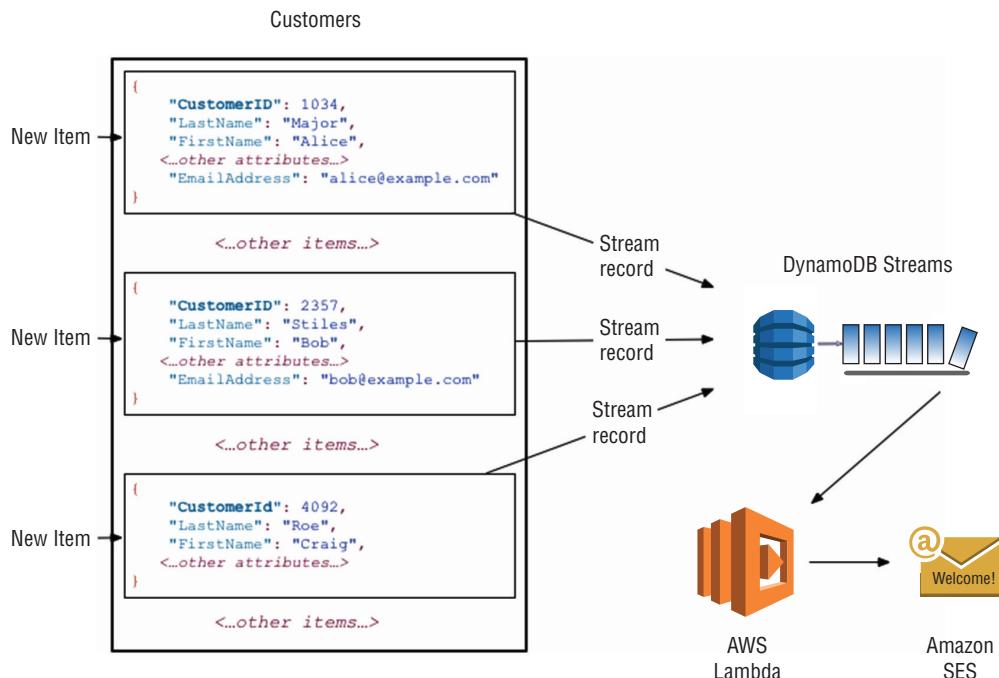
**A new item is added to the table**—The stream captures an image of the entire item, including all of its attributes.

**An item is updated**—The stream captures the “before” and “after” images of any attributes that were modified in the item.

**An item is deleted from the table**—The stream captures an image of the entire item before it was deleted.

Each stream record also contains the name of the table, the event timestamp, and other metadata. Stream records have a lifetime of 24 hours; after that, they are automatically removed from the stream.

Figure 4.16 shows how you can use DynamoDB Streams together with AWS Lambda to create a *trigger*—code that executes automatically whenever an event of interest appears in a stream. For example, consider a *Customers* table that contains customer information for a company. Suppose that you want to send a “welcome” email to each new customer. You could enable a stream on that table and then associate the stream with a Lambda function. The Lambda function would execute whenever a new stream record appears, but only process new items added to the *Customers* table. For any item that has an *EmailAddress* attribute, the Lambda function could invoke Amazon Simple Email Service (Amazon SES) to send an email to that address.

**FIGURE 4.16** Example of Amazon DynamoDB Streams and AWS Lambda

In the example shown in Figure 4.16, the last customer, Craig Roe, will not receive an email because he does not have an EmailAddress.

In addition to triggers, DynamoDB Streams enables other powerful solutions that developers can create, such as the following:

- Data replication within and across AWS regions
- Materialized views of data in DynamoDB tables
- Data analysis by using Amazon Kinesis materialized views

## Read Consistency

DynamoDB replicates data among multiple Availability Zones in a region. When your application writes data to a DynamoDB table and receives an HTTP 200 response (OK), all copies of the data are updated. The data is eventually consistent across all storage locations, usually within 1 second or less. DynamoDB supports both *eventually consistent* and *strongly consistent* reads.

## Eventually Consistent Reads

When you read data from a DynamoDB table immediately after a write operation, the response might not reflect the results of a recently completed write operation. The response might include some stale data. If you repeat your read request after a short time, the response should return the latest data. DynamoDB uses *eventually consistent reads*, unless you specify otherwise.

## Strongly Consistent Reads

When querying data, you can specify whether DynamoDB should return *strongly consistent reads*. When you request a strongly consistent read, DynamoDB returns a response with the most up-to-date data, reflecting updates from all prior write operations that were successful. A strongly consistent read might not be available if there is a network delay or outage.

## Comparison of Consistent Reads

As a developer, it is important to understand the needs of your application. In some applications, eventually consistent reads might be fine, such as a high-score dashboard. In other applications or parts of an application, however, such as a financial or medical system, an eventually consistent read could be an issue. You will want to evaluate your data usage patterns to ensure that you are choosing the right type of reads for each part of your application.

There is an additional cost for strongly consistent reads, and they will have more latency in returning data than an eventually consistent read. So, that cost and timing should also play into your decision.

## Read and Write Throughput

When you create a table or index in DynamoDB, you must specify your capacity requirements for read and write activity. By defining your throughput capacity in advance, DynamoDB can reserve the necessary resources to meet the read and write activity your application requires, while ensuring consistent, low-latency performance. Specify your required throughput value by setting the `ProvisionedThroughput` parameter when you create or update a table.

You specify throughput capacity in terms of read capacity units and write capacity units:

- One *read capacity unit* (RCU) represents one strongly consistent read per second, or two eventually consistent reads per second, for an item up to 4 KB in size. If you need to read an item that is larger than 4 KB, DynamoDB will need to consume additional read capacity units. The total number of read capacity units required depends on the item size and whether you want an eventually consistent or strongly consistent read.
- One *write capacity unit* (WCUs) represents one write per second for an item up to 1 KB in size. If you need to write an item that is larger than 1 KB, DynamoDB must consume additional write capacity units. The total number of write capacity units required depends on the item size.

For example, suppose that you create a table with five read capacity units and five write capacity units. With these settings, your application could do the following:

- Perform strongly consistent reads of up to 20 KB per second ( $4 \text{ KB} \times 5 \text{ read capacity units}$ ).
- Perform eventually consistent reads of up to 40 KB per second (twice as much read throughput).
- Write up to 5 KB per second ( $1 \text{ KB} \times 5 \text{ write capacity units}$ ).

If your application reads or writes larger items (up to the DynamoDB maximum item size of 400 KB), it consumes more capacity units.

If your read or write requests exceed the throughput settings for a table, DynamoDB can throttle that request. DynamoDB can also throttle read requests excess for an index. Throttling prevents your application from consuming too many capacity units. When a request is throttled, it fails with an HTTP 400 code (Bad Request) and a `ProvisionedThroughputExceededException`. The AWS SDKs have built-in support for retrying throttled requests, so you do not need to write this logic yourself.

DynamoDB provides the following mechanisms for managing throughput as it changes:

**Amazon DynamoDB Auto Scaling** DynamoDB automatic scaling actively manages throughput capacity for tables and global secondary indexes. With automatic scaling, you define a range (upper and lower limits) for read and write capacity units. You also define a target utilization percentage within that range. DynamoDB auto scaling seeks to maintain your target utilization, even as your application workload increases or decreases.

**Provisioned throughput** If you aren't using DynamoDB auto scaling, you have to define your throughput requirements manually. As discussed, with this setting you may run into a `ProvisionedThroughputExceededException` if you are throttled. But you can change your throughput with a few clicks.

**Reserved capacity** You can purchase *reserved capacity* in advance, where you pay a one-time upfront fee and commit to a minimum usage level over a period of time. You may realize significant cost savings compared to on-demand provisioned throughput settings.

**On-demand** It can be difficult to plan capacity, especially if you aren't collecting metrics or perhaps are developing a new application and you aren't sure what type of performance you require. With On-Demand mode, your DynamoDB table will automatically scale up or down to any previously reached traffic level. If a workload's traffic level reaches a new peak, DynamoDB rapidly adapts to accommodate the workload. As a developer, focus on making improvements to your application and offload scaling activities to AWS.

## Partitions and Data Distribution

When you are using a table in DynamoDB, the data is placed on multiple partitions (depending on the amount of data and the amount of throughput allocated to it; recall that throughput is determined by RCUs and WCUs). When you allocate RCUs and

WCUs to a table, those RCUs and WCUs are split evenly among all partitions for your table.

For example, suppose that you have allocated 1,000 RCUs and 1,000 WCUs to a table, and this table has 10 partitions allocated to it. Then each partition would have 100 RCUs and 100 WCUs for it to use. If one of your partitions consumes all the RCUs and WCUs for the table, you may receive a `ProvisionedThroughputExceededException` error because one of your partitions is hot. To deal with hot partitions, DynamoDB has two features: burst capacity and adaptive capacity.

### Burst Capacity

The previous example discussed how you had 10 partitions, each with 100 RCUs and 100 WCUs allocated to them. One of your partitions begins to become hot and now needs to consume more than 100 RCUs. Under normal circumstances, you may receive the `ProvisionedThroughputExceededException` error. However, with burst capacity, whenever your partition is not using all of its total capacity, DynamoDB reserves a portion of that unused capacity for later *bursts* of throughput to handle any spike your partition may experience.

At the time of this writing, DynamoDB currently reserves up to 300 seconds (5 minutes) of unused read and write capacity, which means that your partition can handle a peak load for 5 minutes over its normal expected load. Burst capacity is enabled and runs in the background.

### Adaptive Capacity

*Adaptive capacity* is when it is not always possible to distribute read and write activity to a partition evenly. In the example, a partition is experiencing not only peak demand but also consistent demand over and above its normal 100 RCU and 100 WCUs. Suppose that now this partition requires 200 RCUs instead of 100 RCUs.

DynamoDB adaptive capacity enables your application to continue reading and writing to hot partitions without being throttled, provided that the total provisioned capacity for the table is not exceeded. DynamoDB allocates additional RCUs to the hot partition; in this case, 100 more. With adaptive capacity, you will still be throttled for a period of time, typically between 5–30 minutes, before adaptive capacity turns on or activates. So, for a portion of time, your application will be throttled; however, after adaptive capacity allocates the RCUs to the partition, DynamoDB is able to sustain the new higher throughput for your partition and table. Adaptive capacity is on by default, and there is no need to enable or disable it.

## Retrieving Data from DynamoDB

Two primary methods are used to retrieve data from DynamoDB: Query and Scan.

### Query

In DynamoDB, you perform Query operations directly on the table or index. To run the Query command, you must specify, at a minimum, a primary key. If you are querying an index, you must specify both `TableName` and `IndexName`.

The following is a query on a *Music* table in DynamoDB using the Python SDK:

```
import boto3
import json
import decimal

Helper class to convert a DynamoDB item to JSON.
class DecimalEncoder(json.JSONEncoder):
 def default(self, o):
 if isinstance(o, decimal.Decimal):
 if o % 1 > 0:
 return float(o)
 else:
 return int(o)
 return super(DecimalEncoder, self).default(o)

dynamodb = boto3.resource('dynamodb', region_name='us-east-1')

table = dynamodb.Table('Music')

print("A query with DynamoDB")

response = table.query(
 KeyConditionExpression=Key('Artist').eq('Sam Samuel')
)

for i in response['Items']:
 print(i['SongTitle'], "-", i['Genre'], i['Price'])
```

The query returns all of the songs by the artist Sam Samuel in the *Music* table.

## Scan

You can also perform Scan operations on a table or index. The Scan operation returns one or more items and item attributes by accessing every item in a table or a secondary index. To have DynamoDB return fewer items, you can provide a `FilterExpression` operation.

If the total number of scanned items exceeds the maximum dataset size limit of 1 MB, the scan stops, and the results are returned to the user as a `LastEvaluatedKey` value to continue the scan in a subsequent operation. The results also include the number of items exceeding the limit. A scan can result in no table data meeting the filter criteria.

A single Scan operation reads up to the maximum number of items set (if you're using the `Limit` parameter) or a maximum of 1 MB of data and then applies any filtering to the results by using `FilterExpression`. If `LastEvaluatedKey` is present in the response, you must paginate the result set.

Scan operations proceed sequentially; however, for faster performance on a large table or secondary index, applications can request a parallel Scan operation by providing the Segment and TotalSegments parameters.

Scan uses eventually consistent reads when accessing the data in a table; therefore, the result set might not include the changes to data in the table immediately before the operation began. If you need a consistent copy of the data, as of the time that the Scan begins, you can set the ConsistentRead parameter to true.

The following is a scan on a *Movies* table with the Python SDK:

```
// Return all of the data in the index
import boto3
import json
import decimal

Create the DynamoDB Resource
dynamodb = boto3.resource('dynamodb', region_name='us-east-1')

Use the Music Table
table = dynamodb.Table('Music')

Helper class to convert a DynamoDB decimal/item to JSON.
class DecimalEncoder(json.JSONEncoder):
 def default(self, o):
 if isinstance(o, decimal.Decimal):
 if o % 1 > 0:
 return float(o)
 else:
 return int(o)
 return super(DecimalEncoder, self).default(o)

Specify some filters for the scan
Here we are stating that the Price must be between 12 - 30
fe = Key('Price').between(12, 30)
pe = "#g, Price"
Expression Attribute Names for Projection Expression only.
ean = { "#g": "Genre", }

#
response_scan = table.scan(
 FilterExpression=fe,
 ProjectionExpression=pe,
 ExpressionAttributeNames=ean
)

Print all the items
for i in response_scan['Items']:
 print(json.dumps(i, cls=DecimalEncoder))
```

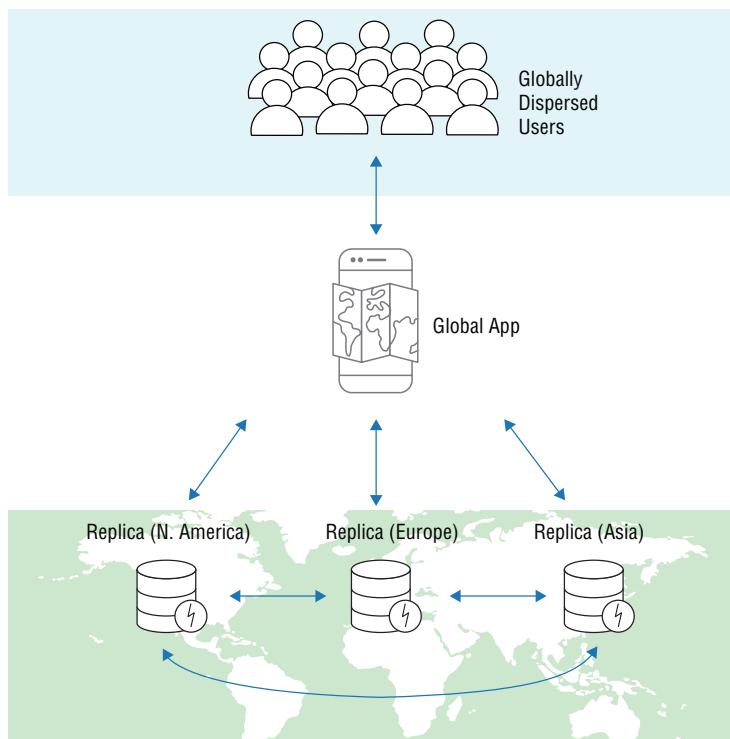
```
while 'LastEvaluatedKey' in response:
 response = table.scan(
 ProjectionExpression=pe,
 FilterExpression=fe,
 ExpressionAttributeNames= ean,
 ExclusiveStartKey=response['LastEvaluatedKey']
)
 for i in response['Items']:
 print(json.dumps(i))
```

As you can see from the Python code, the scan returns all records with a price of between 12 and 30 and the genre and the price. The `LastEvaluatedKey` property is included to continue to loop through the entire table.

## Global Tables

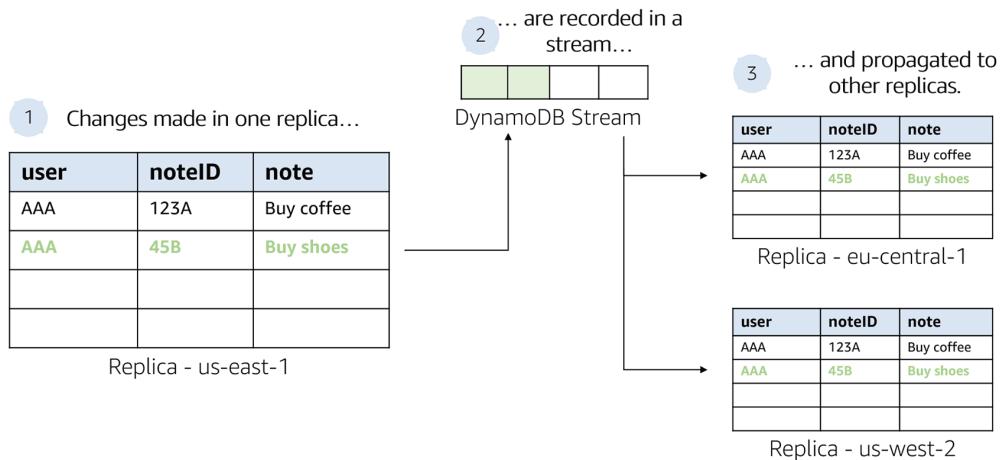
Global tables build upon the DynamoDB global footprint to provide you with a fully managed, multiregion, and multimaster database that provides fast, local, read-and-write performance for massively scaled, global applications. DynamoDB performs all the necessary tasks to create identical tables in these regions and propagate ongoing data changes to all of them. Figure 4.17 shows an example of how global tables can work with a global application and globally dispersed users.

**FIGURE 4.17** Global tables



A *global table* is a collection of one or more DynamoDB tables, all owned by a single AWS account, identified as replica tables. A *replica table* (or replica, for short) is a single DynamoDB table that functions as a part of a global table. Each replica stores the same set of data items. Any given global table can have only one replica table per region, and every replica has the same table name and the same primary key schema. Changes made in one replica are recorded in a stream and propagated to other replicas, as shown in Figure 4.18.

**FIGURE 4.18** Replication flow in global tables



If your application requires strongly consistent reads, then it must perform all of its strongly consistent reads and writes in the same region. DynamoDB does not support strongly consistent reads across AWS Regions.

Conflicts can arise if applications update the same item in different regions at about the same time (concurrent updates). To ensure eventual consistency, DynamoDB global tables use a “last writer wins” reconciliation between concurrent updates whereby DynamoDB makes a best effort to determine the last writer. With this conflict resolution mechanism, all replicas agree on the latest update and converge toward a state in which they all have identical data.

To create a DynamoDB global table, perform the following steps:

1. Create an ordinary DynamoDB table, with DynamoDB Streams enabled, in an AWS Region.
2. Repeat step 1 for every other AWS Region where you want to replicate your data.
3. Define a DynamoDB global table based on the tables that you have created.

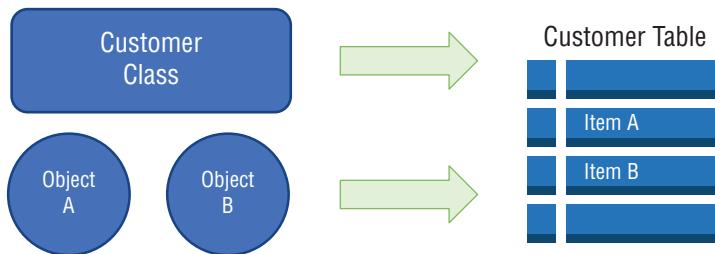
The AWS Management Console automates these tasks so that you can create a global table quickly and easily.

## Object Persistence Model

The DynamoDB object persistence model enables you to map client-side classes to DynamoDB tables. The instances of these classes (objects) map to items in a DynamoDB table.

You can use the object persistence programming interface to connect to DynamoDB; perform create, read, update, and delete operations (CRUD); execute queries; and implement optimistic locking with a version number. Figure 4.19 shows an example of using the object persistence model to map client-side objects in DynamoDB.

**FIGURE 4.19** Object persistence model



Support for the object persistence model is available in the Java and .NET SDKs.

## Amazon DynamoDB Local

DynamoDB Local is the downloadable version of DynamoDB that lets you write and test applications by using the Amazon DynamoDB API without accessing the DynamoDB web service. Instead, the database is self-contained on your computer. When you're ready to deploy your application in production, you can make a few minor changes to the code so that it uses the DynamoDB web service.

Having this local version helps you save on provisioned throughput, data storage, and data transfer fees. In addition, you don't need an internet connection while you're developing your application.

## IAM and Fine-Grained Access Control

You can use AWS IAM to grant or restrict access to DynamoDB resources and API actions. For example, you could allow a user to execute the `GetItem` operation on a *Books* table. DynamoDB also supports fine-grained access control so that you can control access to individual data items and attributes. This means that perhaps you have a *Users* table and you want the specific user to have access only to his or her data. You can accomplish this with fine-grained access control. Use a *condition* inside an IAM policy with the `dynamodb:LeadingKeys` property.

By using `LeadingKeys`, you can limit the user so that they can access only the items where the partition key matches the `userID`. In the following example in the *Users* table,

you want to restrict who can view the profile information to only the user to which the data or profile information belongs:

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "LeadingKeysExample",
 "Effect": "Allow",
 "Action": [
 "dynamodb:GetItem",
 "dynamodb:BatchGetItem",
 "dynamodb:Query",
 "dynamodb:PutItem",
 "dynamodb:UpdateItem",
 "dynamodb:DeleteItem",
 "dynamodb:BatchWriteItem"
],
 "Resource": [
 "arn:aws:dynamodb:us-east-1:accountnumber:table/UserProfiles"
],
 "Condition": {
 "ForAllValues:StringEquals": {
 "dynamodb:LeadingKeys": [
 "${www.amazon.com:user_id}"

],
 "dynamodb:Attributes": [
 "UserId",
 "FirstName",
 "LastName",
 "Email",
 "Birthday"
]
 },
 "StringEqualsIfExists": {
 "dynamodb:Select": "SPECIFIC_ATTRIBUTES"
 }
 }
 }
]
}
```

As you can see in the IAM policy, only the specific user is allowed to access a subset of the total attributes that are defined in the Attributes section of the policy. Furthermore, the SELECT statement specifies that the application must provide a list of specific attributes to act upon, preventing the application from requesting all attributes.

## Backup and Restore

You can create on-demand backups and enable point-in-time recovery for your DynamoDB tables.

*On-demand backups* create full backups of your tables or restore them on-demand at any time. These actions execute with zero impact on table performance or availability and without consuming any provisioned throughput on the table.

*Point-in-time recovery* helps protect your DynamoDB tables from accidental write or delete operations. For example, suppose that a test script accidentally writes to a production DynamoDB table. With point-in-time recovery, you can restore that table to any point in time during the last 35 days. DynamoDB maintains incremental backups of your table. These operations will not affect performance or latency.

## Encryption with Amazon DynamoDB

DynamoDB offers fully managed encryption at rest, and it is enabled by default. DynamoDB uses AWS KMS for encrypting the objects at rest. By default, DynamoDB uses the AWS-owned customer master key (CMK); however, you can also specify your own AWS KMS CMK key that you have created. For more information on AWS KMS, see Chapter 5, “Encryption on AWS.”

## Amazon DynamoDB Best Practices

Now that you understand what DynamoDB is and how you can use it to create a scalable database for your application, review some best practices for using DynamoDB.

### Distribute Workload Evenly

The primary key or partition key portion of a table’s primary key determines the logical partitions in which the table’s data is stored. These logical partitions also affect the underlying physical partitions. As a result, you want to distribute your workload across the partitions as evenly as possible, reducing the number of “hot” partition issues that may arise.

Table 4.5 compares the more common partition key schemas and whether they are good for DynamoDB.

**TABLE 4.5** Amazon DynamoDB Partition Key Recommended Strategies

Partition Key Value	Uniformity
<i>User ID</i> , where the application has many users	Good
<i>Status code</i> , where there are only a few possible status codes	Bad

Partition Key Value	Uniformity
<i>Item creation date</i> , rounded to the nearest time period (for example, day, hour, or minute)	Bad
<i>Device ID</i> , where each device accesses data at relatively similar intervals	Good
<i>Device ID</i> , where even if there are many devices being tracked, one is by far more popular than all the others	Bad

### Comparison of Query and Scan Operations

The Query operation finds items in a table based on primary key values. You must provide the name of the partition key attribute and the value of that attribute. You can provide a sort key attribute name and value to refine the search results (for example, all of the forums with this ID in the last seven days). By default, Query returns all of the data attributes for those items with specified primary keys. The results are sorted by the sort key in ascending order, which can be reversed. Additionally, queries are set to be Eventually Consistent, with an option to change to Strongly Consistent, if necessary.

The Scan operation returns all of the item attributes by accessing every item in the table. It is for this reason that Query is more efficient than the Scan operation.

## Data Warehouse

If you are performing analytics, you may want to use a data warehouse. A *data warehouse* is a central repository of information that you can analyze to make better-informed decisions. Data flows into a data warehouse from transactional systems, relational databases, and other sources, typically on a regular cadence. Business analysts, data scientists, and decision-makers access the data through BI tools, SQL clients, and other analytics applications.

### Data Warehouse Architecture

A data warehouse architecture consists of three tiers. The bottom tier of the architecture is the database server, where data is loaded and stored. The middle tier consists of the analytics engine that is used to access and analyze the data. The top tier is the front-end client that presents results through reporting, analysis, and data mining tools.

A data warehouse works by organizing data into a schema that describes the layout and type of data, such as integer, data field, or string. When data is ingested, it is stored in various tables described by the schema. Query tools use the schema to determine which data tables to access and analyze.

## Data Warehouse Benefits

Benefits of using a data warehouse include the following:

- Better decision-making
- Consolidation of data from many sources
- Data quality, consistency, and accuracy
- Historical intelligence
- Analytics processing that is separate from transactional databases, improving the performance of both systems

The data warehousing landscape has changed dramatically in recent years with the emergence of cloud-based services that offer high performance, simple deployment, near-infinite scaling, and easy administration at a fraction of the cost of on-premises solutions.

## Comparison of Data Warehouses and Databases

A data warehouse is specially designed for data analytics, which involves reading large amounts of data to understand relationships and trends across the data. A database is used to capture and store data, such as recording details of a transaction. Table 4.6 is useful in comparing the characteristics of data warehouses and databases.

**TABLE 4.6** Comparison of Data Warehouse and Database Characteristics

Characteristics	Data Warehouse	Transactional Database
<b>Suitable Workloads</b>	Analytics, reporting, big data	Transaction processing
<b>Data Source</b>	Data collected and normalized from many sources	Data captured as-is from a single source, such as a transactional system
<b>Data Capture</b>	Bulk write operations typically on a predetermined batch schedule	Optimized for continuous write operations as new data is available to maximize transaction throughput
<b>Data Normalization</b>	Denormalized schemas, such as the star schema or snowflake schema	Highly normalized, static schemas
<b>Data Storage</b>	Optimized for simplicity of access and high-speed query performance by using columnar storage	Optimized for high-throughput write operations to a single row-oriented physical block
<b>Data Access</b>	Optimized to minimize I/O and maximize data throughput	High volumes of small read operations

## Comparison of Data Warehouses and Data Lakes

Unlike a data warehouse, a data lake, as described in Chapter 3, “Hello, Storage,” is a centralized repository for all data, including structured and unstructured. A data warehouse uses a predefined schema that is optimized for analytics. In a data lake, the schema is not defined, enabling additional types of analytics, such as big data analytics, full text search, real-time analytics, and machine learning. Table 4.7 compares the characteristics of a data warehouse and a data lake.

**TABLE 4.7** Comparison of Data Warehouse and Data Lake Characteristics

Characteristics	Data Warehouse	Data Lake
<b>Data</b>	Relational data from transactional systems, operational databases, and line-of-business applications	Nonrelational and relational data from IoT devices, websites, mobile apps, social media, and corporate applications
<b>Schema</b>	Designed before the data warehouse implementation (schema-on-write)	Written at the time of analysis (schema-on-read)
<b>Price/Performance</b>	Fastest query results by using higher-cost storage	Query results getting faster by using low-cost storage
<b>Data Quality</b>	Highly curated data that serves as the central version of the truth	Any data that may or may not be curated (in other words, raw data)
<b>Users</b>	Business analysts, data scientists, and data developers	Data scientists, data developers, and business analysts (using curated data)
<b>Analytics</b>	Batch reporting, BI, and visualizations	Machine learning, predictive analytics, data discovery, and profiling

## Comparison of Data Warehouses and Data Marts

A *data mart* is a data warehouse that serves the needs of a specific team or business unit, such as finance, marketing, or sales. It is smaller, is more focused, and may contain summaries of data that best serve its community of users. Table 4.8 compares the characteristics of a data warehouse and a data mart.

**TABLE 4.8** Comparison of Data Warehouse and Data Mart Characteristics

Characteristics	Data Warehouse	Transactional Database
<b>Scope</b>	Centralized, multiple subject areas integrated together	Decentralized, specific subject area
<b>Users</b>	Organization-wide	A single community or department
<b>Data Source</b>	Many sources	A single or a few sources, or a portion of data already collected in a data warehouse
<b>Size</b>	Large—can be 100s of gigabytes to petabytes	Small, generally up to 10s of gigabytes
<b>Design</b>	Top-down	Bottom-up
<b>Data Detail</b>	Complete, detailed data	May hold summarized data

## Amazon Redshift

*Amazon Redshift* is a fast, fully managed, petabyte-scale data warehouse that makes it simple and cost-effective to analyze all your data by using standard SQL and your existing BI tools. With Amazon Redshift, you can run complex analytic queries against petabytes of structured data using sophisticated query optimization, columnar storage on high-performance local disks, and massively parallel query execution. Most results come back in seconds. Amazon Redshift is up to 10 times faster than traditional on-premises data warehouses at 1/10 the cost.

## Architecture

An Amazon Redshift data warehouse is a collection of computing resources called *nodes*, which are organized into a group called a *cluster*. Each cluster runs an Amazon Redshift engine and contains one or more databases. After you provision your cluster, you can upload your dataset and then perform data analysis queries. Each cluster has a leader node and one or more compute nodes, and you have a choice of a hardware platform for your cluster.

## Client Applications

Amazon Redshift integrates with various data loading and extract, transform, and load (ETL) tools and BI reporting, data mining, and analytics tools. It is based on open standard PostgreSQL, so most existing SQL client applications will integrate with Amazon Redshift with only minimal changes. For important differences between Amazon Redshift SQL and PostgreSQL, see the Amazon Redshift documentation.

## Leader Node

The *leader node* acts as the SQL endpoint and receives queries from client applications, parses the queries, and develops query execution plans. The leader node then coordinates a parallel execution of these plans with the compute nodes and aggregates the intermediate results from these nodes. Finally, it returns the results to the client applications. The leader node also stores metadata about the cluster. Amazon Redshift communicates with client applications by using open standard PostgreSQL, JDBC, and ODBC drivers.

## Compute Nodes

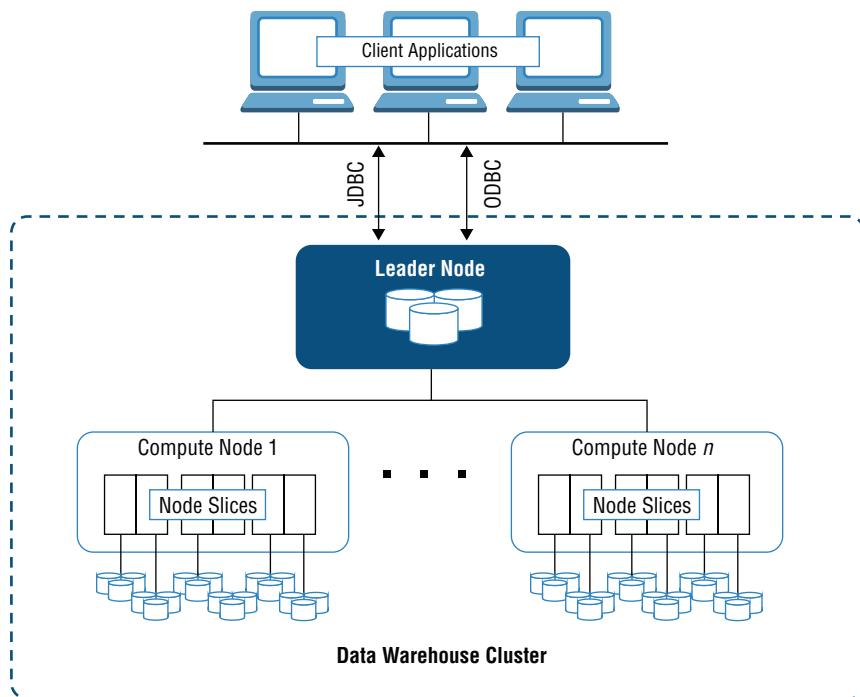
*Compute nodes* execute the query execution plan and transmit data among themselves to serve these queries. The intermediate results are sent to the leader node for aggregation before being sent back to the client applications.

## Node Slices

A compute node is partitioned into slices. Each slice is allocated a portion of the node's memory and disk space, where it processes a portion of the workload assigned to the node. The leader node manages distributing data to the slices and allocates the workload for any queries or other database operations to the slices. The slices then work in parallel to complete the operation. The node size of the cluster determines the number of slices per node.

Figure 4.20 shows the Amazon Redshift data warehouse architecture, including the client applications, JDBC and ODBC connections, leader node, compute nodes, and node slices.

**FIGURE 4.20** Amazon Redshift architecture



## Databases

A cluster contains one or more databases. User data is stored on the compute nodes.

## Hardware Platform Options

When you launch a cluster, one option you specify is the node type. The *node type* determines the CPU, RAM, storage capacity, and storage drive type for each node. There are two categories for node types. The *dense storage* (DS) node types are storage-optimized using large magnetic disks and can provide up to 2 PB of storage capacity. The *dense compute* (DC) node types are compute-optimized. Because they use solid state drive (SSD) storage, they deliver much faster I/O compared to DS node types but provide less storage space at a maximum of 326 TB.

## Table Design

Each database within an Amazon Redshift cluster can support many tables. Like most SQL-based databases, you can create a table using the `CREATE TABLE` command. This command specifies the name of the table, the columns, and their data types. This command also supports specifying compression encodings, distribution strategy, and sort keys in Amazon Redshift.

## Data Types

Each value that Amazon Redshift stores or retrieves has a data type with a fixed set of associated properties. Data types are declared when tables are created. Additional columns can be added to a table by using the `ALTER TABLE` command, but you cannot change the data type on an existing column.

Many familiar data types are available, including the following:

### Numeric data types

- BIGINT
- DECIMAL
- DOUBLE PRECISION
- INTEGER
- REAL
- SMALLINT

### Text data types

- CHAR
- VARCHAR

### Date data types

- DATE
- TIMESTAMP
- TIMESTAMPTZ

## Logical data type

- BOOLEAN

## Compression Encoding

Amazon Redshift uses data compression as one of the key performance optimizations. When you load data for the first time into an empty table, Amazon Redshift samples your data automatically and selects the best compression scheme for each column. Alternatively, you can specify your preferred compression encoding on a per-column basis as part of the `CREATE TABLE` command.

## Distribution Strategy

When you load data into a table, Amazon Redshift distributes the rows of the table to each of the compute nodes according to the table's distribution style. When you execute a query, the query optimizer redistributes the rows to the compute nodes as needed to perform any joins and aggregations. The goal in selecting a table distribution style is to minimize the impact of the redistribution step by locating the data where it needs to be before the query is executed.

This is one of the primary decisions when you're creating a table in Amazon Redshift. You can configure the distribution style of a table to give Amazon Redshift hints as to how the data should be partitioned to meet your query patterns. The style you select for your database affects query performance, storage requirements, data loading, and maintenance. By choosing the best distribution strategy for each table, you can balance your data distribution and significantly improve overall system performance.

When creating a table, you can choose among one of the three distribution styles: EVEN, KEY, or ALL.

**EVEN distribution** Rows are distributed across the slices in a round-robin fashion, regardless of the values in any particular column. It is an appropriate choice when a table does not participate in joins or when there is not a clear choice between KEY distribution or ALL distribution. EVEN is the default distribution type.

**KEY distribution** Rows are distributed according to the values in one column. The leader node attempts to place matching values on the same node slice. Use this style when you will be querying heavily against values of a specific column.

**ALL distribution** A copy of the entire table is distributed to every node. This ensures that every row is collocated for every join in which the table participates. This multiplies the storage required by the number of nodes in the cluster, and it takes much longer to load, update, or insert data into multiple tables. Use this style only for relatively slow-moving tables that are not updated frequently or extensively.

## Sort Keys

Another important decision to make during table creation is choosing the appropriate sort key. Amazon Redshift stores your data on disk in sorted order according to the sort key, and the query optimizer uses sort order when it determines the optimal query plans. Specify an appropriate sort key for the way that your data will be queried, filtered, or joined.

The following are some general guidelines for choosing the best sort key:

- If recent data is queried most frequently, specify the timestamp column as the leading column for the sort key.
- If you do frequent range filtering or equality filtering on one column, specify that column as the sort key.
- If you frequently join a table, specify the join column as both the sort key and the distribution key.

## Loading Data

Loading large datasets can take a long time and consume many computing resources. How your data is loaded can also affect query performance. You can reduce these impacts by using COPY commands, bulk inserts, and staging tables when loading data into Amazon Redshift.



The COPY command loads data in parallel from Amazon S3 or other data sources in a more efficient manner than INSERT commands.

## Querying Data

You can query Amazon Redshift tables by using standard SQL commands, such as using SELECT statements, to query and join tables. For complex queries, you are able to analyze the query plan to choose better optimizations for your specific access patterns.

For large clusters supporting many users, you can configure workload management (WLM) to queue and prioritize queries.

## Snapshots

Amazon Redshift supports snapshots, similar to Amazon RDS. You can create automated and manual snapshots, which are stored in Amazon S3 by using an encrypted Secure Socket Layer (SSL) connection. If you need to restore from a snapshot, Amazon Redshift creates a new cluster and imports data from the snapshot that you specify.

When you restore from a snapshot, Amazon Redshift creates a new cluster and makes it available before all of the data is loaded so that you can begin querying the new cluster immediately. Amazon Redshift will stream data on demand from the snapshot in response to active queries and load all the remaining data in the background.

Achieving proper durability for a database requires more effort and more attention. Even when using Amazon Elastic Block Store (Amazon EBS) volumes, take snapshots on a frozen file system to be consistent. Also, restoring a database might require additional operations other than restoring a volume from a snapshot and attaching it to an Amazon EC2 instance.

## Security

Securing your Amazon Redshift cluster is similar to securing other databases running in the AWS Cloud. To meet your needs, you will use a combination of IAM policies, security groups, and encryption to secure the cluster.

## Encryption

Protecting the data stored in Amazon Redshift is an important aspect of your security design. Amazon Redshift supports encryption of data in transit using SSL-encrypted connections.

You can also enable database encryption for your clusters to help protect data at rest. AWS recommends enabling encryption for clusters that contain sensitive data. You might be required to use encryption depending on the compliance guidelines or regulations that govern your data. Encryption is an optional setting, and it must be configured during the cluster launch. To change encryption on a cluster, you need to create a new cluster and migrate the data.

Amazon Redshift automatically integrates with AWS KMS.



Implement security at every level of your Amazon Redshift architecture, including the infrastructure resources, database schema, data, and network access.

Access to Amazon Redshift resources is controlled at three levels: cluster management, cluster connectivity, and database access. For details on the controls available to help you manage each of these areas, see the Amazon Redshift Cluster Management Guide at <https://docs.aws.amazon.com/redshift/latest/mgmt/welcome.html>.

The following are some best practices for securing your Amazon Redshift deployments:

- Enable and use SSL when connecting to the Amazon Redshift database port.
- Ensure that your data is available only via SSL by setting the `require_ssl` parameter to `true` in the parameter group that is associated with the cluster.
- Use long, random database passwords generated by Amazon Redshift and store them by using a secret management system.
- Enable cluster encryption.
- Secure the S3 bucket by enabling Amazon S3 encryption and configuring access control for Amazon S3.
- Secure the ETL system by enacting access control, auditing/logging, patch management, disk encryption/secure deletion, and SSL connectivity to Amazon S3.
- Secure the BI system by enacting access control, auditing, patching, SSL connectivity to Amazon Redshift, and SSL UI (if applicable).
- Use cluster or VPC security groups to limit Amazon Redshift access only to the necessary IP addresses (for both inbound and outbound flows).
- Enable cluster encryption.

## Amazon Redshift Spectrum

Amazon Redshift also includes *Redshift Spectrum*, allowing you to run SQL queries directly against exabytes of unstructured data in Amazon S3. No loading or transformation is required.

You can use many open data formats, including Apache Avro, CSV, Grok, Ion, JSON, Optimized Row Columnar (ORC), Apache Parquet, RCFfile, RegexSerDe, SequenceFile, TextFile, and TSV. Redshift Spectrum automatically scales query compute capacity based on the data being retrieved, so queries against Amazon S3 run fast, regardless of dataset size.

To use Redshift Spectrum, you need an Amazon Redshift cluster and a SQL client that's connected to your cluster so that you can execute SQL commands. The cluster and the data files in Amazon S3 must be in the same AWS Region.

## In-Memory Data Stores

*In-memory data stores* are used for caching and real-time workloads. AWS provides a variety of in-memory, key-value database options. You can operate your own nonrelational key-value data store in the cloud on Amazon EC2 and Amazon EBS, work with AWS solution providers, or take advantage of fully managed nonrelational services such as *Amazon ElastiCache*.

### Caching

In computing, the data in a *cache* is generally stored in fast-access hardware, such as random-access memory (RAM), and may also be used in correlation with a software component. A cache's primary purpose is to increase data retrieval performance by reducing the need to access the underlying slower storage layer.

Trading off capacity for speed, a cache typically stores a subset of data transiently, in contrast to databases whose data is usually complete and durable.

### Benefits of Caching

A cache provides high-throughput, low-latency access to commonly accessed application data by storing the data in memory. Caching can improve the speed of your application. Caching reduces the response latency, which improves a user's experience with your application.

Time-consuming database queries and complex queries often create bottlenecks in applications. In read-intensive applications, caching can provide large performance gains by reducing application processing time and database access time. Write-intensive applications typically do not see as great a benefit to caching. However, even write-intensive applications normally have a read/write ratio greater than 1, which implies that read caching can be beneficial. In summary, the benefits of caching include the following:

- Improve application performance
- Reduce database cost
- Reduce load on the backend database tier
- Facilitate predictable performance

- Eliminate database hotspots
- Increase read throughput (IOPS)

The following types of information or applications can often benefit from caching:

- Results of database queries
- Results of intensive calculations
- Results of remote API calls
- Compute-intensive workloads that manipulate large datasets, such as high-performance computing simulations and recommendation engines

Consider caching your data if the following conditions apply:

- It is slow or expensive to acquire when compared to cache retrieval.
- It is accessed with sufficient frequency.
- Your data or information for your application is relatively static
- Your data or information for your application is rapidly changing and staleness is not significant.

## Caching Strategies

You can implement different caching strategies for your application. Two primary methods are lazy loading and write through. A *cache hit* occurs when the cache contains the information requested. A *cache miss* occurs when the cache does not contain the information requested.

**Lazy loading** *Lazy loading* is a caching strategy that loads data into the cache only when necessary. When your application requests data, it first makes the request to the cache. If the data exists in the cache (a cache hit), it is retrieved; but if it does not or has expired (a cache miss), then the data is retrieved from your data store and then stored in the cache.

The advantage of lazy loading is that only the requested data is cached. The disadvantage is that there is a cache miss penalty resulting in three trips:

1. The application requests data from the cache.
2. If there is a cache miss, you must query the database.
3. After data retrieval, the cache is updated.

**Write through** The *write-through* strategy adds data or updates in the cache whenever data is written to the database. The advantage of write through is that the data in the cache is never stale. The disadvantage is that there is a write penalty because every write involves two trips: a write to the cache and a write to the database. Another disadvantage is that because most data is never read in many applications, the data or information that is stored in the cluster is never used. This storage incurs a cost for space and overhead due to the duplicate data. In addition, if your data is updated frequently, the cache may be updating often, causing cache churn.

## In-Memory Key-Value Store

An *in-memory key-value store* is a NoSQL database optimized for read-heavy application workloads (such as social networking, gaming, media sharing, and Q&A portals) or compute-intensive workloads (such as a recommendation engine). In-memory caching improves application performance by storing critical pieces of data in memory for low-latency access. Cached information may include the results of I/O-intensive database queries or the results of computationally intensive calculations.

### Benefits of In-Memory Data Stores

The strict performance requirements imposed by real-time applications mandate more efficient databases. Traditional databases rely on disk-based storage. A single user action may consist of multiple database calls. As they accumulate, latency increases. However, by accessing data in memory, in-memory data stores provide higher throughput and lower latency. In fact, in-memory data stores can be one to two orders of magnitude faster than disk-based databases.

As a NoSQL data store, an in-memory data store does not share the architectural limitations found in traditional relational databases. NoSQL data stores are built to be scalable. Traditional relational databases use a rigid table-based architecture. Some NoSQL data stores use a key-value store and therefore don't enforce a structure on the data. This enables scalability and makes it easier to grow, partition, or shard data as data stores grow. When consumed as a cloud-based service, an in-memory data store also provides availability and cost benefits. On-demand access allows organizations to scale their applications as needed in response to demand spikes and at a lower cost than disk-based stores. Using managed cloud services also eliminates the need to administer infrastructure. Database hotspots are reduced, and performance becomes more predictable. Some cloud-based services also offer the benefit of high availability with replicas and support for multiple Availability Zones.

### Benefits of Distributed Cache

A caching layer helps further drive throughput for read-heavy applications. A *caching layer* is a high-speed storage layer that stores a subset of data. When a read request is sent, the caching layer checks to determine whether it has the answer. If it doesn't, the request is sent on to the database. Meeting read requests through the caching layer in this manner is more efficient and delivers higher performance than what can be had from a traditional database alone.

It is also more cost-effective. A single node of in-memory cache can deliver the same read throughput as several database nodes. Instead of provisioning additional instances of your traditional database to accommodate a demand spike, you can drive more throughput by adding one node of distributed cache, replacing several database nodes. The caching layer saves you money because you're paying for one node instead of multiple database nodes, and you get the added benefit of dramatically faster performance for reads.

## Amazon ElastiCache

Developers need a way to maintain super-low latency, even as they accommodate spikes in demand and while controlling infrastructure and database costs and load.

*Amazon ElastiCache* is a web service that makes it easy to deploy, operate, and scale an in-memory cache in the AWS Cloud. The service improves the performance of web applications by allowing you to retrieve information from fast, managed, in-memory caches instead of relying entirely on slower disk-based databases.

ElastiCache automatically detects and replaces failed nodes, reducing the overhead associated with self-managed infrastructures and also provides a resilient system that mitigates the risk of overloaded cloud databases, which slow website and application load times.

ElastiCache currently supports two different open-source, in-memory, key-value caching engines: Redis and Memcached. Each engine provides some advantages.

### Redis

*Redis* is an increasingly popular open-source, key-value store that supports more advanced data structures, such as sorted sets, hashes, and lists. Unlike Memcached, Redis has disk persistence built in, meaning that you can use it for long-lived data. Redis also supports replication, which can be used to achieve Multi-AZ redundancy, similar to Amazon RDS.

### Memcached

*Memcached* is a widely adopted in-memory key store. It is historically the gold standard of web caching. ElastiCache is protocol-compliant with Memcached, and it is designed to work with popular tools that you use today with existing Memcached environments. Memcached is also multithreaded, meaning that it makes good use of larger Amazon EC2 instance sizes with multiple cores.

### Comparison of Memcached and Redis

Although both Memcached and Redis appear similar on the surface, in that they are both in-memory key stores, they are quite different in practice. Because of the replication and persistence features of Redis, ElastiCache manages Redis more as a relational database. Redis ElastiCache clusters are managed as stateful entities that include failover, similar to how Amazon RDS manages database failover.

Conversely, because Memcached is designed as a pure caching solution with no persistence, ElastiCache manages Memcached nodes as a pool that can grow and shrink, similar to an Amazon EC2 Auto Scaling group. Individual nodes are expendable, and ElastiCache provides additional capabilities here, such as automatic node replacement and Auto Discovery.

Consider the following requirements when deciding between Memcached and Redis.  
Use Memcached if you require one or more of the following:

- Object caching is your primary goal, for example, to offload your database.
- You are interested in as simple a caching model as possible.

- You plan to run large cache nodes and require multithreaded performance with the use of multiple cores.

- You want to scale your cache horizontally as you grow.

Use Redis if you require one or more of the following:

- You are looking for more advanced data types, such as lists, hashes, and sets.
- Sorting and ranking datasets in memory help you, such as with leaderboards.
- Your application requires publish and subscribe (pub/sub) capabilities.
- Persistence of your key store is important.
- You want to run in multiple Availability Zones (Multi-AZ) with failover.
- You want transactional support, which lets you execute a group of commands as an isolated and atomic operation.

## Amazon DynamoDB Accelerator

*Amazon DynamoDB Accelerator* (DAX) is a fully managed, highly available, in-memory cache for DynamoDB that delivers up to 10 times the performance improvement—from milliseconds to microseconds—even at millions of requests per second. DAX does all of the heavy lifting required to add in-memory acceleration to your DynamoDB tables, without requiring developers to manage cache invalidation, data population, or cluster management.

With DAX, you can focus on building great applications for your customers without worrying about performance at scale. You do not need to modify application logic, because DAX is compatible with existing DynamoDB API calls. You can enable DAX with a few clicks in the AWS Management Console or by using the AWS SDK, and you pay only for the capacity you provision.

## Graph Databases

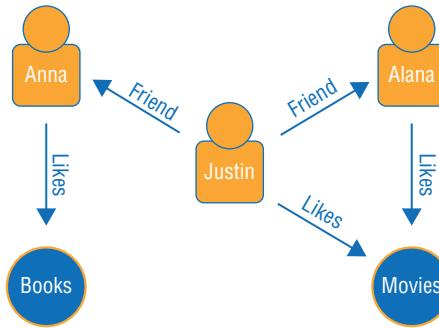
AWS provides a variety of graph database options, such as Amazon Neptune, or you can operate your own graph database in the cloud on Amazon EC2 and Amazon EBS. This section takes a closer look at what exactly is a graph database and when you would want to use one.

Many applications being built today must understand and navigate relationships between highly connected data. This can enable use cases like the following:

- Social applications
- Recommendation engines
- Fraud detection
- Knowledge graphs
- Life sciences
- IT/network

Because the data is highly connected, it is easily represented as a graph, which is a data structure that consists of *vertices* and directed links called *edges*. Vertices and edges can each have properties associated with them. Figure 4.21 depicts a simple graph of relationships between friends and their interests—or social network—that could be stored and queried by using a graph database. A *graph database* is optimized to store and process graph data.

**FIGURE 4.21** Example of a social network diagram



AWS provides a variety of graph database options. You can operate your own graph database in the cloud on Amazon EC2 and Amazon EBS. You can also use Neptune, a fully managed graph database service.

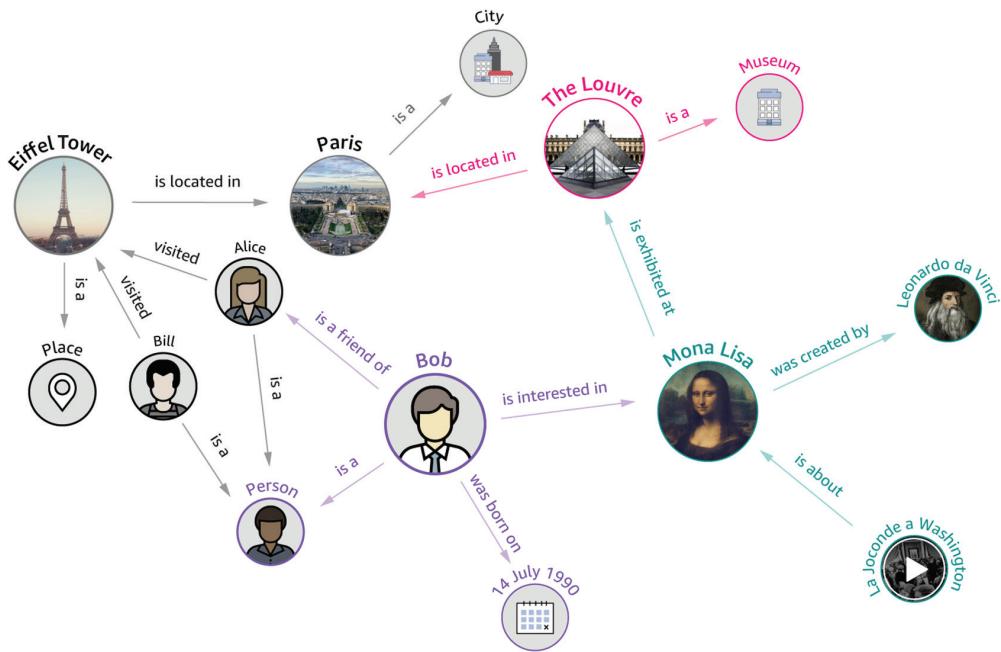
## Amazon Neptune

*Amazon Neptune* is a fast, reliable, fully managed graph database service that makes it easy to build and run applications that work with highly connected datasets. The core of Neptune is a purpose-built, high-performance graph database engine optimized for storing billions of relationships and querying the graph with milliseconds latency.

Neptune is highly available and provides the following features:

- Read replicas
- Point-in-time recovery
- Continuous backup to Amazon S3
- Replication across Availability Zones
- Encryption at rest and in transit

Figure 4.22 shows a knowledge graph that can be powered by Neptune.

**FIGURE 4.22** Example of a graph database architecture running on Amazon Neptune

Neptune supports the popular graph models Property Graph and W3C's RDS and their respective query languages Apache TinkerPop Gremlin and SPARQL. With these models, you can easily build queries that efficiently navigate highly connected datasets. Neptune graph databases include the following use cases:

- Recommendation engines
- Fraud detection
- Knowledge graphs
- Drug discovery
- Network security

## Cloud Database Migration

Data is the cornerstone of successful cloud application deployments. Your evaluation and planning process may highlight the physical limitations inherent to migrating data from on-premises locations into the cloud. Amazon offers a suite of tools to help you move data via networks, roads, and technology partners.

This chapter focuses on the AWS Database Migration Service (AWS DMS) and the AWS Schema Conversion Tool (AWS SCT). Customers also use other AWS services and features

that are discussed in Chapter 3, “Hello, Storage,” for cloud data migration, including the following:

- AWS Direct Connect (DX)
- AWS Snowball
- AWS Snowball Edge
- AWS Snowmobile
- AWS Import/Export Disk
- AWS Storage Gateway
- Amazon Kinesis Data Firehose
- Amazon S3 Transfer Acceleration
- Virtual private network (VPN) connections

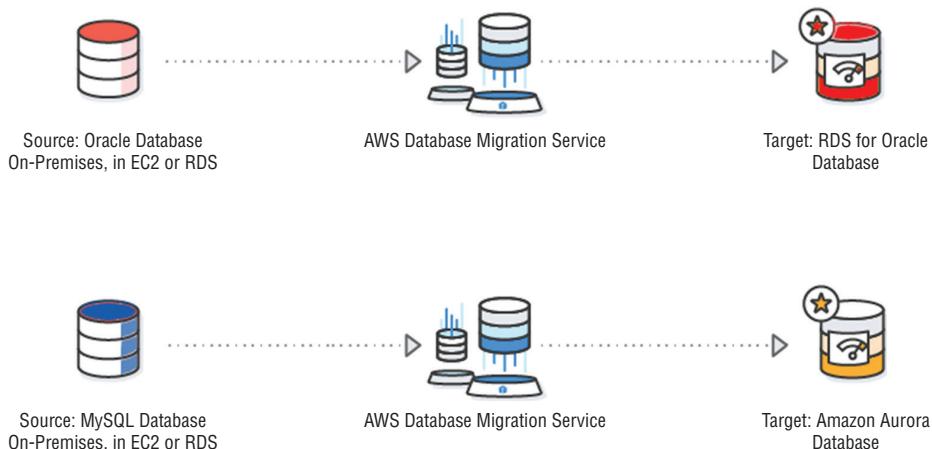
## AWS Database Migration Service

*AWS Database Migration Service* (AWS DMS) helps you migrate databases to AWS quickly and securely. The source database remains fully operational during the migration, minimizing downtime to applications that rely on the database. AWS DMS can migrate your data to and from the most widely used commercial and open-source databases.

The service supports *homogenous database migrations*, such as Oracle to Oracle, in addition to *heterogeneous migrations* between different database platforms, such as Oracle to Amazon Aurora or Microsoft SQL Server to MySQL. You can also stream data to Amazon Redshift, Amazon DynamoDB, and Amazon S3 from any of the supported sources, such as Amazon Aurora, PostgreSQL, MySQL, MariaDB, Oracle Database, SAP ASE, SQL Server, IBM DB2 LUW, and MongoDB, enabling consolidation and easy analysis of data in a petabyte-scale data warehouse. You can also use AWS DMS for continuous data replication with high availability.

Figure 4.23 shows an example of both heterogeneous and homogenous database migrations.

**FIGURE 4.23** Homogenous database migrations using AWS DMS



To perform a database migration, AWS DMS connects to the source data store, reads the source data, and formats the data for consumption by the target data store. It then loads the data into the target data store. Most of this processing happens in memory, though large transactions might require some buffering to disk. Cached transactions and log files are also written to disk.

At a high level, when you're using AWS DMS, complete the following tasks:

- Create a replication server.
  - Create source and target endpoints that have connection information about your data stores.
  - Create one or more tasks to migrate data between the source and target data stores.
- A task can consist of three major phases:
- The full load of existing data
  - The application of cached changes
  - Ongoing replication

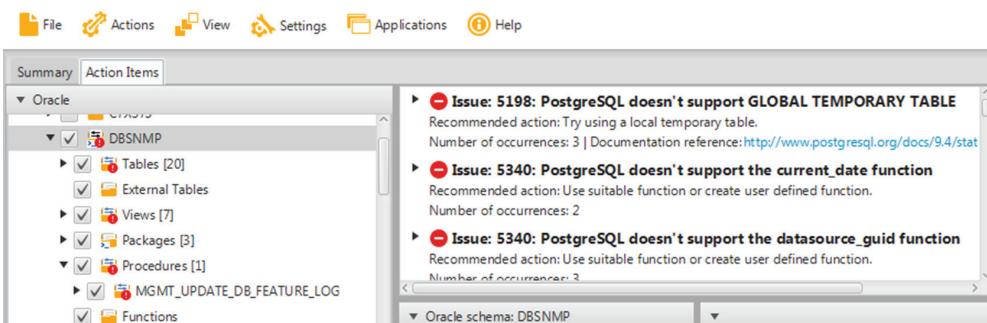
## AWS Schema Conversion Tool

For heterogeneous database migrations, AWS DMS uses the *AWS Schema Conversion Tool* (AWS SCT). AWS SCT makes heterogeneous database migrations predictable by automatically converting the source database schema and a majority of the database code objects, including views, stored procedures, and functions, to a format compatible with the target database. Any objects that cannot be automatically converted are clearly marked so that they can be manually converted to complete the migration.

AWS SCT can also scan your application source code for embedded SQL statements and convert them as part of a database schema conversion project. During this process, AWS SCT performs cloud-native code optimization by converting legacy Oracle and SQL Server functions to their equivalent AWS service, thus helping you modernize the applications at the same time as database migration.

Figure 4.24 is snapshot of the Action Items tab in the AWS SCT report, which shows the items that the tool could not convert automatically. These are the items that you would need to evaluate and adjust manually as needed. The report helps you to determine how much work you would need to do to complete a conversion.

**FIGURE 4.24** AWS SCT action items



After the schema conversion is complete, AWS SCT can help migrate data from a range of data warehouses to Amazon Redshift by using built-in data migration agents.

Your source database can be on-premises, in Amazon RDS, or in Amazon EC2, and the target database can be in either Amazon RDS or Amazon EC2. AWS SCT supports a number of different heterogeneous conversions. Table 4.9 lists the source and target databases that are supported at the time of this writing.

**TABLE 4.9** Source and Target Databases Supported by AWS SCT

Source Database	Target Database on Amazon RDS
Oracle Database	Amazon Aurora, MySQL, PostgreSQL, Oracle
Oracle Data Warehouse	Amazon Redshift
Azure SQL	Amazon Aurora, MySQL, PostgreSQL
Microsoft SQL Server	Amazon Aurora, Amazon Redshift, MySQL, PostgreSQL
Teradata	Amazon Redshift
IBM Netezza	Amazon Redshift
IBM DB2 LUW	Amazon Aurora, MySQL, PostgreSQL
HPE Vertica	Amazon Redshift
MySQL and MariaDB	PostgreSQL
PostgreSQL	Amazon Aurora, MySQL
Amazon Aurora	PostgreSQL
Greenplum	Amazon Redshift
Apache Cassandra	Amazon DynamoDB

## Running Your Own Database on Amazon Elastic Compute Cloud

This chapter focused heavily on the AWS services that are available from a managed database perspective. However, it is important to know that you can also run your own unmanaged database on Amazon EC2, not only for the exam but for managing projects in the real

world. For example, if you want to run MongoDB on Amazon EC2, this is perfectly within the realm of possibility. However, by doing so, you lose the many benefits of using a managed database service.

## Compliance and Security

AWS includes various methods to provide security for your databases and meet the strictest of compliance standards. You can use the following:

- Network isolation through virtual private cloud (VPC)
- Security groups
- AWS resource-level permission controls that are IAM-based.
- Encryption at rest by using AWS KMS or Oracle/Microsoft Transparent Data Encryption (TDE)
- Secure Sockets Layer (SSL) protection for data in transit
- Assurance programs for finance, healthcare, government, and more

## AWS Identity and Access Management

You can use *Identity and Access Management* (IAM) to perform governed access to control who can perform actions with Amazon Aurora MySQL and Amazon RDS for MySQL. Here's an example:

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "AllowCreateDBInstanceOnly",
 "Effect": "Allow",
 "Action": [
 "rds>CreateDBInstance"
],
 "Resource": [
 "arn:aws:rds:*:123456789012:db:test*",
 "arn:aws:rds: * : 123456789012:og:default*",
 "arn:aws:rds:*:123456789012:pg:default**",
 "arn:aws:rds: * : 1234 56789012 :subgrp: default"
],
 }
]
}
```

```
 "Condition": {
 "StringEquals": {
 "rds:DatabaseEngine": "mysql",
 "rds:DatabaseClass": "db.t2.micro"
 }
 }
 }
}
```

## Summary

In this chapter, you learned the basic concepts of different types of databases, including relational, nonrelational, data warehouse, in-memory, and graph databases. From there, you learned about the various managed database services available on AWS. These included Amazon RDS, Amazon DynamoDB, Amazon Redshift, Amazon ElastiCache, and Amazon Neptune. You also saw how you can run your own database on Amazon EC2. Finally, you looked at how to perform homogenous database migrations using the AWS Database Migration Service (AWS DMS). For heterogeneous database migrations, you learned that AWS DMS can use the AWS Schema Conversion Tool (AWS SCT).

## Exam Essentials

**Know what a relational database is.** A relational database consists of one or more tables. Communication to and from relational databases usually involves simple SQL queries, such as “Add a new record” or “What is the cost of product x?” These simple queries are often referred to as online transaction processing (OLTP).

**Know what a nonrelational database is.** Nonrelational databases do not have a hard-defined data schema. They can use a variety of models for data management, such as in-memory key-value stores, graph data models, and document stores. These databases are optimized for applications that have a large data volume, require low latency, and have flexible data models. In nonrelational databases, there is no concept of foreign keys.

**Understand the database options available on AWS.** You can run all types of databases on AWS. You should understand that there are managed and unmanaged options available, in addition to relational, nonrelational, caching, graph, and data warehouses.

**Understand which databases Amazon RDS supports.** Amazon RDS currently supports six relational database engines:

- Microsoft SQL Server
- MySQL
- Oracle
- PostgreSQL
- MariaDB
- Amazon Aurora

**Understand the operational benefits of using Amazon RDS.** Amazon RDS is an AWS managed service. AWS is responsible for patching, antivirus, and the management of the underlying guest OS for Amazon RDS. Amazon RDS greatly simplifies the process of setting a secondary slave with replication for failover and setting up read replicas to offload queries.

**Remember that you cannot access the underlying OS for Amazon RDS DB instances.** You cannot use Remote Desktop Protocol (RDP) or SSH to connect to the underlying OS. If you need to access the OS, install custom software or agents. If you want to use a database engine that Amazon RDS does not support, consider running your database on an Amazon EC2 instance instead.

**Understand that Amazon RDS handles Multi-AZ failover for you.** If your primary Amazon RDS instance becomes unavailable, AWS fails over to your secondary instance in another Availability Zone automatically. This failover is done by pointing your existing database endpoint to a new IP address. You do not have to change the connection string manually; AWS handles the DNS changes automatically.

**Remember that Amazon RDS read replicas are used for scaling out and increased performance.** This replication feature makes it easy to scale out your read-intensive databases. Read replicas are currently supported in Amazon RDS for MySQL, PostgreSQL, and Amazon Aurora. You can create one or more replicas of a database within a single AWS Region or across multiple AWS Regions. Amazon RDS uses native replication to propagate changes made to a source DB instance to any associated read replicas. Amazon RDS also supports cross-region read replicas to replicate changes asynchronously to another geography or AWS Region.

**Know how to calculate throughput for Amazon DynamoDB.** Remember that one read capacity unit (RCU) represents one strongly consistent read per second or two eventually consistent reads per second for an item up to 4 KB in size. For writing data, know that one write capacity unit (WCU) represents one write per second for an item up to 1 KB in size. Be comfortable performing calculations to determine the appropriate setting for the RCU and WCU for a table.

**Know that DynamoDB spreads RCUs and WCUs across partitions evenly.** Recall that when you allocate your total RCUs and WCUs to a table, DynamoDB spreads these across

your partitions evenly. For example, if you have 1,000 RCUs and you have 10 partitions, then you have 100 RCUs allocated to each partition.

**Know the differences between a local secondary index and a global secondary index.**

Remember that you can create local secondary indexes only when you initially create the table; additionally, know that local secondary indexes must share the same partition key as the parent or source table. Conversely, you can create global secondary indexes at any time, with different partitions keys or sort keys.

**Know the difference between eventually consistent and strongly consistent reads.** Know that with eventually consistent reads, your application may retrieve data that is stale; but with strongly consistent reads, the data is always up-to-date.

**Understand the purpose of caching data and which related services are available.** Know why caching is important for your database tier and how it helps to improve your application performance. Additionally, understand the differences between the caching methods (lazy loading and write-through) and the corresponding AWS services (Amazon DynamoDB Accelerator (DAX), ElastiCache for Redis, and ElastiCache for Memcached).

# Resources to Review

What Is a Relational Database?

<https://aws.amazon.com/relational-database/>

AWS Databases:

<https://aws.amazon.com/products/databases/>

AWS Database Blog:

<https://aws.amazon.com/blogs/database/>

A One Size Fits All Database Doesn't Fit Anyone:

<https://www.allthingsdistributed.com/2018/06/purpose-built-databases-in-aws.html>

Amazon Relational Database Service (Amazon RDS) User Guide:

<https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Welcome.html>

Amazon RDS FAQs:

<https://aws.amazon.com/rds/faqs/>

Development and Test on Amazon Web Services:

<https://d1.awsstatic.com/whitepapers/aws-development-test-environments.pdf>

Amazon Redshift Snapshots:

<https://docs.aws.amazon.com/redshift/latest/mgmt/working-with-snapshots.html>

Amazon Aurora:

<https://aws.amazon.com/rds/aurora/>

Amazon Aurora Overview:

[https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/CHAP\\_AuroraOverview.html](https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/CHAP_AuroraOverview.html)

Amazon RDS Resources:

<https://aws.amazon.com/rds/developer-resources/>

Best Practices for Amazon RDS:

[https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP\\_BestPractices.html](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_BestPractices.html)

What Is a Document Database?

<https://aws.amazon.com/nosql/document/>

What Is a Columnar Database?

<https://aws.amazon.com/nosql/columnar/>

What Is NoSQL?

<https://aws.amazon.com/nosql/>

Amazon DynamoDB:

<https://aws.amazon.com/dynamodb/>

What Is Amazon DynamoDB?

<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html>

Amazon DynamoDB Core Components:

<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.CoreComponents.html>

Amazon DynamoDB Developer Guide:

<http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/>

GSI Attribute Projections:

<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GSI.html#GSI.Projections>

Data Warehouse Concepts:

<https://aws.amazon.com/data-warehouse/>

Getting Started with Amazon Redshift:

<http://docs.aws.amazon.com/redshift/latest/gsg/>

Amazon Redshift Database Developer Guide:

<http://docs.aws.amazon.com/redshift/latest/dg/>

Using Amazon Redshift Spectrum to Query External Data:

<https://docs.aws.amazon.com/redshift/latest/dg/c-using-spectrum.html>

What Is a Key-Value Database?

<https://aws.amazon.com/nosql/key-value/>

Amazon ElasticCache for Redis User Guide:

<https://docs.aws.amazon.com/AmazonElastiCache/latest/red-ug/WhatIs.html>

Amazon ElastiCache for Memcached User Guide:

<https://docs.aws.amazon.com/AmazonElastiCache/latest/mem-ug/WhatIs.html>

In-Memory Processing in the Cloud with Amazon ElastiCache (Whitepaper):

[https://d1.awsstatic.com/elasticache/elasticache\\_in\\_memory\\_processing\\_intel.pdf](https://d1.awsstatic.com/elasticache/elasticache_in_memory_processing_intel.pdf)

Performance at Scale with Amazon ElastiCache (Whitepaper):

<https://d1.awsstatic.com/whitepapers/performance-at-scale-with-amazon-elasticache.pdf>

Amazon DynamoDB Accelerator (DAX):

<https://aws.amazon.com/dynamodb/dax/>

Amazon DynamoDB Accelerator (DAX): A Read-Through/Write-Through Cache for DynamoDB:

<https://aws.amazon.com/blogs/database/amazon-dynamodb-accelerator-dax-a-read-throughwrite-through-cache-for-dynamodb/>

What Is a Graph Database?

<https://aws.amazon.com/nosql/graph/>

Amazon Neptune User Guide:

<https://docs.aws.amazon.com/neptune/latest/userguide/intro.html>

AWS Database Migration Service Documentation:

<https://docs.aws.amazon.com/dms/index.html>

Cloud Data Migration:

<https://aws.amazon.com/cloud-data-migration/>

AWS Database Migration Service User Guide:

<http://docs.aws.amazon.com/dms/latest/userguide/>

AWS Database Migration Service Step-by-Step Walkthroughs:

<http://docs.aws.amazon.com/dms/latest/sbs/DMS-SBS-Welcome.html>

AWS Schema Conversion Tool User Guide:

[https://docs.aws.amazon.com/SchemaConversionTool/latest/userguide/CHAP\\_Welcome.html](https://docs.aws.amazon.com/SchemaConversionTool/latest/userguide/CHAP_Welcome.html)

# Exercises

In the following exercises, you will launch two types of databases: the first database is an SQL database on Amazon RDS, and the second is Amazon DynamoDB (NoSQL). For these sets of exercises, you will use the Python 3 SDK. You can download the Python 3 SDK at <https://aws.amazon.com/sdk-for-python/>.

## EXERCISE 4.1

### Create a Security Group for the Database Tier on Amazon RDS

Before you can create your first Amazon RDS database, you must create a security group so that you can allow traffic from your development server to communicate with the database tier. To do this, you must use an Amazon EC2 client to create the security group. Security groups are a component of the Amazon EC2 service, even though you can use them as part of Amazon RDS to secure your database tier.

To create the security group, run the following script:

```
Excercise 4.1
import boto3
import json
import datetime

Let's create some variables we'll use throughout these Excercises in Chapter 4
NOTE: Here we are using a CIDR range for incoming traffic. We have set it to
0.0.0.0/0 which means
ANYONE on the internet can access your database if they have the username and
the password
If possible, specify you're own CIDR range. You can figure out your CIDR range
by visiting the following link
https://www.google.com/search?q=what+is+my+ip
In the variable don't forget to add /32!
If you aren't sure, leave it open to the world

Variables
sg_name = 'rds-sg-dev-demo'
sg_description = 'RDS Security Group for AWS Dev Study Guide'
my_ip_cidr = '0.0.0.0/0'

Create the EC2 Client to create the Security Group for your Database
ec2_client = boto3.client('ec2')

First we need to create a security group
```

```
response = ec2_client.create_security_group(
 Description=sg_description,
 GroupName=sg_name)
print(json.dumps(response, indent=2, sort_keys=True))
Now add a rule for the security group
response = ec2_client.authorize_security_group_ingress(
 CidrIp=my_ip_cidr,
 FromPort=3306,
 GroupName=sg_name,
 ToPort=3306,
 IpProtocol='tcp'
)
print("Security Group should be created! Verify this in the AWS Console.")
```

After running the Python code, verify that the security group was created successfully from the AWS Management Console. You can find this confirmation under the VPC or Amazon EC2 service.

---

## EXERCISE 4.2

### Spin Up the MariaDB Database Instance

Use the Python SDK to spin up your MariaDB database hosted on Amazon RDS.

To spin up the MariaDB database, run the following script and update the Variables section to meet your needs:

```
Excercise 4.2
import boto3
import json
import datetime

Just a quick helper function for date time conversions, in case you want to
print the raw JSON
def date_time_converter(o):
 if isinstance(o, datetime.datetime):
 return o.__str__()

Variables
sg_name = 'rds-sg-dev-demo'
rds_identifier = 'my-rds-db'
db_name = 'mytestdb'
```

*(continued)*

**EXERCISE 4.2 (*continued*)**

```
user_name = 'masteruser'
user_password = 'mymasterpassw0rd1!'
admin_email = 'myemail@myemail.com'
sg_id_number = ''
rds_endpoint = ''

We need to get the Security Group ID Number to use in the creation of the RDS
Instance
ec2_client = boto3.client('ec2')
response = ec2_client.describe_security_groups(
 GroupNames=[
 sg_name
]
)

sg_id_number = json.dumps(response['SecurityGroups'][0]['GroupId'])
sg_id_number = sg_id_number.replace('"','')

Create the client for Amazon RDS
rds_client = boto3.client('rds')

This will create our MariaDB Database
NOTE: Here we are hardcoding passwords for simplicity and testing purposes
only! In production
you should never hardcode passwords in configuration files/code!
NOTE: This will create an MariaDB Database. Be sure to remove it when you are
done.
response = rds_client.create_db_instance(
 DBInstanceIdentifier=rds_identifier,
 DBName=db_name,
 DBInstanceClass='db.t2.micro',
 Engine='mariadb',
 MasterUsername='masteruser',
 MasterUserPassword='mymasterpassw0rd1!',
 VpcSecurityGroupIds=[
 sg_id_number
],
 AllocatedStorage=20,
 Tags=[

]
```

```
'Key': 'POC-Email',
'Value': admin_email
},
{
 'Key': 'Purpose',
 'Value': 'AWS Developer Study Guide Demo'
}
]
)

We need to wait until the DB Cluster is up!
print('Creating the RDS instance. This may take several minutes...')
waiter = rds_client.get_waiter('db_instance_available')
waiter.wait(DBInstanceIdentifier=rds_identifier)

print('Okay! The Amazon RDS Database is up!')
```

After the script has executed, the following message is displayed:

```
Creating the RDS instance. This may take several minutes.
```

After the Amazon RDS database instance has been created successfully, the following confirmation is displayed:

```
Okay! The Amazon RDS Database is up!
```

You can also view these messages from the Amazon RDS console.

---

### EXERCISE 4.3

#### Obtain the Endpoint Value for the Amazon RDS Instance

Before you can start using the Amazon RDS instance, you must first specify your endpoint. In this exercise, you will use the Python SDK to obtain the value.

To obtain the Amazon RDS endpoint, run the following script:

```
Exercise 4.3
import boto3
import json
import datetime

Just a quick helper function for date time conversions, in case you want to
print the raw JSON
```

---

*(continued)*

**EXERCISE 4.3 (continued)**

```
def date_time_converter(o):
 if isinstance(o, datetime.datetime):
 return o.__str__()

Variables
rds_identifier = 'my-rds-db'

Create the client for Amazon RDS
rds_client = boto3.client('rds')

print("Fetching the RDS endpoint...")
response = rds_client.describe_db_instances(
 DBInstanceIdentifier=rds_identifier
)

rds_endpoint = json.dumps(response['DBInstances'][0]['Endpoint']['Address'])
rds_endpoint = rds_endpoint.replace("'", '')
print('RDS Endpoint: ' + rds_endpoint)
```

After running the Python code, the following status is displayed:

```
Fetching the RDS endpoint.. RDS Endpoint:<endpoint_name>
```

If the endpoint is not returned, from the AWS Management Console, under the RDS service, verify that your Amazon RDS database instance was created.

---

**EXERCISE 4.4****Create a SQL Table and Add Records to It**

You now have all the necessary information to create your first SQL table by using Amazon RDS. In this exercise, you will create a SQL table and add a couple of records. Remember to update the variables for your specific environment.

To update the variables, run the following script:

```
Exercise 4.4
import boto3
import json
import datetime
import pymysql as mariadb

Variables
rds_identifier = 'my-rds-db'
```

---

---

```
db_name = 'mytestdb'
user_name = 'masteruser'
user_password = 'mymasterpassw0rd1!'
rds_endpoint = 'my-rds-db.****.us-east-1.rds.amazonaws.com'

Step 1 - Connect to the database to create the table
db_connection = mariadb.connect(host=rds_endpoint, user=user_name,
 password=user_password, database=db_name)
cursor = db_connection.cursor()
try:
 cursor.execute("CREATE TABLE Users (user_id INT NOT NULL AUTO_INCREMENT,
 user_fname VARCHAR(100) NOT NULL, user_lname VARCHAR(150) NOT NULL, user_
 email VARCHAR(175) NOT NULL, PRIMARY KEY (`user_id`))")
 print('Table Created!')
except mariadb.Error as e:
 print('Error: {}'.format(e))
finally:
 db_connection.close()

Step 2 - Connect to the database to add users to the table
db_connection = mariadb.connect(host=rds_endpoint, user=user_name,
 password=user_password, database=db_name)
cursor = db_connection.cursor()
try:
 sql = "INSERT INTO `Users` (`user_fname`, `user_lname`, `user_email`) VALUES
 (%s, %s, %s)"
 cursor.execute(sql, ('CJ', 'Smith', 'casey.smith@somewhere.com'))
 cursor.execute(sql, ('Casey', 'Smith', 'sam.smith@somewhere.com'))
 cursor.execute(sql, ('No', 'One', 'no.one@somewhere.com'))
 # No data is saved unless we commit the transaction!
 db_connection.commit()
 print('Inserted Data to Database!')
except mariadb.Error as e:
 print('Error: {}'.format(e))
 print('Sorry, something has gone wrong!')
finally:
 db_connection.close()
```

After running the Python code, the following confirmation is displayed:

Table Created! Inserted Data to the Database!

Your Amazon RDS database now has some data stored in it.

---

(continued)

**EXERCISE 4.4 (continued)**

In this exercise, you are hardcoding a password into your application code for demonstration purposes only. In a production environment, refrain from hard-coding application passwords. Instead, use services such as AWS Secrets Manager to keep your secrets secure.

**EXERCISE 4.5****Query the Items in the SQL Table**

After adding data to your SQL database, in this exercise you will be able to read or query the items in the *Users* table.

To read the items in the SQL table, run the following script:

```
Exercise 4.5
import boto3
import json
import datetime
import pymysql as mariadb

Variables
rds_identifier = 'my-rds-db'
db_name = 'mytestdb'
user_name = 'masteruser'
user_password = 'mymasterpassw0rd1!'
rds_endpoint = 'my-rds-db.*****.us-east-1.rds.amazonaws.com'

db_connection = mariadb.connect(host=rds_endpoint, user=user_name,
password=user_password, database=db_name)
cursor = db_connection.cursor()
try:
 sql = "SELECT * FROM `Users`"
 cursor.execute(sql)
 query_result = cursor.fetchall()
 print('Querying the Users Table...')
 print(query_result)
except mariadb.Error as e:
 print('Error: {}'.format(e))
 print('Sorry, something has gone wrong!')
```

---

```
finally:
 db_connection.close()
```

After running the Python code, you will see the three records that you inserted in the previous exercise.

---

**EXERCISE 4.6****Remove Amazon RDS Database and Security Group**

You've created an Amazon RDS DB instance and added data to it. In this exercise, you will remove a few resources from your account. Remove the Amazon RDS instance first.

To remove the Amazon RDS instance and the security group, run the following script:

```
Exercise 4.6
import boto3
import json
import datetime

Variables
rds_identifier = 'my-rds-db'
sg_name = 'rds-sg-dev-demo'
sg_id_number = ''

Create the client for Amazon RDS
rds_client = boto3.client('rds')

Delete the RDS Instance
response = rds_client.delete_db_instance(
 DBInstanceIdentifier=rds_identifier,
 SkipFinalSnapshot=True)

print('RDS Instance is being terminated...This may take several minutes.')

waiter = rds_client.get_waiter('db_instance_deleted')
waiter.wait(DBInstanceIdentifier=rds_identifier)

We must wait to remove the security groups until the RDS database has been
deleted, this is a dependency.
print('The Amazon RDS database has been deleted. Removing Security Groups')

Create the client for Amazon EC2 SG
```

---

*(continued)*

**EXERCISE 4.6 (*continued*)**

```
ec2_client = boto3.client('ec2')

Get the Security Group ID Number
response = ec2_client.describe_security_groups(
 GroupNames=[
 sg_name
])
sg_id_number = json.dumps(response['SecurityGroups'][0]['GroupId'])
sg_id_number = sg_id_number.replace('"','')

Delete the Security Group!
response = ec2_client.delete_security_group(
 GroupId=sg_id_number
)

print('Cleanup is complete!')
```

After running the Python code, the following message is displayed:

```
Cleanup is complete!
```

The Amazon RDS database and the security group are removed. You can verify this from the AWS Management Console.

---

**EXERCISE 4.7****Create an Amazon DynamoDB Table**

Amazon DynamoDB is a managed NoSQL database. One major difference between DynamoDB and Amazon RDS is that DynamoDB doesn't require a server that is running in your VPC, and you don't need to specify an instance type. Instead, create a table.

To create the table, run the following script:

```
Exercise 4.7
import boto3
import json
import datetime
```

---

---

```
dynamodb_resource = boto3.resource('dynamodb')

table = dynamodb_resource.create_table(
 TableName='Users',
 KeySchema=[
 {
 'AttributeName': 'user_id',
 'KeyType': 'HASH'
 },
 {
 'AttributeName': 'user_email',
 'KeyType': 'RANGE'
 }
],
 AttributeDefinitions=[
 {
 'AttributeName': 'user_id',
 'AttributeType': 'S'
 },
 {
 'AttributeName': 'user_email',
 'AttributeType': 'S'
 }
],
 ProvisionedThroughput={
 'ReadCapacityUnits': 5,
 'WriteCapacityUnits': 5
 }
)

print("The DynamoDB Table is being created, this may take a few minutes...")
table.meta.client.get_waiter('table_exists').wait(TableName='Users')
print("Table is ready!")
```

After running the Python code, the following message is displayed:

Table is ready!

From the AWS Management Console, under DynamoDB, verify that the table was created.

---

**EXERCISE 4.8****Add Users to the Amazon DynamoDB Table**

With DynamoDB, there are fewer components to set up than there are for Amazon RDS. In this exercise, you'll add users to your table. Experiment with updating and changing some of the code to add multiple items to the database.

To add users to the DynamoDB table, run the following script:

```
Exercise 4.8
import boto3
import json
import datetime
In this example we are not using uuid; however, you could use this to
autogenerated your user IDs.
i.e. str(uuid.uuid4())
import uuid

Create a DynamoDB Resource
dynamodb_resource = boto3.resource('dynamodb')
table = dynamodb_resource.Table('Users')

Write a record to DynamoDB
response = table.put_item(
 Item={
 'user_id': '1234-5678',
 'user_email': 'someone@somewhere.com',
 'user_fname': 'Sam',
 'user_lname': 'Samuels'
 }
)
Just printing the raw JSON response, you should see a 200 status code
print(json.dumps(response, indent=2, sort_keys=True))
```

After running the Python code, you receive a 200 HTTP Status Code from AWS. This means that the user record has been added.

From the AWS Management Console, under DynamoDB, review the table to verify that the user record was added.

---

**EXERCISE 4.9****Look Up a User in the Amazon DynamoDB Table**

In this exercise, you look up the one user you've added so far.

To look up users in the DynamoDB table, run the following script:

```
Exercise 4.9
import boto3
from boto3.dynamodb.conditions import Key
import json
import datetime

Create a DynamoDB Resource
dynamodb_resource = boto3.resource('dynamodb')
table = dynamodb_resource.Table('Users')

Query a some data
response = table.query(
 KeyConditionExpression=Key('user_id').eq('1234-5678')
)

Print the data out!
print(json.dumps(response['Items'], indent=2, sort_keys=True))
```

After running the Python code, the query results are returned in JSON format showing a single user.

---

**EXERCISE 4.10****Write Data to the Table as a Batch Process**

In this exercise, you will write data to the table through a batch process.

To write data using a batch process, run the following script:

```
Exercise 4.10
import boto3
import json
import datetime
import uuid

Create a DynamoDB Resource
```

*(continued)*

**EXERCISE 4.10 (*continued*)**

```
dynamodb_resource = boto3.resource('dynamodb')
table = dynamodb_resource.Table('Users')

Generate some random data
with table.batch_writer() as user_data:
 for i in range(100):
 user_data.put_item(
 Item={
 'user_id': str(uuid.uuid4()),
 'user_email': 'someone' + str(i) + '@somewhere.com',
 'user_fname': 'User' + str(i),
 'user_lname': 'UserLast' + str(i)
 }
)
 print('Writing record # ' + str(i+1) + ' to DynamoDB Users Table')
print('Done!')
```

After running the Python code, the last few lines read as follows:

```
Writing record # 300 to DyanmoDB Users Table Done!
```

From the AWS Management Console, under DynamoDB Table, verify that the users were written to the table.

---

**EXERCISE 4.11****Scan the Amazon DynamoDB Table**

In this exercise, you will scan the entire table.

To scan the table, run the following script:

```
Exercise 4.11
import boto3
import json
import datetime
import uuid

Create a DynamoDB Resource
dynamodb_resource = boto3.resource('dynamodb')
```

---

---

```
table = dynamodb_resource.Table('Users')

Let's do a scan!
response = table.scan()

print('The total Count is: ' + json.dumps(response['Count']))
print(json.dumps(response['Items'], indent=2, sort_keys=True))
```

As you learned in this chapter, scans return the entire dataset located in the table. After running the script, all of the users are returned.

---

## EXERCISE 4.12

### Remove the Amazon DynamoDB Table

In this exercise, you will remove the DynamoDB table that you created in Exercise 4.7.

To remove the table, run the following script:

```
Exercise 4.12
import boto3
import json
import datetime
import uuid

Create a DynamoDB Resource
dynamodb_client = boto3.client('dynamodb')

Delete the Table
response = dynamodb_client.delete_table(TableName='Users')
print(json.dumps(response, indent=2, sort_keys=True))
```

The DynamoDB table is deleted, or it is in the process of being deleted. Verify the deletion from the AWS Management Console, under the DynamoDB service.

---

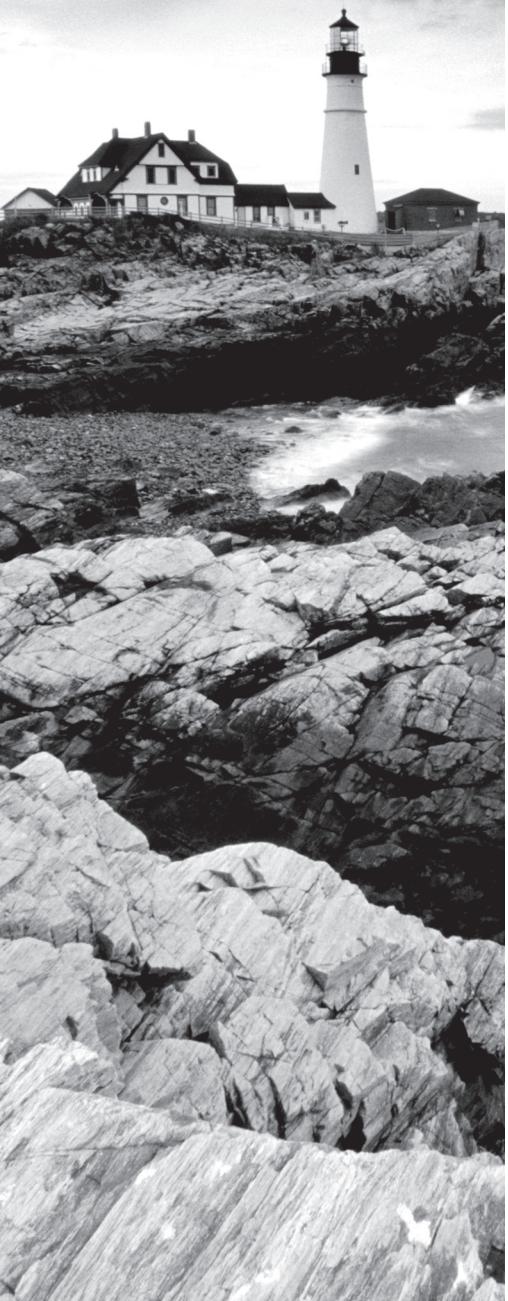
## Review Questions

1. Which of the following does Amazon Relational Database Service (Amazon RDS) manage on your behalf? (Select THREE.)
  - A. Database settings
  - B. Database software installation and patching
  - C. Query optimization
  - D. Hardware provisioning
  - E. Backups
2. Which AWS database service is best suited for managing highly connected datasets?
  - A. Amazon Aurora
  - B. Amazon Neptune
  - C. Amazon DynamoDB
  - D. Amazon Redshift
3. You are designing an ecommerce web application that will scale to potentially hundreds of thousands of concurrent users. Which database technology is best suited to hold the session state for large numbers of concurrent users?
  - A. Relational database by using Amazon Relational Database Service (Amazon RDS)
  - B. NoSQL database table by using Amazon DynamoDB
  - C. Data warehouse by using Amazon Redshift
  - D. MySQL on Amazon EC2
4. How many read capacity units (RCUs) do you need to support 25 *strongly consistent* reads per seconds of 15 KB?
  - A. 100 RCUs
  - B. 25 RCUs
  - C. 10 RCUs
  - D. 15 RCUs
5. How many read capacity units (RCUs) do you need to support 25 *eventually consistent* reads per seconds of 15 KB?
  - A. 10 RCUs
  - B. 25 RCUs
  - C. 50 RCUs
  - D. 15 RCUs

6. How many write capacity units (WCUs) are needed to support 100 writers per second of 512 bytes?
  - A. 129 WCUs
  - B. 25 WCUs
  - C. 10 WCUs
  - D. 100 WCUs
7. Your company is using Amazon DynamoDB, and they would like to implement a write-through caching mechanism. They would like to get everything up and running in only a few short weeks. Additionally, your company would like to refrain from managing any additional servers. You are the lead developer on the project; what should you recommend?
  - A. Build your own custom caching application.
  - B. Implement Amazon DynamoDB Accelerator (DAX).
  - C. Run Redis on Amazon EC2.
  - D. Run Memcached on Amazon EC2.
8. Your company would like to implement a highly available caching solution for its SQL database running on Amazon RDS. Currently, all of its services are running in the AWS Cloud. As their lead developer, what should you recommend?
  - A. Implement your own caching solution on-premises.
  - B. Implement Amazon ElastiCache for Redis.
  - C. Implement Amazon ElastiCache for Memcached.
  - D. Implement Amazon DynamoDB Accelerator (DAX).
9. A company is looking to run analytical queries and would like to implement a data warehouse. It estimates that it has roughly 300 TB worth of data, which is expected to double in the next three years. Which AWS service should you recommend?
  - A. Relational database by using Amazon Relational Database Service (Amazon RDS)
  - B. NoSQL database table by using Amazon DynamoDB
  - C. Data warehouse by using Amazon Redshift
  - D. Amazon ElastiCache for Redis
10. A company is experiencing an issue with Amazon DynamoDB whereby the data is taking longer than expected to return from a query. You are tasked with investigating the problem. After looking at the application code, you realize that a Scan operation is being called for a large DynamoDB table. What should you do or recommend?
  - A. Implement a query instead of a scan, if possible, as queries are more efficient than a scan.
  - B. Do nothing; the problem should go away on its own.
  - C. Implement a strongly consistent read.
  - D. Increase the write capacity units (WCUs).



# Chapter 5



AWS® Certified Developer Official Study Guide  
By Nick Alteen, Jennifer Fisher, Casey Gerena, Wes Gruver, Asim Jalil,  
Heiwad Osman, Marife Pagan, Santosh Patlolla and Michael Roth  
Copyright © 2019 by Amazon Web Services, Inc.

# Encryption on AWS

---

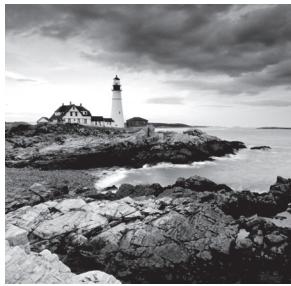
**THE AWS CERTIFIED DEVELOPER –  
ASSOCIATE EXAM TOPICS COVERED IN  
THIS CHAPTER MAY INCLUDE, BUT ARE  
NOT LIMITED TO, THE FOLLOWING:**

**Domain 2: Security**

- ✓ 2.2 Implement encryption using AWS services.

**Domain 3: Development with AWS Services**

- ✓ 3.4 Write code that interacts with AWS services by using APIs, SDKs, and AWS CLI.



## Introduction to Encryption

AWS delivers a secure, scalable cloud computing platform with high availability, offering the flexibility for you to build a wide range of applications. If you require an additional layer of security for the data you store in the AWS Cloud, there are several options for encrypting data at rest. These options range from automated AWS encryption solutions to manual, client-side options. Choosing the right solutions depends on which AWS service you're using and your requirements for key management. This chapter provides an overview of various methods for encrypting data at rest in AWS. Specifically, it covers three options and compares and contrasts the advantages of each option.

Before exploring the different ways that you can use encryption in AWS, the following section describes two services that you can use for your encryption strategy: AWS Key Management Service and AWS CloudHSM.

## AWS Key Management Service

*AWS Key Management Service* (AWS KMS) is a managed AWS service that makes it easy to create and manage encryption keys to encrypt your data across a wide range of AWS services and in your applications. As a secure, resilient service, AWS KMS uses FIPS 140-2 validated cryptographic modules, known as a hardware security module (HSM), to protect your master keys. The Federal Information Processing Standards (FIPS) are responsible for defining security requirements for cryptographic modules. For more information about FIPS 140-2 validation, see <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-2.pdf>.

You can take advantage of a number of AWS KMS features and benefits when developing your applications. You can use AWS KMS to make the applications and data more secure while still enabling you to innovate quickly through an API.

AWS KMS offers the following features:

- Centralized key management
- Integration with other AWS services
- Audit capabilities and high availability
- Custom key store
- Compliance

## Centralized Key Management

AWS KMS provides you with a centralized view of your encryption keys. You can create a *customer master key* (CMK) to control access to your data encryption keys (data keys) and to encrypt and decrypt your data. AWS KMS uses an Advanced Encryption Standard (AES) in 256-bit mode to encrypt and secure your data.

You can use AWS KMS to create keys in one of three ways: by using AWS KMS, by using AWS CloudHSM, or by importing your own key material. Regardless of the method you use to store your keys, you can manage them with AWS KMS through the AWS Management Console or by using the AWS SDK or the AWS CLI. AWS KMS also automatically rotates your keys once a year, without having to re-encrypt data that was previously encrypted.

## Integration with Other AWS Services

AWS KMS provides seamless integration with other AWS services. This integration means that, as a developer, you can quickly create keys to encrypt data that is stored in other AWS services, such as Amazon Simple Storage Service (Amazon S3).

AWS KMS provides an AWS managed master key for a variety of AWS services that integrate with AWS KMS. You can track the AWS managed CMKs in your account, but the service itself manages the keys. For greater control over the encryption process, you can generate your own CMK.

At the time of this writing, AWS KMS supports 51 AWS services, as shown in Figure 5.1.

**FIGURE 5.1** Supported AWS services

AWS Services Integrated with KMS			
Alexa for Business*	Amazon EMR	Amazon SageMaker	AWS CodeDeploy
Amazon Athena	Amazon FSx for Windows File Server	Amazon Simple Email Service (Amazon SES)	AWS CodePipeline
Amazon Aurora	Amazon Simple Storage Service Glacier	Amazon Simple Notification Service (Amazon SNS)	AWS Database Migration Service
Amazon CloudWatch logs	Amazon Kinesis Data Streams	Amazon Simple Queue Service (Amazon SQS)	AWS Glue
Amazon Comprehend*	Amazon Kinesis Data Firehose	Amazon Translate	AWS Lambda
Amazon Connect	Amazon Kinesis Video Streams	Amazon WorkMail	AWS Secrets Manager
Amazon DocumentDB	Amazon Lex	Amazon WorkSpaces	AWS Systems Manager
Amazon DynamoDB*	Amazon Lightsail*	AWS Backup	AWS Snowball
Amazon DynamoDB Accelerator (DAX)*	Amazon Managed Streaming for Kafka (MSK)	AWS Certificate Manager*	AWS Snowball Edge
Amazon Elastic Block Store (Amazon EBS)	Amazon Neptune	AWS Cloud9*	AWS Snowmobile
Amazon Elastic File System (Amazon EFS)	Amazon Redshift	AWS CloudTrail	AWS Storage Gateway
Amazon Elastic Transcoder	Amazon Relational Database Service (RDS)	AWS CodeBuild	AWS X-Ray
Amazon Elasticsearch Service	Amazon Simple Storage Service (Amazon S3)	AWS CodeCommit*	

## Auditing Capabilities and High Availability

If AWS CloudTrail is enabled for your AWS account and Region, API requests and other activity in your AWS account are recorded to log files. With CloudTrail, you can see who has used a particular AWS KMS CMK, the API call that was sent, and when they attempted to use that particular key.

In addition to auditing capabilities, AWS KMS is a fully managed service, which means that as your encryption needs grow or change, AWS KMS can scale automatically to meet those needs. Additionally, because this is a managed service, AWS KMS stores encrypted copies or versions of your keys inside systems that are designed for 99.999999999 percent durability.

The AWS CMKs do not leave the CloudHSM instances. Your keys are stored securely within the AWS Region so that no one, including AWS employees, can retrieve your plain-text keys from AWS KMS. AWS KMS uses FIPS 140-2 validated HSMs to protect your keys and to help ensure the confidentiality and integrity of your data.

## Custom Key Store

You can create your own custom key store in an CloudHSM cluster that you control, enabling you to store your AWS KMS keys in a single-tenant environment instead of the default multi-tenant environment of AWS KMS. The use of a custom key store incurs an additional cost for the CloudHSM cluster.

## Compliance

Achieving compliance for your applications can be a lengthy and difficult process. The security and quality controls in AWS KMS have been validated and certified by a number of industry-specific compliance and regulatory standards. For a full list of compliance standards that have been met, see <https://aws.amazon.com/compliance/services-in-scope/>.

# AWS CloudHSM

AWS *CloudHSM* offers third-party, validated FIPS 140-2, level-three hardware security modules in the AWS Cloud. The hardware security module is a computing device that provides a dedicated infrastructure to support cryptographic operations. You can use CloudHSM to support encryption for your application while running in your own Amazon Virtual Private Cloud (Amazon VPC). This means that your Amazon Elastic Compute Cloud (Amazon EC2) instances can access the CloudHSM device quickly while isolating them from other networks.

CloudHSM provides both asymmetric and symmetric encryption capabilities. Additionally, you can use the CloudHSM software libraries to integrate applications with HSMs in your cluster. The libraries include PKCS #11, Sun Java JCE (Java Cryptography Extension), and Cryptography API: Next Generation (CNG) providers for Microsoft. By using these libraries, you can perform cryptographic operations on the HSMs.

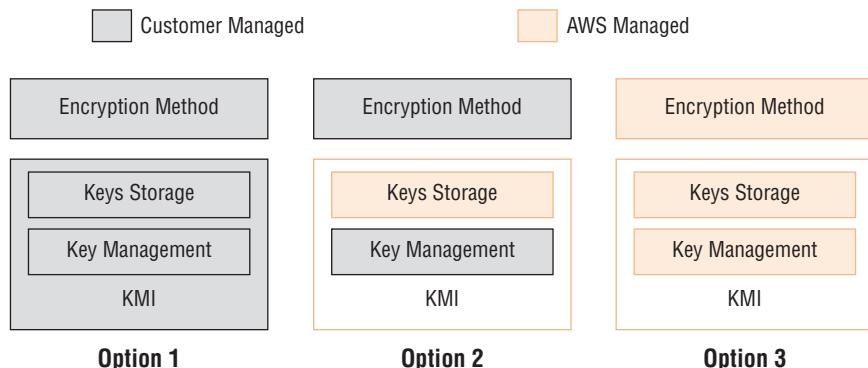
# Controlling the Access Keys

Encryption on any system requires three components: data to encrypt, a method to encrypt the data using a cryptographic algorithm, and the use of encryption keys with the data and the algorithm. Most modern programming languages provide libraries with a wide range of available cryptographic algorithms, such as the Advanced Encryption Standard (AES). Choosing the right algorithm involves evaluating security, performance, and compliance requirements specific to your application. Although the selection of an encryption algorithm is important, protecting the keys from unauthorized access is critical. Managing the security of encryption keys is often performed using a *key management infrastructure* (KMI). A KMI is composed of two subcomponents: the storage layer that protects the plaintext keys and the management layer that authorizes key use. A common way to protect keys in a KMI is to use a hardware security module. An HSM is a dedicated storage and data processing device that performs cryptographic operations using keys on the device. An HSM typically provides tamper evidence, or resistance, to protect keys from unauthorized use. A software-based authorization layer controls who can administer the HSM and which users or applications can use which keys in the HSM.

As you deploy encryption for various data classifications in AWS, it is important to understand exactly who has access to your encryption keys or data and under what conditions. As shown in Figure 5.2, there are three different options for how you and AWS provide the encryption method and the KMI:

- You control the encryption method and the entire KMI.
- You control the encryption method, AWS provides the storage component of the KMI, and you provide the management layer of the KMI.
- AWS controls the encryption method and the entire KMI.

**FIGURE 5.2** Encryption options in AWS



## Option 1: You Control the Encryption Method and the Entire KMI

In this option, you use your own KMI to generate, store, and manage access to keys in addition to controlling all the encryption methods in your applications. This physical location of the KMI and the encryption method can be outside of AWS or in an Amazon EC2 instance that you own. The encryption method can be a combination of open source tools, AWS SDKs, or third-party software and hardware. The important security property of this option is that you have full control over the encryption keys and the execution environment that uses those keys in the encryption code. AWS has no access to your keys and cannot perform encryption or decryption on your behalf. You are responsible for the proper storage, management, and use of keys to ensure the confidentiality, integrity, and availability of your data. You can encrypt data in AWS services, as described in the following sections.

### Amazon Simple Storage Service

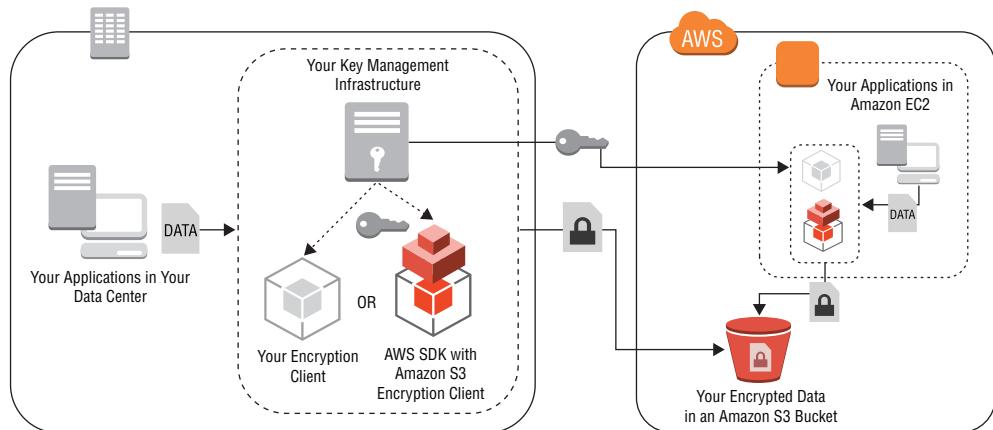
You can encrypt data by using any encryption method you want and then upload the encrypted data using the Amazon Simple Storage Service (Amazon S3) API. Most common application languages include cryptographic libraries that enable you to perform encryption in your applications. There are many commonly available open source tools for data encryption; however, they go beyond the scope of this study guide. After you have encrypted an object and safely stored the key in your KMI, you can upload the encrypted object to Amazon S3 directly with a PUT request. To decrypt this data, issue the GET request in the Amazon S3 API and then pass the encrypted data to your local application for decryption.

AWS provides an alternative to these open source encryption tools with the *Amazon S3 encryption client*, which is an open source set of APIs embedded in the AWS SDKs. This client lets you supply a key from your KMI that can be used to encrypt or decrypt your data as part of the call to Amazon S3. The SDK leverages Java Cryptography Extensions (JCEs) in your application to take your symmetric or asymmetric key as input and encrypt the object before uploading it to Amazon S3. The process is reversed when the SDK is used to retrieve an object. The downloaded encrypted object from Amazon S3 is passed to the client along with the key from your KMI. The underlying JCE in your application decrypts the object.

The Amazon S3 encryption client is integrated into the AWS SDKs for Java, Ruby, and .NET. It provides a transparent drop-in replacement for any cryptographic code that you might have used previously with your application that interacts with Amazon S3. Although AWS provides the encryption method, you control the security of your data because you control the keys for that engine to use. If you're using the Amazon S3 encryption client on-premises, AWS does not have access to your keys or unencrypted data. If you're using the client in an application running in Amazon EC2, a best practice is to pass keys to the client by using secure transport (for example, Secure Sockets Layer [SSL] or Secure Shell [SSH]) from your KMI to help ensure confidentiality. Figure 5.3 shows an example of Amazon S3

client-side encryption from an on-premises system compared with encryption within an Amazon EC2 application.

**FIGURE 5.3** Amazon S3 client-side encryption



## Amazon Elastic Block Store

*Amazon Elastic Block Store* (Amazon EBS) provides block-level storage volumes for use with Amazon EC2 instances. Amazon EBS volumes are network-attached and persist independently from the life of an instance.

**System-level or block-level encryption** Because Amazon EBS volumes are presented to an instance as a block device, you can leverage most standard encryption tools for file system-level or block-level encryption. Some common block-level open source encryption solutions for Linux are *Loop-AES*, *dm-crypt* (with or without LUKS extension), and *TrueCrypt*. Each of these operates below the file system layer using kernel space device drivers to perform the encryption and decryption of data. These tools are useful when you want all data written to a volume to be encrypted regardless of what directory the data is stored in.

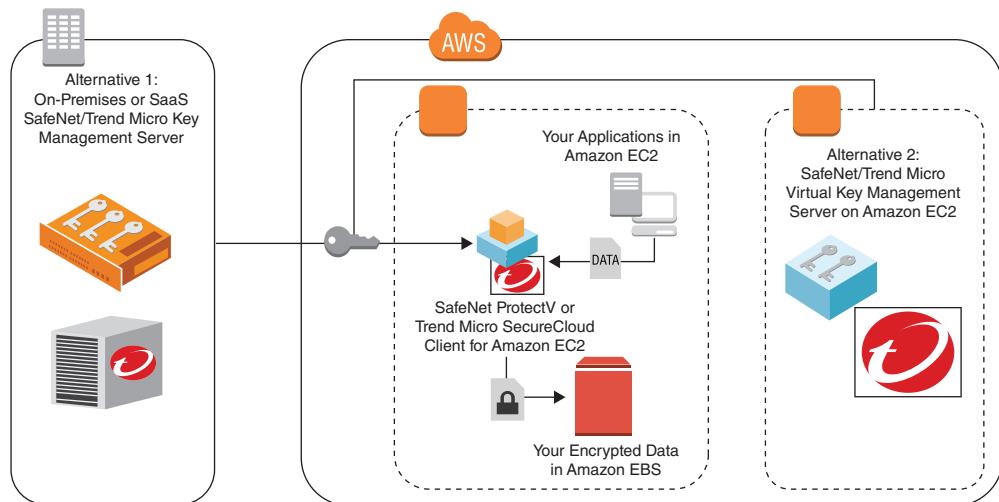
**File-system encryption** You can use file system-level encryption, which works by stacking an encrypted file system on top of an existing file system. This method is typically used to encrypt a specific directory. *eCryptfs* and *EncFs* are two Linux-based open source examples of file system-level encryption tools.

These solutions require you to provide keys either manually or from your KMI. An important caveat with both block-level and file system-level encryption tools is that you can use them only to encrypt data volumes that are not Amazon EBS boot volumes. This is because these tools do not allow you to make a trusted key available automatically to the boot volume at startup.

AWS partner solutions help automate the process of encrypting Amazon EBS volumes in addition to supplying and protecting the necessary keys. Trend Micro SecureCloud and

SafeNet ProtectV are two such partner products that encrypt Amazon EBS volumes and include a KMI. Figure 5.4 shows how you can use the SafeNet and Trend Micro solutions to encrypt data stored on Amazon EBS using keys managed on-premises, via SaaS, or in applications running on Amazon EC2

**FIGURE 5.4** Encryption in Amazon EBS using SafeNet ProtectV or Trend Micro SecureCloud



## AWS Storage Gateway

AWS Storage Gateway is a service connecting an on-premises software appliance with Amazon S3. You can expose it to your network as an iSCSI disk to facilitate copying data from other sources. Data on disk volumes attached to the Storage Gateway are automatically uploaded to Amazon S3 based on policy. You can encrypt source data on the disk volumes by using any of the file encryption methods described previously, such as Bouncy Castle or OpenSSL, before it is written to the disk. To encrypt all the data on the disk volume, you can also use a block-level encryption tool, such as BitLocker or dm-crypt/LUKS, on the iSCSI endpoint exposed by Storage Gateway.

## Amazon Relational Database Service

To encrypt data in *Amazon Relational Database Service* (Amazon RDS) using client-side technology, you must consider how you want data queries to work. Because Amazon RDS does not expose the attached disk it uses for data storage, transparent disk encryption using techniques described in the previous Amazon EBS section is not available. However, you can encrypt database fields in your application selectively by using any of the standard encryption libraries mentioned previously, such as Bouncy Castle and OpenSSL, before the data passes to your Amazon RDS instance.

Although this specific field data does not easily support range queries in the database, queries based on unencrypted fields can still return useful results. The encrypted fields of the returned results can be decrypted by your local application for presentation. To support more efficient querying of encrypted data, you can store a keyed-hash message authentication code (HMAC) of an encrypted field in your schema, and you can supply a key for the hash function. Subsequent queries of protected fields that contain the HMAC of the data being sought would not disclose the plaintext values in the query. This allows the database to perform a query against the encrypted data in your database without disclosing the plaintext values in the query. Any of the encryption methods you choose must be performed on your own application instance before data is sent to the Amazon RDS instance.

## Amazon EMR

*Amazon EMR* provides an easy-to-use Hadoop implementation running on Amazon EC2. Performing encryption throughout the Hadoop operation involves encryption and key management at the following distinct phases:

- Source data
- Hadoop Distributed File System (HDFS)
- Shuffle phase
- Output data

If the source data is not encrypted, then this step can be skipped, and SSL can be used to help protect data in transit to the Amazon EMR cluster. If the source data is encrypted, then your Hadoop job must decrypt the data as it is ingested. If your job flow uses Java and the source data is in Amazon S3, you can use any of the client decryption methods described in the previous Amazon S3 sections.

The storage used for the HDFS mount point is the ephemeral storage of the cluster nodes. Depending on the instance type, there might be more than one mount. To encrypt these mount points, you must use an Amazon EMR bootstrap script that will do the following:

1. Stop the Hadoop service.
2. Install a file-system-encryption tool on the instance.
3. Create an encrypted directory to mount the encrypted file system on top of the existing mount points.
4. Restart the Hadoop service.

For example, you can perform these steps on each of the HDFS mounts by using the open source *eCryptfs* package and an ephemeral key generated in your code. You don't need to worry about persistent storage of this encryption key because the data it encrypts does not persist beyond the life of the HDFS instance.

The shuffle phase involves passing data between cluster nodes before the reduce step. To encrypt this data in transit, when you create your cluster, you can enable SSL with a configure Hadoop bootstrap option.

Finally, to enable encryption of the output data, your Hadoop job should encrypt the output using a key sourced from your KMI. This data can be sent to Amazon S3 for storage in encrypted form.

## **Option 2: You Control the Encryption Method, AWS Provides the KMI Storage Component, and You Provide the KMI Management Layer**

This option is similar to option 1 in that you manage the encryption method, but it differs from option 1 in that the keys are stored in an AWS CloudHSM appliance rather than in a key storage system that you manage on-premises. While the keys are stored in the AWS environment, they are inaccessible to any employee at AWS because only you have access to the cryptographic partitions within the dedicated HSM to use the keys.

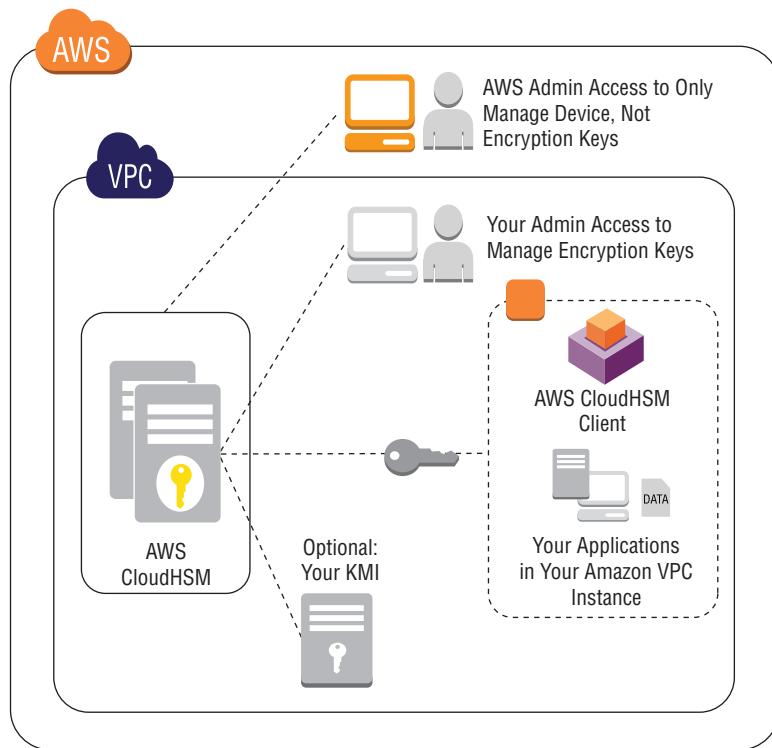
The CloudHSM appliance is a FIPS 140-2, level 3 HSM that has both physical and logical tamper detection and response mechanisms that trigger *zeroization* of the appliance. Zeroization erases the HSM’s volatile memory where any decrypted keys were stored. Zeroization destroys the key that encrypts stored objects, effectively causing all keys on the HSM to be inaccessible and unrecoverable.

### **CloudHSM**

To help you decide whether CloudHSM is appropriate for your deployment, it is important to understand the role that an HSM plays in encrypting data. You can use an HSM to generate and store key material and perform encryption and decryption operations. However, an HSM does not perform any key lifecycle management functions (such as access control policy, key rotation). This means you might need a compatible KMI, in addition to the CloudHSM appliance, before deploying your application. You can deploy the KMI either on-premises or within Amazon EC2. To help protect data and encryption keys, the KMI can communicate to the CloudHSM instance securely over SSL.

### **Amazon Virtual Private Cloud**

Applications must be able to access your CloudHSM appliance in an Amazon Virtual Private Cloud (Amazon VPC). The CloudHSM client interacts with the CloudHSM appliance to encrypt data from your application. You can then send encrypted data to any AWS service for storage. CloudHSM and your custom application support database, disk volume, and file encryption applications. Figure 5.5 shows how the CloudHSM solution works with your applications running on Amazon EC2 in an Amazon VPC.

**FIGURE 5.5** Deploying AWS CloudHSM in an Amazon VPC

To achieve the highest availability and durability of keys in your CloudHSM appliance, AWS recommends deploying multiple CloudHSM applications across different Availability Zones or with an on-premises HSM appliance that you manage.

## Option 3: AWS Controls the Encryption Method and the Entire KMI

AWS provides server-side encryption of your data, transparently managing the encryption method and keys.

### AWS Key Management Service

*AWS Key Management Service (AWS KMS)* is a managed encryption service that lets you provision and use keys to encrypt your data in AWS services and your applications. Master keys in AWS KMS are used in a similar way to how master keys in an HSM are used. Master keys are designed never to be exported from the service. You can send data to the service to be encrypted or decrypted using a specific master key under your account. This

design gives you centralized control over who can access your master keys to encrypt and decrypt data, and it gives you the ability to audit this access.

AWS KMS is natively integrated with other AWS services, including Amazon EBS, Amazon S3, and Amazon Redshift, to simplify encryption of your data within those services. AWS SDKs are integrated with AWS KMS to enable you to encrypt data in your custom applications. For applications that must encrypt data, AWS KMS provides global availability, low latency, and a high level of durability for your keys.

AWS KMS and other services that encrypt your data directly use a method called *envelope encryption* to balance performance and security. Figure 5.6 describes the flow of envelope encryption.

**FIGURE 5.6** Flow of envelope encryption

1. A data key is generated by the AWS service at the time you request your data to be encrypted.



2. Data key is used to encrypt your data.



3. The data key is then encrypted with a key-encrypting key unique to the service storing your data.



4. The encrypted data key and the encrypted data are then stored by the AWS storage service on your behalf.



The key-encrypting keys that are used to encrypt data keys are stored and managed separately from the data and the data keys. Strict access controls are placed on the encryption keys designed to prevent unauthorized use by AWS employees. When you need access to your plaintext data, this process is reversed. The encrypted data key is decrypted using the key-encrypting key; the data key is used to decrypt your data.

The following AWS services offer a variety of encryption features from which you can choose.

## Amazon S3

There are three ways to encrypt your data in Amazon S3 using server-side encryption.

**Server-side encryption** You can set an API flag or use the AWS Management Console to encrypt data before it is written to disk in Amazon S3. Each object is encrypted with a unique data key. As an additional safeguard, this key is encrypted with a periodically rotated master key managed by Amazon S3. Amazon S3 server-side encryption uses 256-bit Advanced Encryption Standard (AES) keys for both object and master keys. This feature is offered at no additional cost beyond what you pay for using Amazon S3.

**Server-side encryption using customer-provided keys** You can use your own encryption key while uploading an object to Amazon S3. Amazon S3 uses this encryption key to encrypt your data using AES-256. After the object is encrypted, the encryption key is deleted from the Amazon S3 system that used it to protect your data. When you retrieve this object from Amazon S3, you must provide the same encryption key in your request. Amazon S3 verifies that the encryption key matches, decrypts the object, and returns the object to you. This feature is offered at no additional cost beyond what you pay for using Amazon S3.

**Server-side encryption using AWS KMS** You can encrypt your data in Amazon S3 by defining an AWS KMS master key within your account. This master key is used to encrypt the unique object key (referred to as a data key, as shown in Figure 5.6) that ultimately encrypts your object.

When you upload your object, a request is sent to AWS KMS to create an object key. AWS KMS generates this object key and encrypts it using the master key that you specified earlier; AWS KMS returns this encrypted object key along with the plaintext object key to Amazon S3. The Amazon S3 web server encrypts your object using the plaintext object key, stores the now encrypted object (with the encrypted object key), and deletes the plaintext object key from memory.

To retrieve this encrypted object, Amazon S3 sends the encrypted object key to AWS KMS. AWS KMS decrypts the object key using the correct master key and returns the decrypted (plaintext) object key to Amazon S3. With the plaintext object key, Amazon S3 decrypts the encrypted object and returns it to you.

Amazon S3 also enables you to define a default encryption policy. You can specify that all objects are encrypted when stored. You can also define a bucket policy that rejects uploads of unencrypted objects.

## Amazon EBS

When creating a volume in Amazon EBS, you can choose to encrypt it using an AWS KMS master key within your account that encrypts the unique volume key that will ultimately encrypt your EBS volume. After you make your selection, the Amazon EC2 server sends an authenticated request to AWS KMS to create a volume key. AWS KMS generates this volume key, encrypts it using the master key, and returns the plaintext volume key and

the encrypted volume key to the Amazon EC2 server. The plaintext volume key is stored in memory to encrypt and decrypt all data going to and from your attached EBS volume. When the encrypted volume (or any encrypted snapshots derived from that volume) needs to be re-attached to an instance, a call is made to AWS KMS to decrypt the encrypted volume key. AWS KMS decrypts this encrypted volume key with the correct master key and returns the decrypted volume key to Amazon EC2.

## Amazon EMR

*S3DistCp* is an Amazon EMR feature that moves large amounts of data from Amazon S3 into HDFS, from HDFS to Amazon S3, and between Amazon S3 buckets. With *S3DistCp*, you can request Amazon S3 to use server-side encryption when it writes Amazon EMR data to an Amazon S3 bucket. This feature is offered at no additional cost beyond what you pay for using Amazon S3 to store your Amazon EMR data.

## Amazon Redshift

When creating an Amazon Redshift cluster, you can choose to encrypt all data in user-created tables. For server-side encryption of an Amazon Redshift cluster, you can choose from the following options:

**256-bit AES keys** Data blocks (included backups) are encrypted using random 256-bit AES keys. These keys are themselves encrypted using a random 256-bit AES database key, which is encrypted by a 256-bit AES cluster master key that is unique to your cluster. The cluster master key is encrypted with a periodically rotated regional master key unique to the Amazon Redshift service that is stored in separate systems under AWS control. This feature is offered at no additional cost beyond what you pay for using Amazon Redshift.

**CloudHSM cluster master key** The 256-bit AES cluster master key used to encrypt your database keys is generated in your CloudHSM or by using HSM appliance on-premises. This cluster master key is then encrypted by a master key that never leaves your HSM.

When the Amazon Redshift cluster starts, the cluster master key is decrypted in your HSM and used to decrypt the database key. The database key is sent to the Amazon Redshift hosts and resides only in memory for the life of the cluster. If the cluster ever restarts, the cluster master key is again retrieved from your HSM—it is not stored on disk in plaintext. This option lets you more tightly control the hierarchy and lifecycle of the keys used to encrypt your data. This feature is offered at no additional cost beyond what you pay for using Amazon Redshift (and CloudHSM, if you choose this option for storing keys).

**AWS KMS cluster master key** The 256-bit AES cluster master key used to encrypt your database keys is generated in AWS KMS. This cluster master key is then encrypted by a master key within AWS KMS. When the Amazon Redshift cluster starts up, the cluster master key is decrypted in AWS KMS and used to decrypt the database key, which is sent to the Amazon Redshift hosts to reside only in memory for the life of the cluster. If the cluster ever restarts, the cluster master key is again retrieved from the hardened security appliance

in AWS KMS—it is not stored on disk in plaintext. This option lets you define fine-grained controls over the access and use of your master keys and audit these controls through AWS CloudTrail. In addition to encrypting data generated within your Amazon Redshift cluster, you can also load encrypted data into Amazon Redshift from Amazon S3 that was previously encrypted using the Amazon S3 Encryption Client and keys that you provide. Amazon Redshift supports the decryption and re-encryption of data going between Amazon S3 and Amazon Redshift to protect the full lifecycle of your data.

These server-side encryption features across multiple services in AWS enable you to encrypt your data easily by setting the configuration in the AWS Management Console by making a CLI or API request for the given AWS service. AWS automatically and securely manages the authorized use of encryption keys. Because unauthorized access to those keys could lead to the disclosure of your data, AWS has built systems and processes with strong access controls that minimize the chance of unauthorized access. AWS had these systems verified by third-party audits to achieve security certifications, including SOC 1, 2, and 3; PCI-DSS; and FedRAMP.

## Summary

If you take all responsibility for the encryption method and the KMI, you can have granular control over how your applications encrypt data. However, that granular control comes at a cost—both in terms of deployment effort and an inability to have AWS services tightly integrate with your applications' encryption methods. As an alternative, you can choose a managed service that enables easier deployment and tighter integration with AWS Cloud services. This option offers checkbox encryption for several services that store your data, control over your own keys, secured storage for your keys, and auditability on all data access attempts.

## Exam Essentials

**Know how to define key management infrastructure (KMI).** A KMI consists of two infrastructure components. The first component is a storage layer that protects plaintext keys. The second component is a management layer that authorizes use of stored keys.

**Understand the available options for how you and AWS provide encryption using a KMI.** With the first option, you control the encryption method in addition to the entire KMI. In the second option, you control the encryption method and the management layer of the KMI, and AWS provides the storage layer. In the third option, AWS controls the encryption method and both components of the KMI.

**Understand the maintenance trade-offs of each key management option.** For any options that involve customers managing the components of the KMI or encryption method,

maintenance increases significantly. The increased maintenance also reduces your ability to take advantage of built-in integrations between AWS KMS and other services. For options that involve using built-in AWS functionality, additional maintenance is required only when migrating legacy applications to take advantage of new features.

**Understand the encryption options available in Amazon S3.** Regardless of the key management tools and process, you are able to encrypt any objects before uploading them to an Amazon S3 bucket. However, any custom encryption logic adds to processing overhead for the encryption and decryption of the data. AWS provides the Amazon S3 encryption client to help streamline this process (available in the Java, Ruby, and .NET AWS SDKs). When encrypting data on-premises, AWS has no visibility into the encryption keys or mechanisms used. For server-side encryption, Amazon S3 supports AWS-managed keys, customer-managed keys, and encryption using AWS KMS.

**Understand the encryption options available in Amazon EBS.** Like any on-premises block storage, Amazon EBS supports both block-level and file-system encryption. However, an important caveat with block-level and file-system encryption tools, such as TrueCrypt and eCryptfs, is that you cannot use them to encrypt the boot volume of an Amazon EC2 instance. Amazon EBS supports encryption by using customer-managed keys and AWS KMS.

**Understand the encryption options available in Amazon RDS.** Because Amazon RDS does not expose the underlying file system of databases, block-level and file-system encryption options are not available. However, standard libraries for encryption of database fields are fully supported. It is important to evaluate the types of queries that will be run against a database before selecting an encryption process, as this could affect the ability to run queries on encrypted data.

## Resources to Review

AWS Key Management Service FAQs:

<https://aws.amazon.com/kms/faqs/>

AWS CloudHSM FAQs:

<https://aws.amazon.com/cloudhsm/faqs/>

Amazon S3: Protecting Data Using Encryption:

<https://docs.aws.amazon.com/AmazonS3/latest/dev/UsingEncryption.html>

Amazon S3 Default Encryption for S3 Buckets:

<https://docs.aws.amazon.com/AmazonS3/latest/dev/bucket-encryption.html>

AWS Security Blog:

<https://aws.amazon.com/blogs/security/tag/encryption/>

Encrypting Amazon RDS Resources:

<https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Overview.Encryption.html>

Amazon EBS Encryption:

<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EBSEncryption.html>

Amazon Redshift Database Encryption:

<https://docs.aws.amazon.com/redshift/latest/mgmt/working-with-db-encryption.html>

# Exercises

## EXERCISE 5.1

### Configure an Amazon S3 Bucket to Deny Unencrypted Uploads

In this exercise, you will enforce object encryption for an Amazon S3 bucket by using a bucket policy to reject PUT requests without encryption headers.

1. Sign in to the AWS Management Console, and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. Create a new bucket with a name of your choice.
3. Apply the following policy to the bucket:

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "DenyIncorrectEncryption",
 "Effect": "Deny",
 "Principal": "*",
 "Action": "s3:PutObject",
 "Resource": "arn:aws:s3:::<bucket_name>/*",
 "Condition": {
 "StringNotEquals": {
 "s3:x-amz-server-side-encryption": "AES256"
 }
 },
 {
 "Sid": "DenyMissingEncryption",
 "Effect": "Deny",
 }
]
}
```

*(continued)*

**EXERCISE 5.1 (*continued*)**

```
"Principal": "*",
"Action": "s3:PutObject",
"Resource": "arn:aws:s3:::<bucket_name>/*",
"Condition": {
 "Null": {
 "s3:x-amz-server-side-encryption": true
 }
}
]
```

4. From the AWS Identity and Access Management (IAM) console, open the policy simulator.
5. Select an existing policy with access to the bucket that you created.
6. Test the PutObject Amazon S3 action with and without the x-amz-server-side-encryption header.

When you are uploading a new object or making a copy of an existing object, you can specify whether you want Amazon S3 to encrypt your data by adding the header to the API request.

---

**EXERCISE 5.2****Create and Disable an AWS Key Management Service (AWS KMS) Key**

In this exercise, you will create a customer master key (CMK) in the AWS Management Console and then disable it. You can disable and re-enable the AWS KMS CMKs that you manage.

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at <https://console.aws.amazon.com/kms>.
  2. Choose **Create key**.
  3. Provide values for the key alias, description, and tag(s), and then choose **Next**.  
The alias name cannot begin with aws. The aws prefix is reserved by AWS to represent AWS managed CMKs in your account.
  4. Select one or more IAM users who can administer the CMK and then choose **Next**. Make sure to select your IAM user.
  5. Select one or more IAM users to use the CMK for cryptographic operations. Make sure to select your IAM user.
-

- 
6. Choose **Finish** to create the CMK.
  7. Locate the key in the AWS KMS console.
  8. Select the check box next to the alias of the CMK that you want to disable.
  9. Choose **Key actions > Disable**.

If you disable a CMK, you cannot use it to encrypt or decrypt data until you re-enable it.

---

### EXERCISE 5.3

#### Create an AWS KMS Customer Master Key with the Python SDK

In this exercise, you will create a new AWS KMS customer master key (CMK) using the AWS Command Line Interface (AWS CLI). You will use Python as one of the supported programming languages.

1. To create the AWS KMS CMK, run the following Python script:

```
import boto3
import json

kms_client = boto3.client('kms', region_name='us-west-1')

response = kms_client.create_key(
 Description='My KMS Key',
 KeyUsage='ENCRYPT_DECRYPT',
 Origin='AWS_KMS',
 Tags=[
 {
 'TagKey': 'KeyPurpose',
 'TagValue': 'dev-on-aws-key'
 },
],
)
print(response)
```

2. To list the CMKs and describe the available keys, run the following script:

```
import boto3
import json
List the KMS Keys by ID
kms_client = boto3.client('kms', region_name='us-west-1')
try:
 response = kms_client.list_keys()
```

---

*(continued)*

**EXERCISE 5.3 (*continued*)**

```
except ClientError as e:
 logging.error(e)

print(json.dumps(response, indent=4, sort_keys=True))
```

3. To describe the keys inside AWS KMS, run the following script:

```
import boto3
import json

List the KMS Keys by ID
kms_client = boto3.client('kms', region_name='us-west-1')

Describe the Keys
for key in response['Keys']:
 try:
 key_info = kms_client.describe_key(KeyId=key['KeyArn'])
 key_id = key_info['KeyMetadata']['KeyId']
 key_arn = key_info['KeyMetadata']['Arn']
 key_state = key_info['KeyMetadata']['KeyState']
 key_description = key_info['KeyMetadata']['Description']
 print('Key ID: ' + key_id)
 print('Key ARN: ' + key_arn)
 print('Key State: ' + key_state)
 print('Key Description: ' + key_description)
 print('-----')
 except ClientError as e:
 logging.error(e)
```

4. To delete the AWS KMS key, run the following script:

```
import boto3

kms_client = boto3.client('kms', region_name='us-west-1')
response = kms_client.schedule_key_deletion(
 KeyId='fasdf1-2451b-151-bea2-easdfg8',
 PendingWindowInDays=7
)

print(response, indent=4, sort_keys=True)
```

---

# Review Questions

1. Which components are required in an encryption system? (Select THREE.)
  - A. A user to upload data
  - B. Data to encrypt
  - C. A database to store encryption keys
  - D. A method to encrypt data
  - E. A cryptographic algorithm
2. Which are the components of key management infrastructure (KMI)? (Select TWO.)
  - A. Storage layer
  - B. Data layer
  - C. Management layer
  - D. Encryption layer
3. Which of the following are methods for you and AWS to provide an encryption method and key management infrastructure (KMI)? (Select THREE.)
  - A. You control the encryption method and key management, and AWS provides the storage component of the KMI.
  - B. You control the storage component of the KMI, and AWS provides the encryption method and key management.
  - C. You control the encryption method and KMI.
  - D. AWS controls the encryption method and the entire KMI.
  - E. None of the above.
4. Which option uses AWS Key Management Service (AWS KMS) to manage keys to provide server-side encryption to Amazon Simple Storage Service (Amazon S3)?
  - A. Amazon S3 managed encryption keys (SSE-S3)
  - B. Customer-provided encryption keys (SSE-C)
  - C. Use client-side encryption
  - D. None of the above
5. Which AWS encryption service provides asymmetric encryption capabilities?
  - A. AWS Key Management Service (AWS KMS).
  - B. AWS CloudHSM.
  - C. AWS does not provide asymmetric encryption services.
  - D. None of the above.

6. Which AWS encryption service provides symmetric encryption capabilities? (Select TWO.)
  - A. AWS Key Management Service (AWS KMS).
  - B. AWS CloudHSM.
  - C. AWS does not provide symmetric encryption services.
  - D. None of the above.
7. An organization is using Amazon Simple Storage Service (Amazon S3), and it would like to ensure that all objects that are stored in Amazon S3 are encrypted. However, it does not want to be responsible for managing any of the encryption keys. As their lead developer, which service and feature should you recommend?
  - A. Server-side encryption with AWS Key Management Service (SSE-KMS).
  - B. Customer-provided encryption keys (SSE-C).
  - C. Amazon S3 managed encryption keys (SSE-S3).
  - D. This is not possible in AWS.
8. Which feature of AWS Key Management Service (AWS KMS) enables you to use an AWS CloudHSM cluster for the storage of your encryption keys?
  - A. Centralized key management
  - B. AWS CloudHSM
  - C. Custom key stores
  - D. S3DistCp
9. An organization is using AWS Key Management Service (AWS KMS) to support encryption and would like to encrypt Amazon Elastic Block Store (Amazon EBS) volumes. It wants to encrypt its volumes quickly, with little development time. As their lead developer, what should you recommend?
  - A. Implement AWS KMS to encrypt the Amazon EBS volumes.
  - B. Use open source or third-party encryption tooling.
  - C. Use AWS CloudHSM.
  - D. AWS does not provide a mechanism to encrypt Amazon EBS volumes.
10. Which of the following AWS services does not integrate with AWS Key Management Service (AWS KMS)?
  - A. Amazon Elastic Block Store (Amazon EBS)
  - B. Amazon Simple Storage Service (Amazon S3)
  - C. Amazon Redshift
  - D. None of the above

# Chapter 6



# Deployment Strategies

---

AWS® Certified Developer Official Study Guide  
By Nick Alteen, Jennifer Fisher, Casey Gerena, Wes Gruver, Asim Jalil,  
Heiward Osman, Marife Pagan, Santosh Patlolla and Michael Roth  
Copyright © 2019 by Amazon Web Services, Inc.

**THE AWS CERTIFIED DEVELOPER –  
ASSOCIATE EXAM TOPICS COVERED IN  
THIS CHAPTER MAY INCLUDE, BUT ARE  
NOT LIMITED TO, THE FOLLOWING:**

## Domain 1: Deployment

- ✓ 1.1 Deploy written code in AWS using existing CI/CD pipelines, processes, and patterns.
- ✓ 1.2 Deploy applications using AWS Elastic Beanstalk.

Content may include the following:

- Environments and architectures
- Environment variables
- Software Development Lifecycle (SDLC)
- AWS services for automating deployments
- AWS Cloud tiers: Web servers, worker applications, and databases
- Deployment strategies

## Domain 2: Security

- ✓ 2.3 Implement application authentication and authorization.

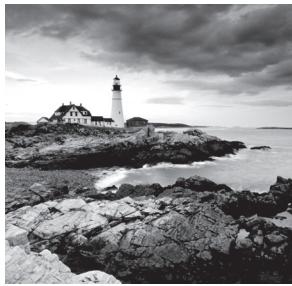
Content may include the following:

- AWS Identity and Access Management (IAM) Roles in AWS Elastic Beanstalk

## Domain 5: Monitoring and Troubleshooting

Content may include the following:

- Monitoring AWS Elastic Beanstalk
- Troubleshooting AWS Elastic Beanstalk



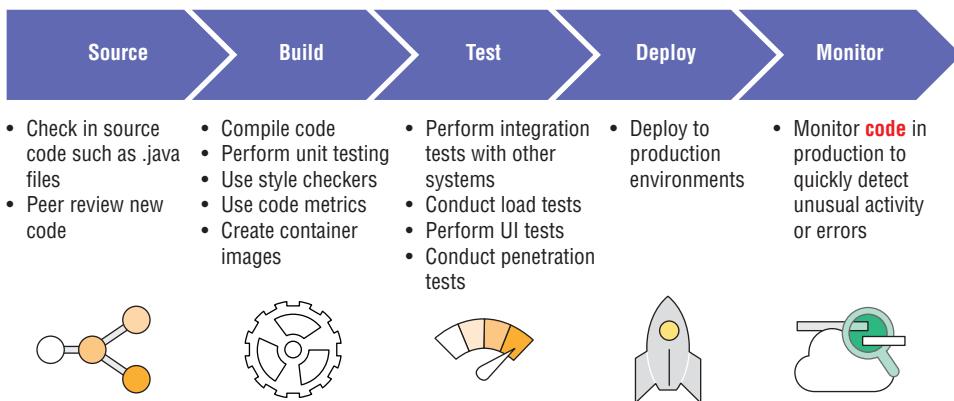
## Deployments on the AWS Cloud

As a developer in the cloud, you will often create *three-tier architectures* that consist of a web tier, an application tier, and a database tier. To enable your customers to use your application immediately, you must rapidly deploy both your infrastructure and code. The AWS Cloud environment offers several deployment options and several ways to provision AWS services to set up highly available and reliable applications. Ideally, deployments are seamless streams of automated processes that create, build, deploy, monitor, and modify code throughout the entire *software development lifecycle* (SDLC). This stream of processes must be continuous and fully integrated with your AWS services.

In a traditional environment, deployments can require substantial time to push the code to multiple environments. AWS helps to speed up this process by automating the actions required to deploy code to your environments. When you need to upload and demonstrate a code project in the cloud, you can launch an application in minutes.

### Phases of the Release Lifecycle

Each team's release lifecycle is different based on the needs of the team. Nearly all traditional release lifecycles are composed of five major phases, as shown in Figure 6.1: Source, Build, Test, Production, and Monitor. Each phase of the cycle provides increased confidence that the code will work in the intended way for customers. This also translates to a release lifecycle implemented in the AWS Cloud environment.

**FIGURE 6.1** Major phases of the release lifecycle

## Source Phase

During the *Source phase*, developers check changes into a source code repository. Many teams require peer feedback on code changes before delivering code to production or target environments. Teams may use several methods for code reviews, such as pair programming and tool-assisted options.

## Build Phase

During the *Build phase*, an application's source code is built, and the quality of the code is tested on the build machine. The most common types of quality checks are automated tests that do not require a server to execute and can be initiated from a test harness. Some teams extend their quality tests to include code metrics and style checks. There is an opportunity for automation any time a human must decide on the code.

## Test Phase

The goal of the *Test phase* is to perform tests that cannot be done during the Build phase and that require the software to be deployed to production-like stages. Often, these tests include testing integration with other live systems, load testing, user interface (UI) testing, and penetration testing. AWS has many different stages to which it deploys. Teams deploy to preproduction stages where their application interacts with other systems to ensure that the newly changed software works in an integrated environment.

## Deployment Phase

In the Deployment phase, code is deployed to production. Different teams have different deployment strategies, though it is common to set goals to reduce risk when deploying new changes and minimize the impact when a bad change is rolled into production.

## Monitor Phase

During the Monitor phase, you must check the application to detect unusual activities and errors quickly.

You can automate each of these phases without automating the entire release lifecycle.

## Environment Variables

Before you deploy your infrastructure and code, first determine the environmental variables. The SDLC in a traditional infrastructure contains manual implementations to release, test, and deploy code, in addition to the corresponding required documentation. Most SDLC models would benefit from an efficient lifecycle with accurate execution and no administration.

In traditional deployments, operating system patching, updates, language versioning, and infrastructure changes frequently do not occur in synchronization and may not always match between environments. It is often difficult to replicate configurations between environments in a traditional infrastructure. Unit tests, user acceptance tests, and load tests can produce different results in test and production environments. These results differ because of differences in environment variables, such as unapplied patches or nonupdated configurations. An ideal test environment matches the production environment exactly, providing the capability to test an application as if it were running in production and to receive accurate results.

Additionally, the SDLC requires the maintenance of phases that you customize to meet business requirements. This includes configurations to audit and perform quality assurance. AWS Cloud solutions automatically align the phases and environments. You can choose an AWS service to automate deployments seamlessly, saving you hours that you would normally spend managing your infrastructure and code. Although certain environmental variables are clearly defined as business requirements, others evolve as you commit changes. Code modifications are also inherently environment-based and focused on the configuration of the modification itself.

AWS services enable you to deploy applications rapidly and manage environments and the multiple tiers of your infrastructure (web servers, worker applications, and databases) automatically. You can automatically provision and manage resources for the environments and configurations that you create. Your configurations can include Auto Scaling groups, security groups, Amazon Elastic Compute Cloud (Amazon EC2) instances, other AWS resources, and AWS Identity and Access Management (IAM) roles to manage resources from AWS deployment services.

## Software Development Lifecycle with AWS Cloud

Determine how to manage the SDLC with AWS services based on environment variables, infrastructure tiers, and the type of applications or services you launch. Each of the AWS services you use in a deployment has its own configuration, or service-specific settings,

which affect your deployment implementation. Consider the types of deployments that you will perform so that you can implement the most appropriate services into your seamless chain of events or a pipeline. This seamless or “continuous” chain of events on the AWS Cloud is the continuous integration/continuous deployment (CI/CD) pipeline.

## Continuous Integration/Continuous Deployment

The CI/CD pipeline helps developers implement continuous builds, tests, and code deployments with multiple AWS resources and a continuous integration server. You can integrate AWS Elastic Beanstalk with the CI/CD pipeline as one of the deployment resources. You can also use AWS CodeCommit as a CI/CD resource paired with a Git repository, from which Elastic Beanstalk can extract and deploy code.

*Continuous integration* (CI) is the software development practice in which you continuously integrate (or check in) all code changes into a main branch of a central repository. This practice enables you to verify your code changes early and often with an automated build and test process. Whenever you check in code changes, engineers can automate various processes, such as building assets and testing code syntax. By implementing continuous integration practices, teams become more productive and develop new features more quickly. Teams also write scripts to validate the functionality and improve the quality of the software being released.

*Continuous delivery* (CD) is the software development practice in which all code changes are automatically prepared and always deployable (ready to go into production) at a single step.

Continuous delivery extends continuous integration to include testing production-like stages and running verification testing against those deployments. Although continuous delivery can extend to a production deployment, it requires manual intervention between a code check-in and when that code is available for customer use.

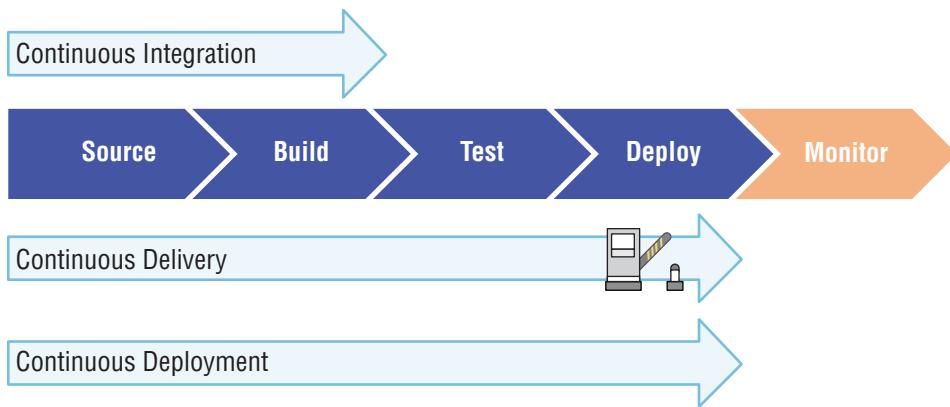
Practicing continuous delivery means that teams gain a greater level of certainty that their software will work in production.

Continuous deployment extends continuous delivery and is the automated release of software to customers, from check-in through production, without human intervention. Continuous deployment helps customers gain value quickly from the code base, with the development team getting faster feedback on the changes made.

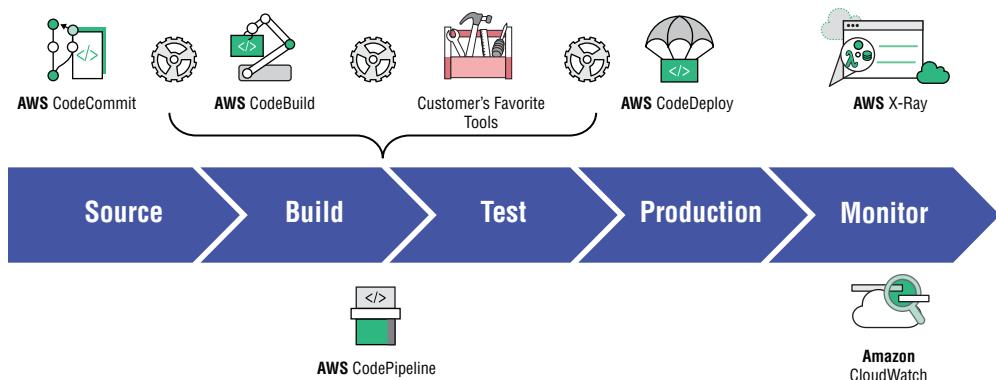


An important distinction between continuous delivery and continuous deployment is that in continuous deployment, changes are automatically released to production after build/test stages; there is no manual approval step.

Figure 6.2 displays the CI/DI pipeline.

**FIGURE 6.2** CI/DI pipeline

The CI/DI pipeline integrates with other AWS Code services, as illustrated in Figure 6.3.

**FIGURE 6.3** AWS Code services

**AWS CodePipeline** *AWS CodePipeline* is a service for fast and reliable application updates. You can model and visualize the software release process. To build, test, and deploy your code every time there is a code change, integrate this service with third-party tools and AWS.

**AWS CodeCommit** *AWS CodeCommit* is a secure, highly scalable, managed source-control service that hosts private Git repositories. It enables you to store and manage assets (such as documents, source code, and binary files) privately in the AWS Cloud.

**AWS CodeBuild** *AWS CodeBuild* compiles source code, runs tests, and produces ready-to-deploy software packages. There is no need to manage build servers.

**AWS CodeDeploy** AWS *CodeDeploy* automates code deployments to any instance. It handles the complexity of updating your applications, which avoids downtime during application deployment. It deploys to Amazon EC2 or on-premises servers, in any language and on any operating system. It also integrates with third-party tools and AWS.

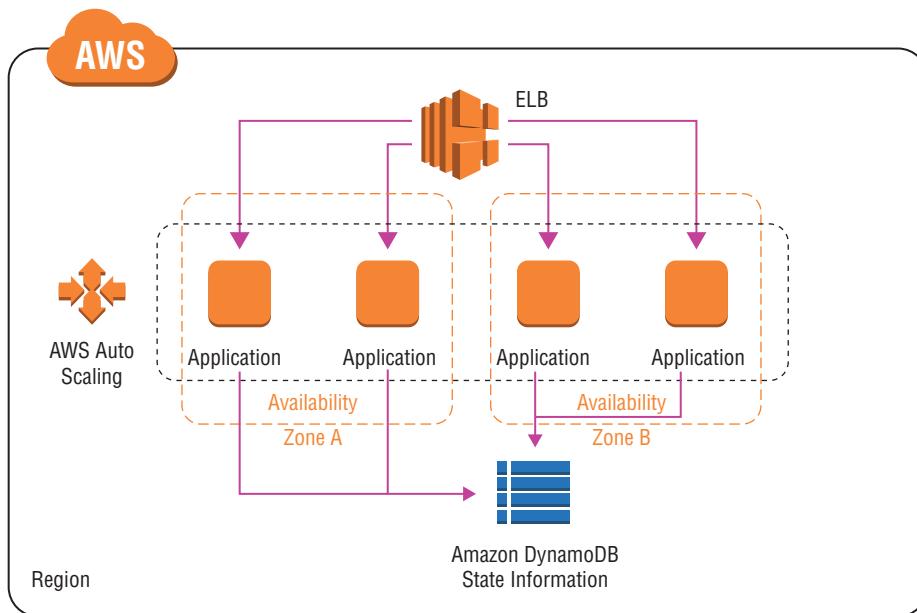
## Deploying Highly Available and Scalable Applications

Load balancing is an integral part to directing and managing traffic among your instances. As you launch applications in your environments, you will want them to have high performance and high availability for your users. To enable both of these features, a load balancer will be necessary.

*Elastic Load Balancing* (ELB) supports three types of load balancers: Application Load Balancers, Network Load Balancers, and Classic Load Balancers. You select a load balancer based on your application needs.

- The *Application Load Balancer* provides advanced request routing targeted at delivery of modern application architectures, including microservices and container-based applications. It simplifies and improves the security of your application by ensuring that the latest Secure Sockets Layer (SSL)/Transport Layer Security (TLS) ciphers and protocols are used at all times. The Application Load Balancer operates at the request level (Layer 7) to route HTTP/HTTPS traffic to its targets: Amazon EC2 instances, containers, and IP addresses based on the content of the request. It is ideal for advanced load balancing of HTTP and HTTPS traffic.
- The *Network Load Balancer* operates at the connection level (Layer 4) to route TCP traffic to targets: Amazon EC2 instances, containers, and IP addresses based on IP protocol data. It is the best option for load balancing of TCP traffic because it's capable of handling millions of requests per second while maintaining ultra-low latencies. Network Load Balancer is optimized to handle sudden and volatile traffic patterns while using a single static IP address per Availability Zone. It is integrated with other popular AWS services, such as AWS Auto Scaling, Amazon Elastic Container Service (Amazon ECS), and AWS CloudFormation. Amazon ECS provides management for deployment, scheduling, and scaling, and management of containerized applications.
- The *Classic Load Balancer* provides basic load balancing across multiple Amazon EC2 instances and operates at both the request level and the connection level. The Classic Load Balancer is intended for applications that were built within the EC2-Classic network. When you're using Amazon Virtual Private Cloud (Amazon VPC), AWS recommends the Application Load Balancer for Layer 7 and Network Load Balancer for Layer 4).

Figure 6.4 displays the flow for deploying highly available and scalable applications.

**FIGURE 6.4** Deploying highly available and scalable applications

The flow for deploying highly available and scalable applications includes the following components:

- Multiple Availability Zones and AWS Regions.
- Health check and failover mechanism.
- Stateless application that stores the session state in a cache server or database.
- AWS services that help you to achieve your goal. For example, Auto Scaling helps you maintain high availability and scalability.



Elastic Load Balancing and Auto Scaling are designed to work together.

## Deploying and Maintaining Applications

AWS provides several services to manage your application and resources, as shown in Figure 6.5.

**FIGURE 6.5** Deployment and maintenance services

With AWS Elastic Beanstalk, you do not have to worry about managing the infrastructure for your application. You deploy your application, such as a Ruby application, in a Ruby container, and Elastic Beanstalk takes care of scaling and managing it.

AWS OpsWorks is a configuration and deployment management tool for your Chef or Puppet resource stacks. Specifically, *OpsWorks for Chef Automate* enables you to manage the lifecycle of your application in layers with Chef recipes. It provides custom Chef cookbooks for managing many different types of layers so that you can write custom Chef recipes to manage any layer that AWS does not support.

AWS CloudFormation is infrastructure as code. The service helps you model and set up AWS resources so that you can spend less time managing them. It is a template-based tool, with formatted text files in JSON or YAML. You can create templates to define what AWS infrastructure you want to build and any relationships that exist among the parts of your AWS infrastructure.



Use AWS CloudFormation templates to provision and configure your stack resources.

## Automatically Adjust Capacity

Use AWS Auto Scaling to monitor the AWS resources that are part of your application. The service automatically adjusts capacity to maintain steady, predictable performance. You can build scaling plans to manage your resources, including Amazon EC2 instances and Spot Fleets, Amazon Elastic Container Registry (Amazon ECR) tasks, Amazon DynamoDB tables and indexes, and Amazon Aurora Replicas.

AWS Auto Scaling makes scaling simple, with recommendations that allow you to optimize performance, costs, or balance between them. If you are already using EC2 Auto Scaling to scale your Amazon EC2 instances dynamically, you can now combine it with AWS Auto Scaling to scale additional resources for other AWS services. With AWS Auto Scaling, your applications have the right resources at the right time.

## Auto Scaling Groups

An Auto Scaling group contains a collection of Amazon EC2 instances that share similar characteristics. This collection is treated as a logical grouping to manage the scaling of instances. For example, if a single application operates across multiple instances, you might want to increase the number of instances in that group to improve the performance of the application or decrease the number of instances to reduce costs when demand is low.

You can use the Auto Scaling group to scale the number of instances automatically based on criteria that you specify or maintain a fixed number of instances even if an instance becomes unhealthy. This automatic scaling and maintaining the number of instances in an Auto Scaling group make up the core functionality of the EC2 Auto Scaling service.

An Auto Scaling group launches enough Amazon EC2 instances to meet its desired capacity. The Auto Scaling group maintains this number of instances by performing periodic health checks on the instances in the group. If an instance becomes unhealthy, the group terminates the unhealthy instance and launches another instance to replace it.

You can use scaling policies to increase or decrease the number of instances in your group dynamically to meet changing conditions. When the scaling policy is in effect, the Auto Scaling group adjusts the desired capacity of the group and launches or terminates the instances as needed. You can also manually scale or scale on a schedule.

## AWS Elastic Beanstalk

*AWS Elastic Beanstalk* is an AWS service that you can use to deploy applications, services, and architecture. It provides provisioned scalability, load balancing, and high availability. It uses common languages, including Java, .NET, PHP, Node.js, Python, Ruby, Go, and Docker, on common-type web servers, such as Apache, NGINX, Passenger, and IIS.



Elastic Beanstalk charges only for the resources you use to run your application.

Elastic Beanstalk is a solution that enables the automated deployments and management of applications on the AWS Cloud. Elastic Beanstalk can launch AWS resources automatically with Amazon Route 53, AWS Auto Scaling, Elastic Load Balancing, Amazon EC2, and Amazon Relational Database Service (Amazon RDS) instances, and it allows you to customize additional AWS resources.

Deploy applications without worrying about managing the underlying technologies, including the following:

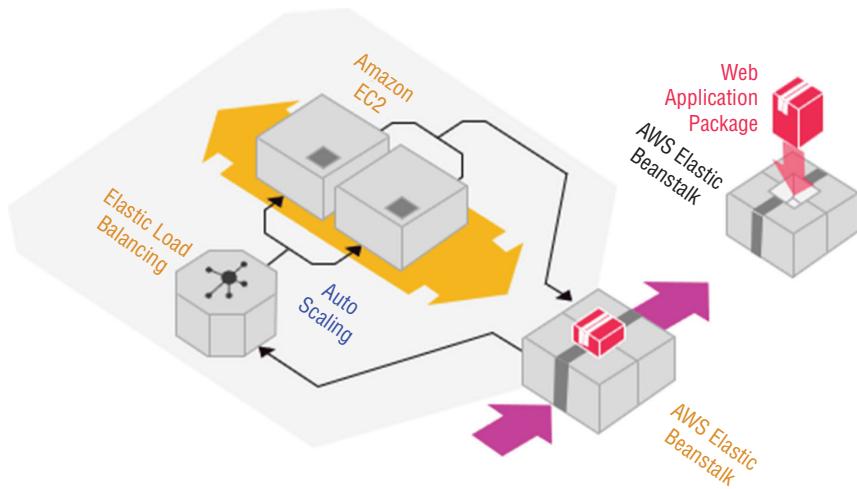
### Components

- Environments
- Application versions
- Environment configurations

### Permission Model

- Service role
- Instance profile

Figure 6.6 displays the Elastic Beanstalk underlying technologies.

**FIGURE 6.6** AWS Elastic Beanstalk underlying technologies

Elastic Beanstalk supports customization and N-tier architectures. It mitigates common manual configurations required in a traditional infrastructure deployment model. With Elastic Beanstalk, you can also create repeatable environments and reduce redundancy, thus rapidly updating environments and facilitating service-managed application stacks. You can deploy multiple environments in minutes and use various automated deployment strategies.



AWS Elastic Beanstalk allows you to focus on building your application.

## Implementation Responsibilities

AWS and our customers share responsibility for achieving a high level of software component security and compliance. This shared model reduces your operational burden. The service you select determines the level of your responsibility. For example, Elastic Beanstalk helps you perform your side of the shared responsibility model by providing a managed updates feature. This feature automatically applies patch and minor updates for an Elastic Beanstalk supported platform version.

### Developer Teams

Using AWS Elastic Beanstalk, you build full-stack environments for web and worker tiers. The service provides a preconfigured infrastructure.

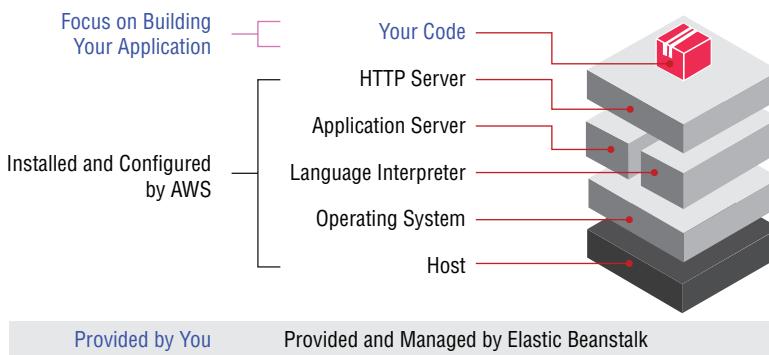
- Single-instance (development, low cost)
- Load balanced, AWS Auto Scaling (production)

## Elastic Beanstalk Responsibilities

Elastic Beanstalk provisions the necessary infrastructure resources, such as the load balancer, Auto Scaling group, security groups, and database (optional). It also provides a unique domain name for your application (for example, `yourapp.elasticbeanstalk.com`).

Figure 6.7 displays Elastic Beanstalk responsibilities.

**FIGURE 6.7** AWS Elastic Beanstalk responsibilities



## Working with Your Source Repository

Developer teams generally begin their SDLC processes by managing their source code in a source repository. Uploading and managing the multiple changes on application source code is a repeated process. With Elastic Beanstalk, you can create an application, upload a version of the application as a source bundle, and provide pertinent information about the application.

The first step is to integrate Elastic Beanstalk with your source code to create your source bundle. As your source repository, you can install Git for your applications or use an existing repository and map your current branch from a local repository in Git to retrieve the source code.

Alternatively, you can use AWS CodeCommit as a source control system to retrieve source code. By using Elastic Beanstalk with the AWS CodeCommit repository, you extract from a current branch on CodeCommit.

To deploy a new application or application version, Elastic Beanstalk works with source bundles or packaged code. Prepare the code package with all of the necessary code dependencies and components.

Elastic Beanstalk can either retrieve the source bundle from a source repository or download the bundle from an Amazon Simple Storage Service (Amazon S3) bucket. You can use the IAM role to grant Elastic Beanstalk access to all services. The service accesses the source bundle from the location you designate, extracts the components from the bundle, deploys new application versions by launching the code, creates and

configures the infrastructure, and allocates the platform on Amazon EC2 instances to run the code.

The application runs on the resources and instances that the service generates. Your configuration for these resources and your application will become your environment settings, supporting the entire configuration of your deployment. Each deployment has an auto-incremented deployment identity (ID), so you are able to manage your multiple running deployments. Think of these as multiple running code releases in the AWS Cloud.



You can also work with different hosting services, such as GitHub or Bitbucket, with your code source.

## Concepts

AWS Elastic Beanstalk enables you to manage all the resources that run your application as environments. This section describes some key Elastic Beanstalk concepts.

### Application

Elastic Beanstalk focuses on managing your applications as environments and all of the resources to run them. Each application that launches in the service is a logical collection of environment variables and components, application versions, and environment configurations.

### Application Versions

Application versions are iterations of the application's deployable code. Application versions in Elastic Beanstalk point to an Amazon S3 object with the code source package. An application can have many versions, with each version being unique. You can deploy and access any application version at any time. For example, you may want to deploy different versions for different types of tests.

### Environment

Each Elastic Beanstalk environment is a separate version of the application, and that version's AWS Cloud components deploy onto AWS resources to support that version. Each environment runs one application version at a time, but you can run multiple environments, with the same application on each, along with its own customizations and resources.

### Environment Tier

To launch an environment, you must first choose an environment tier. Elastic Beanstalk provisions the required resources to support both the infrastructure and types of requests

the application will support. The environment can launch and access other AWS resources. For example, it may pull tasks from Amazon Simple Queue Service (Amazon SQS) queues or store temporary configuration files in Amazon S3 buckets (according to your customizations). Each environment will then have an environment configuration—a collection of settings and parameters based on your customizations that define associated resources and how the environment will work.

## Environment Configuration

You can change your environment to create, modify, delete, or deploy resources and change the settings for each. Your environment configuration saves to a configuration template exclusive to each environment and is accessible by either the Elastic Beanstalk application programming interface (API) calls or the service's command line interface (EB CLI).

In Elastic Beanstalk, you can run either a web server environment or a worker environment. Figure 6.8 displays an example of a web server environment running in Elastic Beanstalk with Amazon Route 53 as the domain name service (DNS) and ELB to route traffic to the web server instances.

**FIGURE 6.8** Application running on AWS Elastic Beanstalk

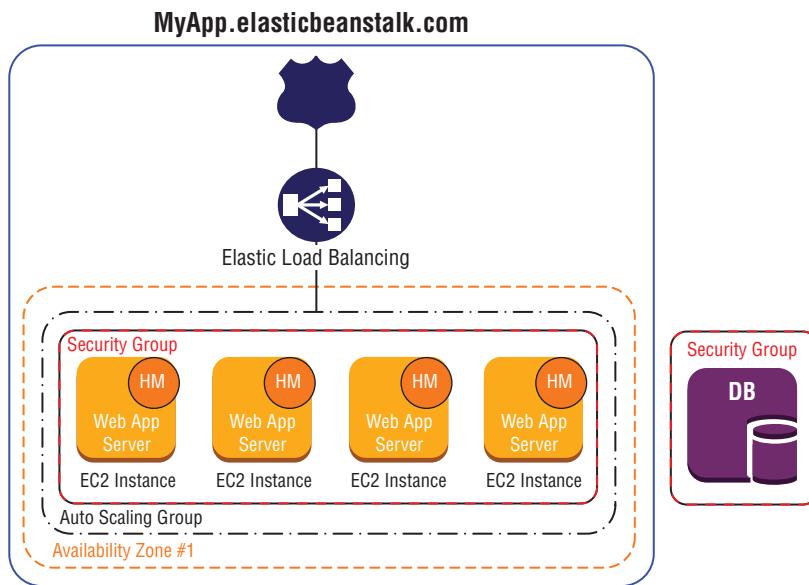
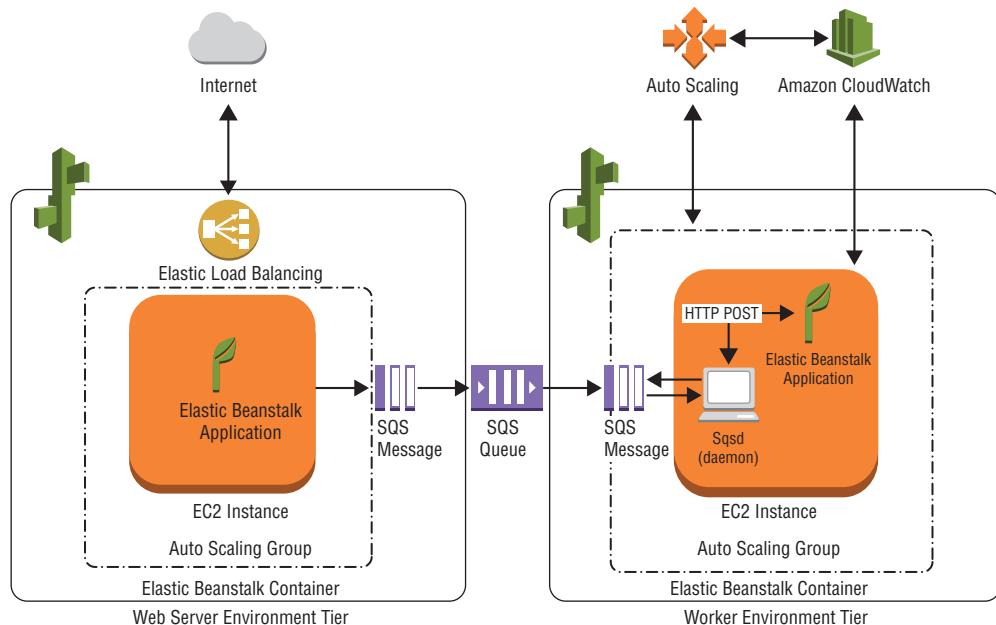


Figure 6.9 shows a worker environment architecture, where AWS resources create configurations, such as Auto Scaling groups, Amazon EC2 instances, and an IAM role, to manage resources for your worker applications.

**FIGURE 6.9** Worker tier on AWS Elastic Beanstalk

For the worker environment tier, Elastic Beanstalk creates and provisions additional resources and files to support the tier. This includes services like Amazon SQS queues operating between worker applications, AWS Auto Scaling groups, security groups, and EC2 instances.

The worker environment infrastructure uses all of your customization and provision resources to determine the types of requests it receives.

### Docker Containers

You can also use Docker containers with Elastic Beanstalk to run your applications from a container. Install Docker, choose the software you require, and select the Docker images you want to launch. Define your runtime environment, platform, programming language, and application dependencies and tools. Docker containers are self-contained and include configurations and software that you specify for your application to run. Each Docker container restarts automatically if another container crashes. When you choose to deploy your applications with Docker containers, your infrastructure is provisioned with capacity provisioning, load balancing, scaling, and health monitoring, much like a noncontainer environment. You can continue to manage your application and the AWS resources you use.

Docker requires platform configurations that enable you to launch single or multicontainer deployments. A single container deployment launches a single Docker image, and your application uses a single container configuration for a single Amazon EC2 instance.

A multicontainer deployment uses the Amazon ECS to launch a cluster of containers with Docker images. A multicontainer configuration is applied to each instance. You can also run preconfigured Docker platform configurations with generic customization for popular software stacks that you want to use for your application.

## AWS Elastic Beanstalk Command Line Interface

Elastic Beanstalk has its own command line interface separate from the AWS CLI tool. To create deployments from the command line, you download and install the AWS Elastic Beanstalk CLI (EB CLI).

Table 6.1 lists common EB CLI commands.

**TABLE 6.1** Common AWS Elastic Beanstalk Commands

Command	Definition
eb init application-name	Sets default values for Elastic Beanstalk applications with the EB CLI configuration wizard
eb create	Creates a new environment and deploys an application version to it
eb deploy	Deploys the application source bundle from the initialized project directory to the running application
eb clone	Clones an environment to a new environment so that both have identical environment settings
eb codesource	Configures the EB CLI to deploy from an AWS CodeCommit repository, or disables AWS CodeCommit integration and uploads the source bundle from your local machine

## Customizing Environment Configurations

You can use Elastic Beanstalk to customize the platforms used to support your application and your infrastructure. To do so, create a configuration file in the `ebextensions` directory (or `.ebextensions`) to include with your web application's source code. The configuration file allows for simple and advanced customizations of your environment and contains settings for your AWS resources. To deploy customized resources to support your application source bundle, use YAML to configure the file.

The configuration file has several sections. The `option_settings` section defines your configuration option values for your AWS resources. The `resources` section adds further customization in your application environment beyond the service functionality, which includes AWS CloudFormation-supported resources that Elastic Beanstalk can access and

run. The remaining sections allow for fine-grained configurations to integrate packages, sources, files, and container commands.



Launch environments from integrated development environment (IDE) tools to avoid poorly formatted configurations and source bundles that could cause unrecoverable failures.

You apply configuration files in the ebextensions directory to Elastic Beanstalk stacks. The stacks are the AWS resources that you allocate for your infrastructure and application. If you have any resource, such as Amazon VPC, Amazon EC2, or Amazon S3, that was updated or configured, these files deploy with your changes. You can zip your ebextension files, upload, and apply them to multiple application environments. You can view your environment variables in option\_settings for future evaluation or changes. These are accessible from the AWS Management Console, command line, and API calls.



You can view Elastic Beanstalk stacks in AWS CloudFormation, but always use the Elastic Beanstalk service and ebextensions to make modifications. This way, edits and modifications to the application stacks are simplified without introducing unrecoverable failures.

Elastic Beanstalk generates logs that you can view to troubleshoot your environments and resources. The logs display Amazon EC2 operational logs and logs that are specific to servers running for your applications.

## Integrating with Other AWS Services

Elastic Beanstalk automatically integrates or manages other AWS services with application code to provision efficient working environments. However, you might find it necessary to add additional services, such as Amazon S3 for content storage or Amazon DynamoDB for data records, to work with an environment. To grant access between any integrated service and Elastic Beanstalk, you must configure permissions in IAM.

### Amazon S3

You can use Amazon S3 to store static content you want to integrate with your application and point directly to objects you store in Amazon S3 from your application or from other resources. In addition to setting permissions in IAM policies, take advantage of presigned URLs for controlled Amazon S3 GET and PUT operations.

### Amazon CloudFront

You can integrate your Elastic Beanstalk environment with Amazon CloudFront, which provides content delivery and distribution through the use of edge locations throughout the world. This can decrease the time in which your content is delivered to you, as the content

is cached and routed through the closest edge location serving you. After you deploy your application on Elastic Beanstalk, use the Amazon CloudFront content delivery network (CDN) to cache static content from your application. To identify the source of your content in Amazon CloudFront, you can use URL path patterns to cache your content and then retrieve it from the cache. This approach serves your content more rapidly and offloads requests directly sourced from your application.

## AWS Config

With *AWS Config*, you can visualize configuration history and how configurations evolve over time. Tracking changes helps you to fulfill compliance obligations and meet auditing requirements. You can integrate AWS Config directly with your application and its versions or your Elastic Beanstalk environment. *You can customize AWS Config to record changes per resource, per region, or globally.* In the AWS Config console, you can select Elastic Beanstalk resource types to record specific applications and environment resources. You can view the recorded information in the AWS Config dashboard under Resource Inventory.

## Amazon RDS

Various options are available for creating databases for your environment, such as Amazon Relational Database Service (Amazon RDS) for SQL databases and Amazon DynamoDB for NoSQL databases. Elastic Beanstalk can create a database and store and retrieve data for any of your environments. Each service has its own features to handle scaling, capacity, performance, and availability.

To store, read, or write to your data records, you can set up an Amazon RDS database instance or an Amazon DynamoDB table by using the same configuration files for your other service option settings. You must create connections to the database, which require you to set up password management in Elastic Beanstalk. Your configurations are saved in the ebextensions directory. You can also create direct connections, within your application code or application configuration files, to both internal and external databases. When using Amazon RDS, avoid accidentally deleting and re-creating databases without a properly installed backup. To reduce the risk of losing data, take a manual snapshot of the master Amazon RDS database immediately before deleting.



If you create periodic tasks with a worker environment, Elastic Beanstalk automatically creates an Amazon DynamoDB table to perform leader election and stores task information.

## Amazon ElastiCache

For caching capabilities, you can integrate Amazon ElastiCache service clusters with the Elastic Beanstalk environment. If you use a nonlegacy container, you can set your configuration files to use the supported container and then offload requests to the cache cluster.

Doing so enables you to increase the performance of your application and databases running in your Elastic Beanstalk environment.

## AWS Identity and Access Management Roles

Elastic Beanstalk integrates with AWS Identity and Access Management (IAM) roles to enable access to the services you require to run your architecture.

When you launch the service to create an environment, a default service role and instance profile are created for you through the service API. Managed policies for resources permissions are also attached, including policies for Elastic Beanstalk instance health monitoring within your infrastructure and platform updates that can be made on behalf of the service. These policies, called `AWSElasticBeanstalkEnhancedHealth` and `AWSElasticBeanstalkService`, attach to the default service role and enable the default service role to specify a trusted entity and trust policy.

When you use commands from the EB CLI, the role allows automatic management of the AWS Cloud that services you run. The service creates an environment, if you don't identify it specifically; creates a service-linked role; and uses it when you spin up a new environment. To create the environment successfully, the `CreateServiceLinkedRole` policy must be available in your IAM account.

You use IAM roles to automate the management of allocated services for your application through Elastic Beanstalk. With IAM, you can also launch code with inline policies. It is important to understand how the service creates and uses the roles to keep your application and data secure.



For IAM to manage the policies for the account better, create policies at the account level.

## Deployment Strategies

A *deployment* is the process of copying content and executing scripts on instances in your deployment group. To accomplish this, AWS CodeDeploy performs the tasks outlined in the AppSpec configuration file. For both Amazon EC2 on-premises instances and AWS Lambda functions, the deployment succeeds or fails based on whether individual AppSpec tasks complete successfully.

After you have created a deployment, you can update it as your application or service changes. You can update a deployment by adding or removing resources from a deployment, thus updating the properties of existing resources in a deployment.



A serverless application is typically a combination of AWS Lambda and other AWS services.

To create seamless deployments, choose an effective deployment strategy. Each strategy has specific advantages relative to different use cases. Appropriate strategies help create deployments where you experience minimal or no downtime, and you can apply the strategy for different purposes within your environments. Each change needs a strategy that best fits your application deployments.

## All-at-Once and In-Place Deployments

An *all-at-once deployment* applies updates to all your instances at once. When you execute this strategy, you experience downtime, as all instances receive the change at the same time.

This is an appropriate strategy for simple, immediate update requirements when it's not critical to have your application always available, and you're comfortable with the site being offline for a short duration. To enable all-at-once updates, set a deployment policy either in the AWS Management Console or in the command line (`DeploymentPolicy`).

When you perform an *in-place deployment*, AWS CodeDeploy stops currently running applications on the target instance, deploys the latest revision, restarts applications, and validates successful deployment. In-place deployments can support the automatic configuration of a load balancer. In this case, the instance is deregistered from the load balancer before deployment and registered again after the deployment processes successfully.

In-place updates are also available for your platform updates, such as a coding-language platform update for a web server. Select the new platform and then run the update from the AWS Management Console or command line directly as a platform update.



AWS Lambda does not support in-place deployments.

## Rolling Deployments

A *rolling deployment* applies changes to all of your instances by rolling the updates from one instance to another. Elastic Beanstalk can deploy configuration changes in batches. This approach reduces possible downtime during implementation of the change and allows available instances to run while you deploy.

As updates are applied in a batch, the batch will be out of service for a short period while the changes propagate and then relaunch with the new configuration. When the change is complete, the service moves on to the next batch of instances to apply the changes. With this strategy, you can implement both periodic changes and pauses between updates. For example, you might specify a time to wait between health-based updates so that instances must pass health checks before moving on to the next batch. If the rolling update fails, the service begins another rolling update for a rollback to the previous configuration.

Rolling updates include changes for Auto Scaling group configurations, Amazon EC2 instance configurations, and Amazon VPC settings. It is an effective method for updating an application version on fleets of instances through the Elastic Beanstalk service. To enable

rolling updates, set a deployment policy either in the AWS Management Console or in the command line (`DeploymentPolicy`) and choose this strategy along with specific options. You can select *Rolling* or *Rolling with additional batch*. By using *Rolling with additional batch*, you can launch a new batch of instances before you begin to take instances out of service for your rolling updates. This option provides an available batch for rollback from a failed update. After the deployment is successfully executed, Elastic Beanstalk terminates the instances from the additional batch. This is helpful for a critical application that must continue running with less downtime than the standard rolling update.

## Blue/Green Deployment

When high availability is critical for applications, you may want to choose a *blue/green deployment*, where your newer environment will be separate from your existing environment. The running production environment is considered the *blue environment*, and the newer environment with your update is considered the *green environment*. When your changes are ready and have gone through all tests in your green environment, you can swap the CNAMEs of the environments to redirect traffic to the newer running environment. This strategy provides an instantaneous update with typically zero downtime.

When you deploy to AWS Lambda functions, blue/green deployments publish new versions of each function. Traffic shifting then routes requests to the new functioning versions according to the deployment configuration you define.

If your infrastructure contains Amazon RDS database instances, the data does not automatically transfer to the new environment. Without performing backups, you will experience data loss when you use the blue/green strategy. If you have Amazon RDS instances in your infrastructure, implement a different deployment strategy or a series of steps to create snapshot backups outside of Elastic Beanstalk before you execute this type of deployment.

## Immutable Deployment

An *immutable deployment* is best when an environment requires a total replacement of instances, rather than updates to an existing part of an infrastructure. This approach implements a safety feature for updates and rollbacks. Elastic Beanstalk creates a temporary Auto Scaling group behind your environment's load balancer to contain the new instances with the updates you apply. If the update fails, the rollback process terminates the Auto Scaling group. Immutable instances implement a number of health checks. If all instances pass these checks, Elastic Beanstalk transfers the new configurations to the original Auto Scaling group, providing an additional check before you apply your changes to other instances. Enhanced health reports evaluate instance health in the update. After the updates are made, Elastic Beanstalk deletes the temporary Auto Scaling group of the older instances.



During this type of deployment, your capacity doubles for a short duration between the updates and terminations of instances. Before you use this strategy, verify that your instances have a low on-demand limit and enough capacity to support immutable updates.

See Table 6.2 for feature comparisons between all deployment strategies. The checkmark indicates options that the deployment strategy supports.

**TABLE 6.2** Deployment Strategies

Method	Impact of Failed Deployment	Deploy Time	Zero Downtime	No DNS Change	Rollback Process	Code Deployed To
All-at-once	Downtime	(L)		✓	Redeploy	Existing instances
In-place	Downtime	(L)		✓	Redeploy	Existing instances
Rolling	Single batch out of service; any successful batches before failure running new application version	(L)(L)	✓	✓	Redeploy	Existing instances
Rolling with additional batch	Minimal if first batch fails; otherwise, similar to Rolling	(L)(L)(L)	✓	✓	Redeploy	New and existing instances
Blue/Green	Minimal	(L)(L)(L) (L)	✓		Swap URL	New instances
Immutable	Minimal	(L)(L)(L) (L)	✓	✓	Redeploy	New instances

## Container Deployments

Elastic Beanstalk enables you to launch your applications with Docker containers. With a Docker container, you can create a runtime environment with all of the dependencies, packages, and tools that your application may require to run. Your container can have all of the configurations necessary for your application. By using Docker with Elastic Beanstalk, you have the infrastructure for capacity provisioning, scalability, load balancing, and health monitoring for the instances that run on containers. The containers integrate with your

Amazon VPC for network requirements and with IAM to enable resource management. You can launch different software engines with containers to provide various options and third-party tools to run containers.

You can choose from single container configurations and multicontainer configurations. A single container runs one container per instance. A multicontainer runs multiple applications or engines on one instance, with all of the software and settings you require. Preconfigured options are available with Docker, and you can integrate them with instances that run in your architecture through Elastic Beanstalk.

## Monitoring and Troubleshooting

After you launch your code, check on its performance and availability. You can monitor statistics and view information about the health of your application, its environment, and specific services from the AWS Management Console. Elastic Beanstalk also creates alerts that trigger at established thresholds to monitor your environment's health. In the AWS Management Console, the AWS Elastic Beanstalk Monitoring page shows aggregated statistics and graphs for your applications and resources. Each environment is color-coded to indicate the environment's status. You can see at a glance whether your environment is available online at any point in time. Metrics gathered by the resources in your environment are published to Amazon CloudWatch in five-minute intervals. You can adjust the time range for the statistics and graphs and customize your views of the metrics.

Figure 6.10 shows an example of the statistics that you can view for your environment.

**FIGURE 6.10** Health dashboard on AWS Elastic Beanstalk

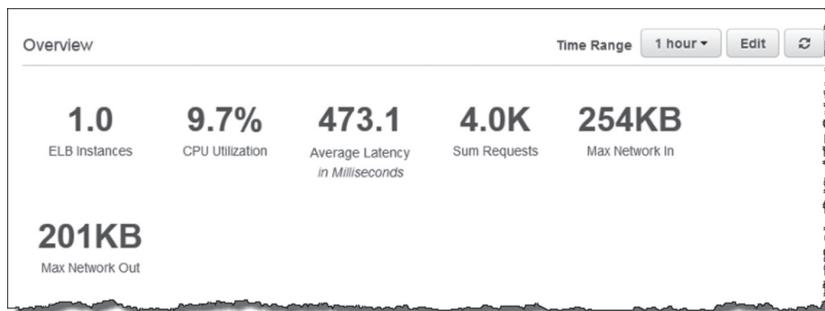


Figure 6.11 shows an example of the graphs that you can view.

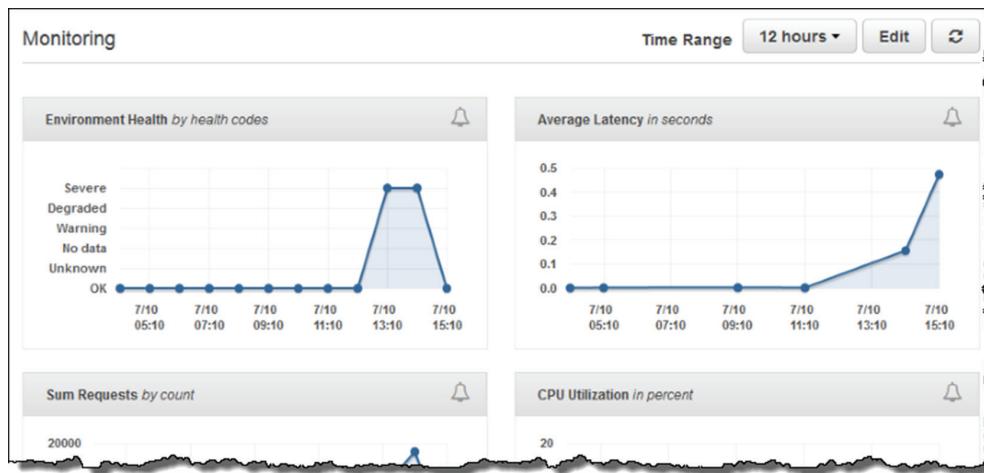
**FIGURE 6.11** Metrics for monitoring on AWS Elastic Beanstalk

Table 6.3 defines the AWS Elastic Beanstalk Monitoring page colors.

**TABLE 6.3** AWS Elastic Beanstalk Health Page Color Definitions

Color	Description
Gray	Your environment is being updated.
Green	Your environment has passed the most recent health check. At least one instance in your environment is available and taking requests.
Yellow	Your environment has failed one or more health checks. Requests to your environment are failing.
Red	Your environment has failed three or more health checks, or an environment resource has become unavailable. Requests are consistently failing.

By default, Elastic Beanstalk displays Amazon EC2, Auto Scaling, and Elastic Load Balancing metrics for your application environments. These metrics are available to you on your AWS Elastic Beanstalk Monitoring page as soon as you deploy your application environment. You can access the health status from the AWS Management Console or the EB CLI.

## Basic Health Monitoring

To access the health status from the AWS Management Console, select the Elastic Beanstalk service and then select the tab for your specific application environment. An

environment overview shows your architecture's instance status details, resource details, and filter capabilities. Health statuses are indicated in four distinct colors.

To access the health status from the EB CLI, enter the `eb health` command. The output shows the environment and the health of associated instances. Enhanced health reporting also provides the following seven health statuses, which are single-word descriptors that provide a better indication of the state of your environment:

```
ok warning degraded severe info pending unknown
```

You can also use the `eb status` command in the EB CLI or the `DescribeEnvironments` API call to retrieve the health status for an environment. You can check the health of the overall environment or the individual services of Amazon EC2 or an Elastic Load Balancing load balancer. Health checks on your Elastic Load Balancing port execute both for the default port 80 and a custom Elastic Load Balancing port/path.

For GET requests with the load balancer, `200 OK` is the default success code and indicates a healthy status. The service can also return `400` level responses. You can also configure a health check URL for custom static page responses.



Be sure to adjust the caching time to live for any health check static pages or URLs in Amazon CloudFront or for any caching mechanism you may use.

Elastic Beanstalk also reports missing configurations or other issues that could affect the health of the application environment.

## Enhanced Health Monitoring

There are two types of reporting: the default health information about your resources and the enhanced health reporting that provides you more information for monitoring health.

You can use the enhanced health reporting feature to gather additional resource data and display graphs and statistics of environment health in greater detail. This is important when you deploy multiple versions of your application and when you need to analyze factors that could be degrading your application's availability or performance. You can view these details in the AWS Elastic Beanstalk Monitoring page from the AWS Management Console. These reports require the creation of two IAM roles: a *service role* to allow access between the services and Elastic Beanstalk and an *instance profile* to write logs into an Amazon S3 bucket.



Running the enhanced health report requires a version 2 or newer platform configuration that supports all platforms except Windows Server with IIS. The enhanced health reports provide data directly to Elastic Beanstalk and do not run through Amazon CloudWatch.

By default, health monitoring on Elastic Beanstalk does not publish metrics to Amazon CloudWatch, so you are not charged for the metrics. There are also custom metrics that you can run and view, for which you are not charged a fee. You can enable custom metrics by using the PutMetricData operation in worker environments. For example, you might have an Amazon SQS daemon that publishes custom metrics for environment health under the same environment namespace. You can also enable custom metrics from Amazon CloudWatch, but AWS charges for these additional metrics you publish to your monthly Amazon CloudWatch. To save costs, use the available metrics on the Elastic Beanstalk service, or enable the custom metrics that you need, paying only for what you use.

Elastic Beanstalk runs a health agent to provide detailed health resource data for enhanced health monitoring. The health agent runs in the Amazon Machine Image (AMI) for each instance operating system on a platform configuration for your application. The agent analyzes system metrics and logs to communicate the health status to Elastic Beanstalk. You receive alerts, data, and actionable insights that you can use to monitor your applications and understand, prevent, and respond to performance issues.

You can monitor recent health events that you have enabled on Elastic Beanstalk in real time. There are several health event types that can change as an environment transitions from the create state to the run state. Figure 6.12 displays the health events available in the AWS Elastic Beanstalk Monitoring page and examples of the details that allow you to respond to issues identified.

**FIGURE 6.12** Events on AWS Elastic Beanstalk

Recent Events			Show All
Time	Type	Details	
2015-07-09 16:06:40 UTC-0700	INFO	Environment health has transitioned from Severe to Ok	
2015-07-09 16:04:41 UTC-0700	WARN	Environment health has transitioned from Ok to Severe. 100.0 % of the requests are erroring with HTTP 4xx	
2015-07-09 15:10:45 UTC-0700	INFO	Environment health has transitioned from Info to Ok	
2015-07-09 15:09:26 UTC-0700	INFO	Environment update completed successfully.	
2015-07-09 15:09:26 UTC-0700	INFO	Successfully deployed new configuration to environment.	

Elastic Beanstalk integrates with AWS CloudTrail to capture Elastic Beanstalk API calls as log files that you can store in an Amazon S3 bucket. To view additional actions occurring with your running resources, you can also capture AWS API calls in your code using AWS CloudTrail.

# Summary

In this chapter, you learned about the features of Elastic Beanstalk, how to automate deployments for your multi-tier architectures, and different deployment strategies. You also discovered options for configuring your environments and managing your resources with services such as IAM, Amazon VPC, Amazon EC2, and Amazon S3.

## Exam Essentials

**Know how to deploy AWS Elastic Beanstalk.** Know how to deploy an application AWS Elastic Beanstalk and what platforms it supports. To complete the exam successfully, you should also understand how the architectures and services interact with the web, application, and database tiers. Focus on foundational services and how you create and work with Elastic Beanstalk.

**Know about ebextensions.** Understand ebextensions and the part they play in the service configuration. Be able to recognize the stacks you create and how to change them.

**Know about Elastic Beanstalk resources.** Understand how to manage resources with Elastic Beanstalk, including IAM. Understand the definitions and differentiate between the functions of the default IAM service role and the instance profile, which are automatically created. Understand permissions for your AWS resources in your environment.

**Know Elastic Beanstalk deployment strategies.** Understand what deployment strategies you can use, their differences, and which ones would be best for different use cases and other resources. Know which strategy offers less downtime and which is best suited for complex changes.

**Know about Elastic Beanstalk components.** Understand all of the components of Elastic Beanstalk, including applications, environments, versions, configurations, and the AWS resources it launches and with which it integrates. Know how to retain or dispose of resources as needed.

**Know about Elastic Beanstalk different environment tiers.** Know the differences between the single-instance tier and the web-server environment tier and when to choose one over the other. Understand the services and features used for both.



---

On the test itself, do not get sidetracked with small details about Elastic Beanstalk. Focus your understanding on how it works as a whole and interacts with other services.

# Resources to Review

AWS Elastic Beanstalk Install:

<https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/eb-cli3-install.html>

AWS Elastic Beanstalk Developer Guide:

<https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/Welcome.html>

AWS Elastic Beanstalk Concepts:

<https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/concepts.html>

Using the EB CLI with AWS CodeCommit:

<https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/eb-cli-codecommit.html>

EB CLI Command Reference:

<https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/eb3-cmd-commands.html>

Deploying AWS Elastic Beanstalk Applications from Docker Containers:

[https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create\\_deploy\\_docker.html](https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create_deploy_docker.html)

Using AWS Elastic Beanstalk with Amazon Relational Database Service:

<https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/AWSHowTo.RDS.html>

Preconfigured Docker Containers:

[https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create\\_deploy\\_dockerpreconfig.html](https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create_deploy_dockerpreconfig.html)

AWS Elastic Beanstalk Supported Platforms:

<https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/concepts.platforms.html>

Enhanced Health Reporting and Monitoring:

<https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/health-enhanced.html>

# Exercises

## EXERCISE 6.1

### Deploy Your Application

In this exercise, you will sign up for an AWS account.

1. Verify that your source code is packaged as a .zip file and is ready to be retrieved from either your source repository directory or an Amazon S3 bucket.  
You can choose a sample application available from the AWS Management Console.
2. Launch the AWS Management Console.
3. To select a region in which to launch the application, select **AWS Elastic Beanstalk > Region**.
4. Select **AWS Elastic Beanstalk Service**.
5. Select **Get Started** or **Create New Application**. The Get Started option takes you through a wizard of guided steps to launch your first application. After this initial start, the Create New Application dialog box will be displayed for future launches.
6. Select the type of application that you want to deploy.
7. Enter an application name.
8. Select the application platform for your code.
9. For your coding language, select the preconfigured platform.
10. Select **Upload your application**.
11. Locate the file directory where your .zip file of your code resides or choose the Amazon S3 bucket with the .zip file and select **Upload**.
12. Choose **Next**.
13. To use the architecture with high availability, select **High Availability**.
14. Modify the configurations for your architecture.
15. Select **Add databases**.
16. Select **RDS database**.
17. Choose **Create App (Application)**.

You now have successfully deployed an application on Elastic Beanstalk.

---

**EXERCISE 6.2****Deploy a Blue/Green Solution**

In this exercise, you will deploy a blue/green solution.

1. Sign in to your AWS account.
2. Navigate to your existing AWS Elastic Beanstalk environment and application or upload the sample.

You can launch a sample application from this location:

<https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/tutorials.html>

3. Clone your environment or launch a new environment with your new version.
4. Deploy the second application version to the new environment. Test that the new version is running.
5. From the new environment dashboard, select **Actions > Swap Environment URLs**.
6. Under **Select an Environment to Swap**, select the current environment name.
7. Choose **Swap**.

The Elastic Beanstalk service swaps the CNAME records between the two environments.

8. On the dashboard, under **Recent Events**, verify the swap.

You have successfully deployed a blue/green solution on AWS Elastic Beanstalk.

---

**EXERCISE 6.3****Change Your Environment Configuration on AWS Elastic Beanstalk**

In this exercise, you will change your environment configuration on AWS Elastic Beanstalk. Use an existing application that is running on Elastic Beanstalk.

1. Sign in to your AWS account.
  2. Navigate to your existing AWS Elastic Beanstalk environment and application.
  3. Choose Configuration.
  4. On the **Capacity Configuration** tab, choose **Modify**.
  5. Under **Auto Scaling Group**, select **Load balanced**.
  6. In the Instances row, change Max to 4 and Min to 2.
-

- 
7. On the **Modify capacity** page, choose **Save**.
  8. On the **Configuration overview** page, choose **Apply**.
  9. On the warning message, choose **Confirm**.

The environment might take a few minutes to update. After your environment is updated, verify your changes.

10. Navigate to the Amazon EC2 service dashboard.
11. Choose **Load Balancers**.
12. Check for the instance-id value that matches your Elastic Beanstalk environment instance-id value and view the load balancers.

You have successfully changed an environment configuration on AWS Elastic Beanstalk.

---

#### EXERCISE 6.4

### Update an Application Version on AWS Elastic Beanstalk

In this exercise, you will update an application version on AWS Elastic Beanstalk from the AWS Management Console.

1. Sign in to your AWS account.
2. Upload a second version of your application that matches the configuration for your current running environment.

If you are using a sample solution application, you can find other versions at the following address:

<https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/GettingStarted.html#GettingStarted.Walkthrough.DeployApp>

3. On the **AWS EB applications** page, select `getting-started-app`.
4. Select `GettingStartedApp-env`.
5. In **Overview**, choose **Upload and Deploy**.
6. Select **Choose File** and upload the next version of your source bundle that you created or downloaded.

The console is automatically populated with the version label based on the name of the archive that you upload. For later deployments, if you use a source bundle with the same name, you must type a unique version label.

---

*(continued)*

**EXERCISE 6.4 (*continued*)**

7. Choose **Deploy**. Elastic Beanstalk deploys your application to your Amazon EC2 instances.

You can view the status of the deployment on the environment's dashboard. The Environment Health status turns gray while the application version is being updated. When deployment is complete, Elastic Beanstalk executes an application health check. The status reverts to green when the application responds to the health check. The environment dashboard shows the new running version as the new version label. Your new application version is added to the table of application versions.

8. To view the table, select **Application Versions**.

You have updated an application version on AWS Elastic Beanstalk.

---

# Review Questions

1. Which of the following AWS services enables you to automate your build, test, deploy, and release process every time there is a code change?
  - A. AWS CodeCommit
  - B. AWS CodeDeploy
  - C. AWS CodeBuild
  - D. AWS CodePipeline
2. Which of the following resources can AWS Elastic Beanstalk use to create a web server environment? (Select FOUR.)
  - A. Amazon Cognito User Pool
  - B. AWS Serverless Application Model (AWS SAM) Local
  - C. Auto Scaling group
  - D. Amazon Elastic Compute Cloud (Amazon EC2)
  - E. AWS Lambda
3. Which of the following languages is not supported by AWS Elastic Beanstalk?
  - A. Java
  - B. Node.js
  - C. Objective C
  - D. Go
4. What does the AWS Elastic Beanstalk service do?
  - A. Deploys applications and architecture
  - B. Stores static content
  - C. Directs user traffic to Amazon Elastic Compute Cloud (Amazon EC2) instances
  - D. Works with dynamic cloud changes as an IP address
5. Which operating systems does AWS Elastic Beanstalk support? (Select TWO.)
  - A. Amazon Linux
  - B. Ubuntu
  - C. Windows Server
  - D. Fedora
  - E. Jetty

6. Which of the following components can AWS Elastic Beanstalk deploy? (Select TWO.)
  - A. Amazon Elastic Compute Cloud (Amazon EC2) instances with write capabilities to an Amazon DynamoDB table
  - B. A worker application using Amazon Simple Queue Service (Amazon SQS)
  - C. An Amazon Elastic Container Service (Amazon ECS) cluster supporting multiple containers
  - D. A mixed fleet of Spot and Reserved Instances with four applications running in each environment
  - E. A mixed fleet of Reserved Instances scheduled between 9 a.m. to 5 p.m. and On-Demand Instances used for processing data workloads when needed randomly
7. Which of the following operations can AWS Elastic Beanstalk do? (Select TWO.)
  - A. Access an Amazon Simple Storage Service (Amazon S3) bucket
  - B. Connect to an Amazon Relational Database Service (Amazon RDS) database
  - C. Install agents for Amazon GuardDuty service
  - D. Create and manage Amazon WorkSpaces
8. Which service can be used to restrict access to AWS Elastic Beanstalk resources?
  - A. AWS Config
  - B. Amazon Relational Database Service (Amazon RDS)
  - C. AWS Identity and Access Management (IAM)
  - D. Amazon Simple Storage Service (Amazon S3)
9. Which AWS Identity and Access Management (IAM) entities are used when creating an environment? (Select TWO.)
  - A. Federated role
  - B. Service role
  - C. Instance profile
  - D. Profile role
  - E. User name and access keys
10. Which of the following describes how customers are charged for AWS Elastic Beanstalk?
  - A. A monthly fee based on an hourly rate for use.
  - B. A one-time upfront cost for each environment running.
  - C. No additional charges.
  - D. A fee is charged only when scaling to support traffic changes.

11. Which account is billed for user-accessed AWS resources allocated by AWS Elastic Beanstalk?
  - A. The account running the services
  - B. The cross-account able to access the shared services
  - C. The cross-account with the Amazon Simple Storage Service (Amazon S3) bucket holding a downloaded copy of the code artifact
  - D. All accounts involved
12. What can you *not* do to an Amazon Relational Database Service (Amazon RDS) instance with AWS Elastic Beanstalk?
  - A. Create a database connection.
  - B. Create a supported Oracle edition.
  - C. Retain a database instance despite the deletion of the environment's database.
  - D. Create a snapshot of the existing database (before deletion).



# Chapter

# 7



AWS® Certified Developer Official Study Guide  
By Nick Alteen, Jennifer Fisher, Casey Gerena, Wes Gruver, Asim Jalil,  
Heiward Osman, Marife Pagan, Santosh Patlolla and Michael Roth  
Copyright © 2019 by Amazon Web Services, Inc.

# Deployment as Code

---

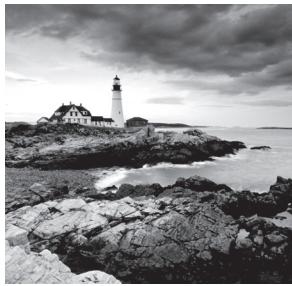
**THE AWS CERTIFIED DEVELOPER –  
ASSOCIATE EXAM TOPICS COVERED IN  
THIS CHAPTER MAY INCLUDE, BUT ARE  
NOT LIMITED TO, THE FOLLOWING:**

## Domain 1: Deployment

- ✓ 1.1 Deploy written code in AWS using existing CI/CD pipelines, processes, and patterns.
- ✓ 1.3 Prepare the application deployment package to be deployed to AWS.
- ✓ 1.4 Deploy serverless applications.

## Domain 3: Development with AWS services

- ✓ 3.4 Write code that interacts with AWS services by using Application Programming Interfaces (APIs), Software Development Kits (SDKs), and AWS Command Line Interface (CLI).



# Introduction to AWS Code Services

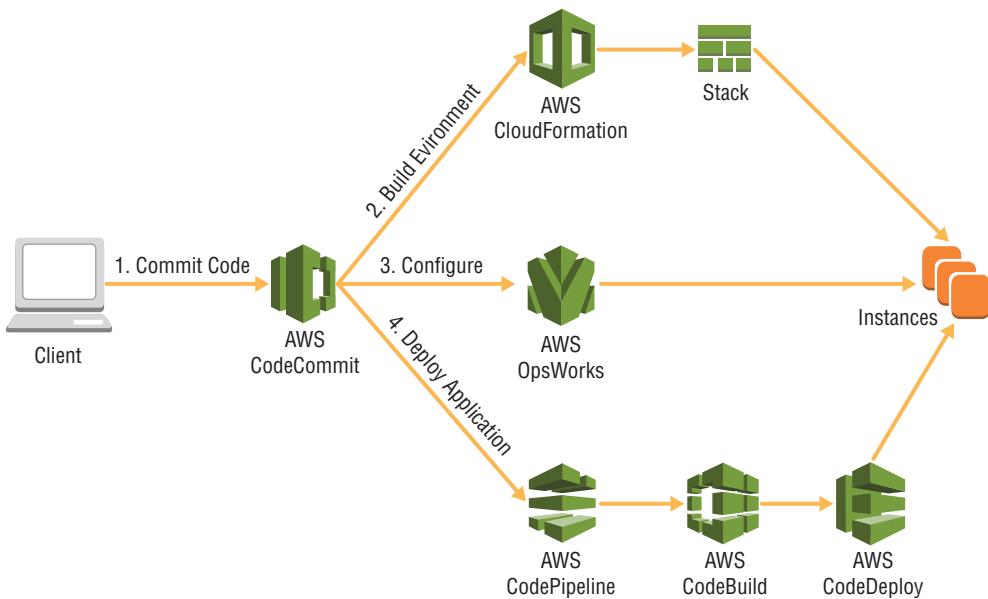
In the previous chapter, you learned about deploying code packages to AWS Elastic Beanstalk. This is a great way to migrate existing applications to highly available, fault-tolerant infrastructure. As your experience with Amazon Web Services (AWS) deployment increases over time, you may find a need to customize your deployment workflow further than what is supported within a single service. AWS provides a number of deployment services designed for flexibility, empowering customers with complex infrastructure and application deployment requirements.

This chapter introduces the AWS “Code” services. These services are responsible for creating the foundation of a repeatable application, infrastructure, and configuration deployment process. As each service is explained, you will see how they fit into an “enterprise as code” philosophy. You use this approach with each aspect of an enterprise to deploy, configure, and maintain over time via versioned code. (This includes the process to deploy code.) The primary components of an enterprise as code are application, infrastructure, and configuration, though you can take advantage of many more, such as monitoring, compliance, and audit practices.

## Continuous Delivery with AWS CodePipeline

The AWS “Code” services lay the foundation to deploy different parts of an enterprise starting from a source repository. You start with *AWS CodePipeline* to create a *continuous integration/continuous deployment pipeline* (CI/CD) that integrates various sources, tests, deployments, or other components. AWS CodePipeline implements *AWS CodeCommit* as a source in that it acts as the initialization point of your deployment process. *AWS CodeBuild* allows you to pull code and packages from various sources to create publishable build artifacts. Lastly, *AWS CodeDeploy* allows you to deploy compiled artifacts to infrastructure in your environment. AWS CodePipeline is not limited to deploying application code; it can also be used to provision, configure, and manage infrastructure.

In a fully realized enterprise as code, a single commit to a source repository can kick off processes, such as those shown in Figure 7.1.

**FIGURE 7.1** Branch view

## Benefits of Continuous Delivery

Organizations can realize a number of benefits from automating the process of testing and preparing software changes. First, there is reduced manual effort required to ensure code changes are tested prior to release. By automating tests, they are consistently run against every change made to a code repository.

Second, developers are no longer tasked with completing steps other than checking in code changes. After the change has been pushed to a source repository, initiation of the build/test process automatically begins. This allows the developers to focus on what they do best: develop software.

Third, the fact that changes are tested immediately after check-in ensures that more bugs are caught earlier in the development process. If bugs are not caught soon, the effort and cost to remediate the errors increases the further they make it in the release process.

Lastly, continuous delivery ensures that quality changes are delivered faster. This increases quality with decreased time to market. So, before you start considering storage options, take time to evaluate your data and decide which of these dimensions your data falls under. This will help you decide what type of storage is best for your data.

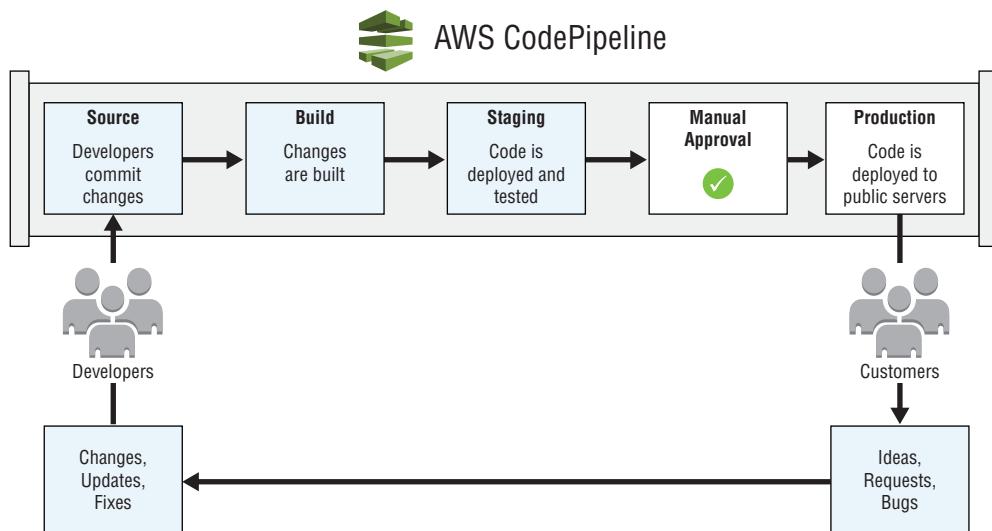
# Using AWS CodePipeline to Automate Deployments

AWS *CodePipeline* is a continuous integration and continuous delivery service for fast and reliable application and infrastructure updates. AWS CodePipeline builds, tests, and deploys your code every time there is a code change, based on the release process models you define. This enables you to deliver features and updates rapidly and reliably. You can easily build an end-to-end solution with prebuilt plugins for popular third-party services like GitHub, or you can integrate your own custom plugins into any stage of your release process. With AWS CodePipeline, you pay only for what you use. There are no up-front fees or long-term commitments.

## What Is AWS CodePipeline?

AWS CodePipeline is the underpinning of CI/CD processes in AWS. Because you define your delivery workflow as a set of stages and actions, multiple changes can be run simultaneously through the same set of processing steps every time. In Figure 7.2, the developer team is responsible for committing changes to a source repository. AWS CodePipeline automatically detects and moves into the source stage. The code change (revision) passes to the build stage, where changes are built into a package or product ready for deployment. A staging deployment is done where users can manually review the functionality that the changes introduce or modify. Before final production release, an authorized user provides a manual approval. After production release, further code changes can reliably pass through the same pipeline.

**FIGURE 7.2** AWS CodePipeline workflow



AWS CodePipeline provides a number of built-in integrations to other AWS services, such as AWS CloudFormation, AWS CodeBuild, AWS CodeCommit, AWS CodeDeploy, Amazon Elastic Container Service (ECS), Elastic Beanstalk, AWS Lambda, AWS OpsWorks Stacks, and Amazon Simple Storage Service (Amazon S3). Some partner tools include GitHub (<https://github.com>) and Jenkins (<https://jenkins.io>). Customers also have the ability to create their own integrations, which provides a great degree of flexibility.

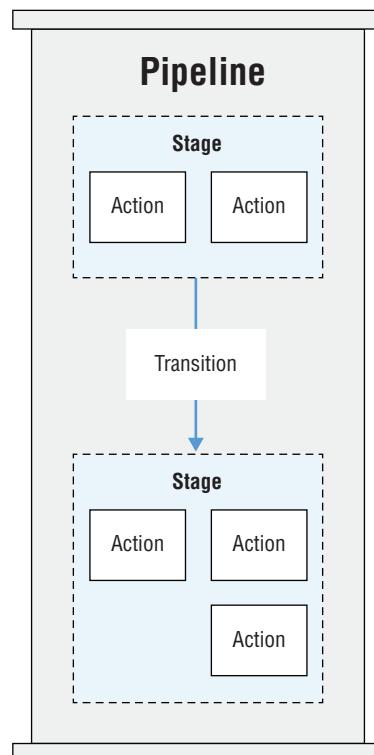
You define workflow steps through a visual editor within the AWS Management Console or via a JavaScript Object Notation (JSON) structure for use in the AWS CLI or AWS SDKs. Access to create and manage release workflows is controlled by AWS Identity and Access Management (IAM). You can grant users fine-grained permissions, controlling what actions they can perform and on which workflows.

AWS CodePipeline provides a dashboard where you can review real-time progress of revisions, attempt to retry failed actions, and review version information about revisions that pass through the pipeline.

## AWS CodePipeline Concepts

There are a number of different components that make up AWS CodePipeline and the workflows (*pipelines*) created by customers. Figure 7.3 displays the AWS CodePipeline concepts.

**FIGURE 7.3** Pipeline structure



## Pipeline

A *pipeline* is the overall workflow that defines what transformations software changes will undergo.



You cannot change the name of a pipeline. If you would like to change the name, you must create a new pipeline.

## Revision

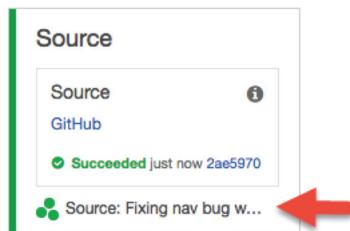
A *revision* is the work item that passes through a pipeline. It can be a change to your source code or data stored in AWS CodeCommit or GitHub or a change to the version of an archive in Amazon S3. A pipeline can have multiple revisions flowing through it at the same time, but a single stage can process one revision at a time. A revision is immediately picked up by a source action when a change is detected in the source itself (such as a commit to an AWS CodeCommit repository).



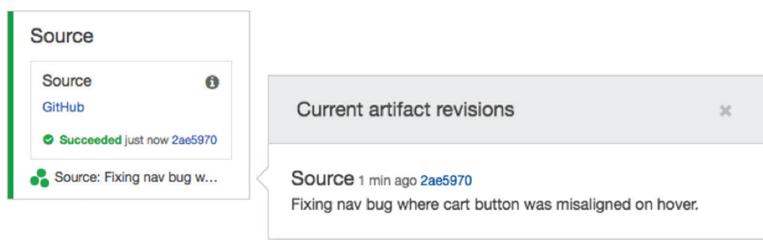
If you use Amazon S3 as a source action, you must enable versioning on the bucket.

Details of the most recent revision to pass through a stage are kept within the stage itself and are accessible from the console or AWS CLI. To see the last revision that was passed through a source stage, for example, you can select the revision details at the bottom of the stage, as shown in Figure 7.4.

**FIGURE 7.4** Source stage



Depending on the source type (Amazon S3, AWS CodeCommit, or GitHub), additional information will be accessible from the revision details pane (such as a link to the commit on <https://github.com>), as shown in Figure 7.5.

**FIGURE 7.5** Revision details

## Stage

A *stage* is a group of one or more actions. Each stage must have a unique name. Should any one action in a stage fail, the entire stage fails for this revision.

## Action

An *action* defines the work to perform on the revision. You can configure pipeline actions to run in series or in parallel. If all actions in a stage complete successfully for a revision, it passes to the next stage in the pipeline. However, if one action fails in the stage, the revision will not pass further through the pipeline. At this point, the stage that contains the failed action can be retried for the same revision. Otherwise, a new revision is able to pass through the stage.



**NOTE** A pipeline must have two or more stages. The first stage includes one or more source actions only. Only the first stage may include source actions.



**WARNING** Every action in the same stage must have a unique name.

## Source

The *source* action defines the location where you store and update source files. Modifications to files in a source repository or archive trigger deployments to a pipeline. AWS CodePipeline supports these sources for your pipeline:

- Amazon S3
- AWS CodeCommit
- GitHub



*A single pipeline can contain multiple source actions.* If a change is detected in one of the sources, all source actions will be invoked.

To use GitHub as a source provider for AWS CodePipeline, you must authenticate to GitHub when you create a pipeline. You provide GitHub credentials to authorize AWS CodePipeline to connect to GitHub to list and view repositories accessible by the authenticating account. For this link, AWS recommends that you create a service account user so that the lifecycle of personal accounts is not tied to the link between AWS CodePipeline and GitHub.

After you authenticate GitHub, a link is created between AWS CodePipeline for this AWS region and GitHub. This allows IAM users to list repositories and branches accessible by the authenticated GitHub user.

## Build

You use a *build* action to define tasks such as compiling source code, running unit tests, and performing other tasks that produce output artifacts for later use in your pipeline. For example, you can use a build stage to import large assets that are not part of a source bundle into the artifact to deploy it to Amazon Elastic Compute Cloud (Amazon EC2) instances. AWS CodePipeline supports the integrations for the following build actions:

- AWS CodeBuild
- CloudBees
- Jenkins
- Solano CI
- TeamCity

## Test

You can use *test* actions to run various tests against source and compiled code, such as lint or syntax tests on source code, and unit tests on compiled, running applications. AWS CodePipeline supports the following test integrations:

- AWS CodeBuild
- BlazeMeter
- Ghost Inspector
- Hewlett Packard Enterprise (HPE) StormRunner Load
- Nouvola
- Runscope

## Deploy

The *deploy* action is responsible for taking compiled or prepared assets and installing them on instances, on-premises servers, serverless functions, or deploying and updating infrastructure using AWS CloudFormation templates. The following services are supported as deploy actions:

- AWS CloudFormation
- AWS CodeDeploy
- Amazon Elastic Container Service
- AWS Elastic Beanstalk
- OpsWorks Stacks
- Xebia Labs

## Approval

An *approval* action is a manual gate that controls whether a revision can proceed to the next stage in a pipeline. Further progress by a revision is halted until a manual approval by an IAM user or IAM role occurs.



Specifically, the `codepipeline:PutApprovalResult` action must be included in the IAM policy.

Upon approval, AWS CodePipeline approves the revision to proceed to the next stage in the pipeline. However, if the revision is not approved (rejected or the approval expires), the change halts and will stop progress through the pipeline. The purpose of this action is to allow manual review of the code or other quality assurance tasks prior to moving further down the pipeline.



Approval actions cannot occur within source stages.

You must approve actions manually within seven days; otherwise, AWS CodePipeline rejects the code. When an approval action rejects, the outcome is equivalent to when the stage fails. You can retire the action, which initiates the approval process again. Approval actions provide several options that you can use to provide additional information about what you choose to approve.

**Publish approval notifications** Amazon Simple Notification Service (Amazon SNS) sends notices to one or more targets that approval is pending.

**Specify a Universal Resource Locator (URL) for review** You can include a URL in the approval action notification, for example, to review a website published to a fleet of test instances.

**Enter comments for approvers** You can add additional comments in the notifications for the reviewer's reference.

## Invoke

You can customize the *invoke* action within AWS CodePipeline if you leverage the power and flexibility of AWS Lambda. Invoke actions execute AWS Lambda functions, which allows arbitrary code to be run as part of the pipeline execution. Uses for custom actions in your pipeline can include the following:

- Backing up data volumes, Amazon S3 buckets, or databases
- Interacting with third-party products, such as posting messages to Slack channels
- Running through test interactions with deployed web applications, such as executing a test transaction on a shopping site
- Updating IAM Roles to allow permissions to newly created resources



When you deploy changes to multiple AWS Elastic Beanstalk environments, for example, you can use AWS Lambda to invoke a stage to swap the environment CNAMEs (`SwapEnvironmentCNAMEs`). This effectively implements blue/green deployments via AWS CodePipeline.

## Artifact

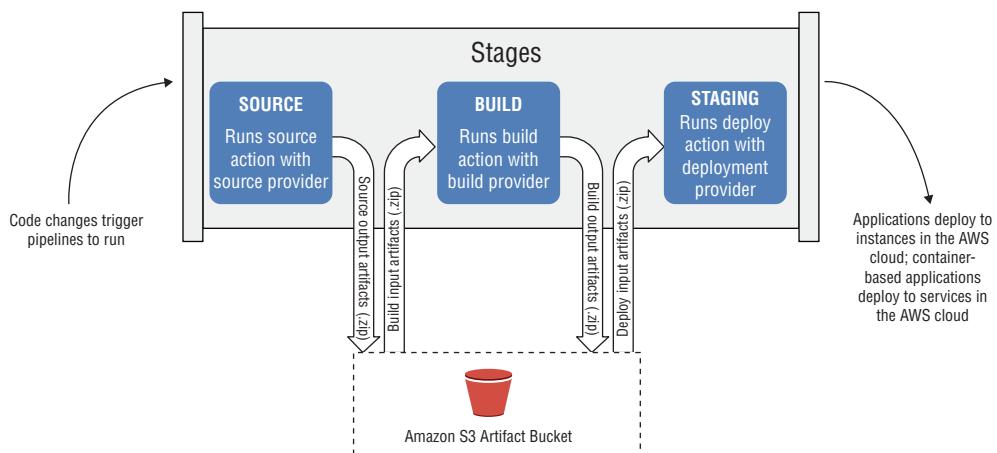
*Artifacts* are actions that act on a file or set of files. Artifacts can pass between actions and stages in a pipeline to provide a final result or version of the files. For example, an artifact that passes from a build action would deploy to Amazon EC2 during a deploy action.



Multiple actions in a single pipeline cannot output artifacts with the same name.

Every stage makes use of the Amazon S3 artifact bucket that you define when you create the pipeline. Depending on the type of action(s) in the stage, AWS CodePipeline will package the output artifact. For example, the output artifact of a source action would be an archive (.zip) containing the repository contents, which would then act as the input artifact to a build action.

For an artifact to transition between stages successfully, you must provide unique input and output artifact names. In Figure 7.6, the output artifact name for the source action must match the input artifact for the corresponding build action.

**FIGURE 7.6** Artifact transition

## Transition

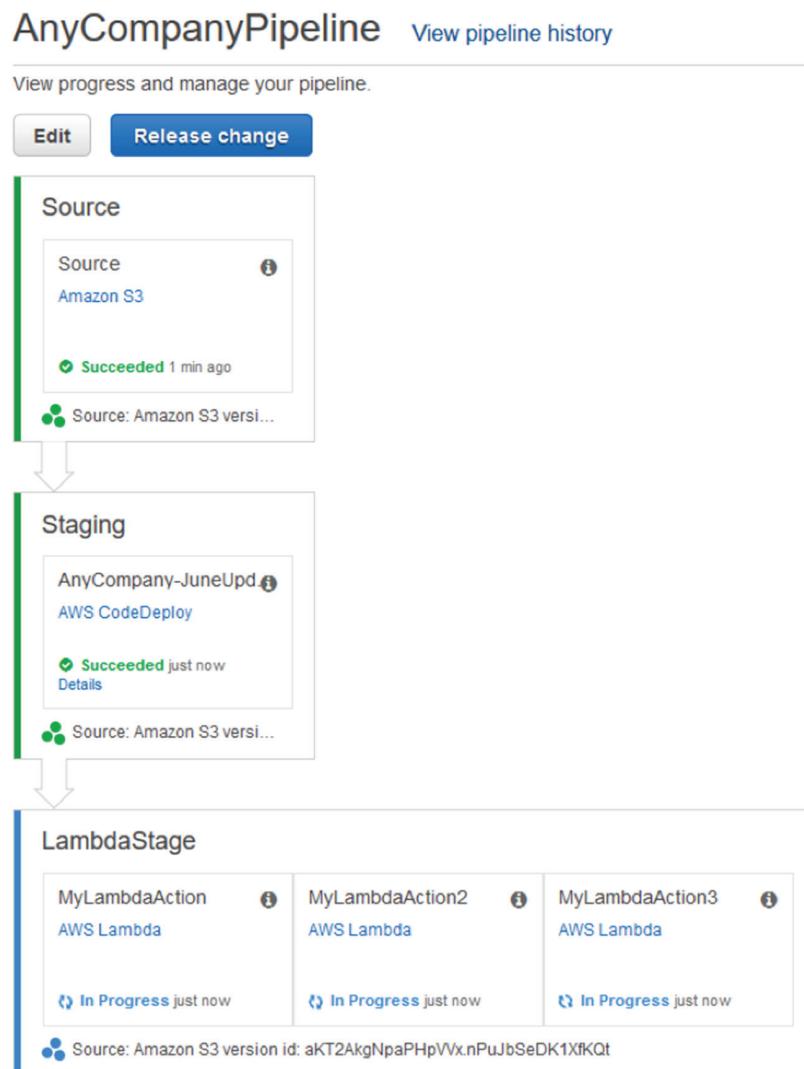
*Transitions* connect stages in a pipeline and define which stages should transition to one another. When all actions in a stage complete successfully, the revision passes to the next stage(s) in the pipeline.

You can manually disable transitions, which stops all revisions in the pipeline once they complete the preceding stage (successfully or unsuccessfully). Once you enable the transition again, the most recent successful revision resumes. Other previous successful revisions will not resume through the pipeline at this time. This concept also applies to stages that are not yet available by the time the next revision completes. If more than one revision completes while the next stage is unavailable, they will be batched. This means that the most current revision will continue through the pipeline once the next stage becomes available.

## Managing Approval Actions

*Approval* actions halt further progress through a pipeline until an authorized IAM user or IAM rule approves the transition. You can use approvals to review changes manually before final release into production, or as a code review step.

Figure 7.7 shows a pipeline with three stages: Source, Staging, and LambdaStage. The Source stage contains a source action referencing an Amazon S3 bucket. The source action has already completed and passed the source artifact to Staging. In Staging, the deploy action deploys the source artifact to Amazon EC2 with AWS CodeDeploy. If this action completes successfully, the LambdaStage stage begins, which also deploys to Amazon EC2 via AWS CodeDeploy.

**FIGURE 7.7** Full pipeline

## AWS CodePipeline Service Limits

Table 7.1 lists the AWS CodePipeline service limits.

**TABLE 7.1** AWS CodePipeline Service Limits

Limit	Value
Pipelines per region	US East (N. Virginia) (us-east-1): 40 US West (Oregon) (us-west-2): 60 EU (Ireland) (eu-west-1): 60 Other supported regions: 20
Stages per pipeline	Minimum: 2 Maximum: 10
Actions per stage	Minimum: 1 Maximum: 20
Parallel actions per stage	Maximum: 10
Sequential actions per stage	Maximum: 10
Maximum artifact size	Amazon S3 source: 2 GB AWS CodeCommit source: 1 GB GitHub source: 1 GB



When you deploy to AWS CloudFormation, the maximum artifact size is 256 MB.

## AWS CodePipeline Tasks

The remainder of this section will focus on the tasks you need to build and execute a simple pipeline and how to outline the requirements to build cross-account pipelines. This concept is particularly important for organizations that have multiple AWS accounts, especially when you separate environments across accounts (such as Account A for development, Account B for Quality Assurance [QA], and Account C for production), as AWS CodePipeline will need access to resources in each account to automate deployments successfully.



Before you start the next steps, make sure that you have an IAM user with an access key and secret access key and that the user has sufficient AWS CodePipeline permissions.

## Create an AWS CodePipeline

It is best to name your pipeline something meaningful, such as Dev\_S3\_Bucket. After you select a source provider (Amazon S3, AWS CodeCommit, or GitHub), you must enter a full object path. This corresponds to the .zip archive that will be tracked for changes.

When you select Amazon S3, AWS CodePipeline creates an Amazon CloudWatch Events rule, IAM role, and AWS CloudTrail trail. These are the default methods that notify AWS CodePipeline of changes to the source archive. You can also use AWS CodePipeline to check regularly for changes. This, however, will provide a slower update experience.

You select AWS CodeBuild, Jenkins, or Solano CI for a build provider.



The Jenkins build provider requires you to install the AWS CodePipeline plugin for Jenkins on the server.

The Solano CI build provider requires authentication to GitHub with a valid user. After authenticating to GitHub, you must authenticate to Solano CI.

If you do not select a build provider, you must select a deployment provider (if you select a build provider, the deployment step is optional). This option is useful if you desire the pipeline execution to be a finished build artifact, such as the case with custom media transcoding with AWS CodeBuild. The available providers for the deployment stage are Amazon ECS, AWS CloudFormation, AWS CodeDeploy, AWS Elastic Beanstalk, and AWS OpsWorks Stacks.

AWS Elastic Beanstalk allows customers to automate deployment of application archives to one or more Amazon EC2 instances. It also handles health checks, load balancing, log gathering, and other important tasks automatically. Since it requires a bundled application archive to upload to instances for deployment, it is a natural fit for AWS CodePipeline, which provides artifacts as archives. To deploy to AWS Elastic Beanstalk from AWS CodePipeline, simply provide the application and environment name.



For deployment to AWS Elastic Beanstalk, the maximum application archive size is 512 MB. The deployment artifact must not exceed this size, or the deployment will fail.

You select a service role for AWS CodePipeline to access AWS resources within your account. You can select an existing IAM role or create a new role.



You can only select IAM roles with a trust policy that allows AWS CodePipeline to assume them.

## Start a Pipeline

After you create a pipeline, the first stage updates the source repository or archive, and then the pipeline will automatically begin execution. To rerun the pipeline for the most recent

revision, select Release Change in the AWS CodePipeline console, or invoke the `aws codepipeline start-pipeline-execution` AWS CLI command.

```
aws codepipeline start-pipeline-execution --name SamplePipeline
```

## Retry a Failed Action

If a pipeline action fails for any reason, you can retry that action on the same revision in the console or use the `aws codepipeline retry-stage-execution` AWS CLI command. However, there are certain situations where a failed action may become ineligible for retries.

- The pipeline itself has changed after the action failed.
- Other actions in the same stage have not completed.
- The retry attempt is already in progress.

## Create a Cross-Account Pipeline

In some architectures, environments may be spread across two or more AWS accounts. You can implement a single CI/CD workflow with AWS CodePipeline that interacts with resources in multiple AWS accounts.



If an organization has separate accounts for development, test, and production workloads, you can leverage one pipeline to deploy to resources in all three. To do so, you must create and shard several components between accounts.



A source action of Amazon S3 cannot reference buckets in accounts other than the pipeline account.

## Pipeline Account Steps

In the following steps, the account that contains the pipeline will be the *pipeline account*. The account to deploy resources will be the *target account*.

1. Create an AWS Key Management Service (AWS KMS) key in the pipeline account, and apply it to the pipeline. This key encrypts artifacts that pass between stages, and you configure it to allow access to the target account in a later step. After you create the AWS KMS key, you apply a key policy that allows access to the key by both the AWS CodePipeline service role in the pipeline account and the Amazon Resource Name (ARN) of the target account.
2. Apply a bucket policy to the Amazon S3 bucket for the pipeline. This policy must grant access to the bucket by the target account.
3. Create a policy that allows the pipeline account to assume a role in the target account. You attach this policy to the AWS CodePipeline service role.

## Target Account Steps



If you deploy revisions to Amazon EC2 instances (as with AWS CodeDeploy), you apply a policy to the instance role that allows access to the Amazon S3 bucket that the AWS CodePipeline uses in the pipeline account. Additionally, the instance role must also have a policy that allows access to the AWS KMS key.

1. Create an IAM role that contains a trust relationship policy that allows the pipeline account to assume the role.
2. Create an IAM policy that allows access to deploy to the pipeline's resources. Attach this policy to the IAM role.
3. Create an IAM policy that allows access to the Amazon S3 bucket in the pipeline account, and attach it to the IAM role. After completing the previous steps, revisions that pass through the pipeline account will be accessible by the target account.

# Using AWS CodeCommit as a Source Repository

AWS *CodeCommit* is a fully managed source control service that makes it easy for companies to host secure and highly scalable private Git repositories. AWS CodeCommit eliminates the need to operate your own source control system or worry about scaling its infrastructure. You can use AWS CodeCommit to store anything securely, from source code to binaries, and it works seamlessly with your existing Git tools.

## What Is AWS CodeCommit?

Before any activities can occur to deploy applications, you must first have a location where you can store and version application code in a reliable fashion. AWS CodeCommit is a cloud-based, highly available, and redundant version control service. AWS CodeCommit leverages the Git framework, and it is fully compatible with existing tooling. There are a number of benefits to this service, such as the following:

- Automatic encryption in-transit and at rest.
- Scaling to handle rapid release cycles and large repositories.
- Access control to the repository using IAM users, IAM roles, and IAM policies.
- Hypertext Transfer Protocol Secure (HTTPS) and Secure Shell (SSH) connectivity.

However, the biggest benefit of AWS CodeCommit is built-in integration with multiple other AWS services, like AWS CodePipeline. With these integrations, AWS CodeCommit acts as the initial step to automate application code releases.

## AWS CodeCommit Concepts

This section details the concepts behind AWS CodeCommit.

### Credentials

When you interact with AWS, you specify your AWS security credentials to verify who you are and whether you have permission to access the resources that you request. AWS uses the security credentials to authenticate and authorize your requests.

### HTTPS

HTTPS connectivity to a Git-based repository requires a username and password, which pass to the repository as part of a request. To use AWS CodeCommit with HTTPS credentials, you must first add them to an IAM user with sufficient permissions to interact with the repository. To create Git credentials for your IAM user, you open the IAM console, and select the user who will need to authenticate to the AWS CodeCommit repository via HTTPS.

AWS generates security credentials for the usernames and passwords, and they cannot be set to custom values.



Make sure to download or copy the credentials because the password will be lost after you close the success window.

After you configure your Git CLI/application to use the repository's HTTPS endpoint and the username/password, you will have access to the AWS CodeCommit repository.

### SSH

With SSH authentication, there is no need to install the AWS CLI to connect to your repository. However, you perform some additional configuration tasks.

- Your IAM user must have the ability to manage their own SSH keys. To accomplish this, you add the `IAMUserSSHKeys` managed policy to the account.
- Scaling to handle rapid release cycles and large repositories.
- For Windows users, install a bash emulator, such as Git Bash.

To configure SSH authentication to AWS CodeCommit repositories, follow these steps:

1. In the IAM console, select the user account you want to modify.
2. Upload the public SSH key on the Security Credentials tab.
3. Copy the SSH key identity (ID). This follows the form `APKAEIGHANK3EXAMPLE`.

4. Update the `~/.ssh/config` file on your workstation to include these contents:

```
Host git-codecommit.*.amazonaws.com
User YOUR_SSH_KEY_ID
IdentityFile YOUR_PRIVATE_KEY_FILE
```

5. To verify the configuration, test a simple SSH connection to the AWS CodeCommit endpoint, as shown here:

```
Format: ssh git-codecommit.[REGION_CODE].amazonaws.com
ssh git-codecommit.us-east-1.amazonaws.com
```

### Use the Credential Helper

The previous HTTPS and SSH authentication methods both rely on additional credentials aside from IAM access/secret keys. It is also possible to authenticate to AWS CodeCommit with IAM credentials and the *AWS CodeCommit credential helper*. The credential helper translates IAM credentials to those that AWS CodeCommit can use to perform Git actions, such as to clone a repository or merge a pull request. To configure the credential helper on your workstation, do the following:

1. Install and configure the AWS CLI.
2. Install Git.
3. Configure Git to leverage the credential helper from the AWS CLI with these commands:

```
git config --global \
credential.helper '!aws codecommit credential-helper $@'
git config --global credential.UseHttpPath true
```

Once complete, HTTPS interactions with the AWS CodeCommit repository should work as expected.



The credential helper authentication method is the only one available for root and federated IAM Users.

## Development Tools and Integrated Development Environment

AWS CodeCommit integrates automatically with any development tools that support IAM credentials. Additionally, after you set up HTTPS Git credentials, you are able to use any tools that support this authentication mechanism instead. Examples of supported integrated development environment (IDE) include the following:

- AWS Cloud9
- Eclipse
- IntelliJ
- Visual Studio

## Repository

A *repository* (repo) is the foundation of AWS CodeCommit. This is the location where you store source code files, track revisions, and merge contributions (commits). When you create a repository, it will contain an empty master branch by default. To configure additional branches and commit code changes, you connect the repository to a local workstation where changes can be made before you upload or push them.



Repository names must be unique within an individual AWS account; however, you can change them without re-creating the repository. When you change a repository name, you need to update any local copies of the repository to have their remote point to the new HTTPS or SSH URL with the `git remote add` command.

## Repository Notifications

AWS CodeCommit supports triggers via Amazon SNS, which you can use to leverage other AWS services for post-commit actions, such as firing a webhook with AWS Lambda after a commit is pushed to a development branch. To implement this, AWS CodeCommit uses AWS CloudWatch Events. You create event rules that trigger for each of the event types that you select in AWS CodeCommit. Event types that will fire notifications include the following:

- Pull Request Update Events
- Create a Pull Request
- Close a Pull Request
- Update Code in a Pull Request
- Title or Description Changes
- Pull Request Comment Events
- Commit Comment Events
- Comments on Code Changes
- Comments on Files in a Commit
- Comments on the Commit Itself



If you change the name of the repository through the AWS CLI or SDK, the notifications cease to function. (This behavior is not present when you change names in the AWS Management Console.) To restore lost notifications, delete the settings and configure them a second time.

## Repository Triggers

Repository triggers are not the same as notifications, as the events that fire each differ greatly. Use *repository triggers* to send notifications to Amazon SNS or AWS Lambda during these events:

- Push to Existing Branch
- Create a Branch or Tag
- Delete a Branch or Tag

Triggers are similar in functionality to webhooks used by other Git providers, like GitHub. You can use triggers to perform automated tasks such as to start external builds, to notify administrators of code pushes, or to perform unit tests. There are some restrictions on how to configure triggers.

- The trigger destination, Amazon SNS or AWS Lambda, must exist in the same AWS region as the repository.
- If the destination is Amazon SNS in another AWS account, the Amazon SNS topic must have a policy that allows notifications from the account that contains the repository.

## Cross-Account Access to a Different Account

In some situations, the repository that contains the application source code may be located in a separate AWS account from the IAM user/role attempting to access it. In these situations, there are several steps that you must perform in the *repository account* and the *user account*.

### Repository Account Actions

1. Create a policy for access to the repository. This policy should allow users in the user account to access one or more specific repositories, as well as (optionally) to view a list of all repositories.
2. Attach this policy to a role in the same account, and allow users in the user account to assume this role.

### User Account Actions

1. Create an IAM user or IAM group. This user or group will be able to access the repository after the next step.
2. Assign a policy to the user or group that allows them to assume the role created in the repository account as part of the previous steps.

Once these steps are complete, the IAM user will first need to assume the cross-account role before you attempt to clone or otherwise access the repository. You adjust the AWS

credentials file `~/.aws/config` (Linux/macOS) or `drive:\Users\username\.aws\config` (Windows). A profile will be added to this config file that specifies the cross-account role to assume.

```
[profile MyCrossAccountProfile]
region = US East (Ohio)
role_arn=arn:aws:iam:111122223333:role/MyCrossAccountContributorRole
source_profile=default
```

Lastly, you need to modify the AWS CLI credential helper so that you use `MyCrossAccountProfile`.

```
git config --global credential.helper \
'!aws codecommit credential-helper --profile MyCrossAccountProfile $@'
```

From this point, the IAM user in the user account will be able to clone and interact with the repository in the repository account.

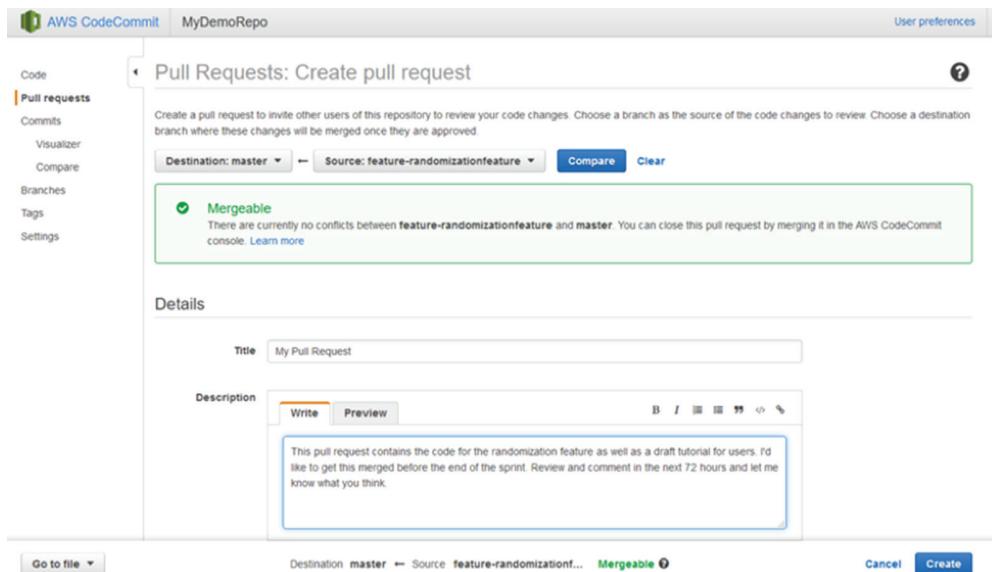
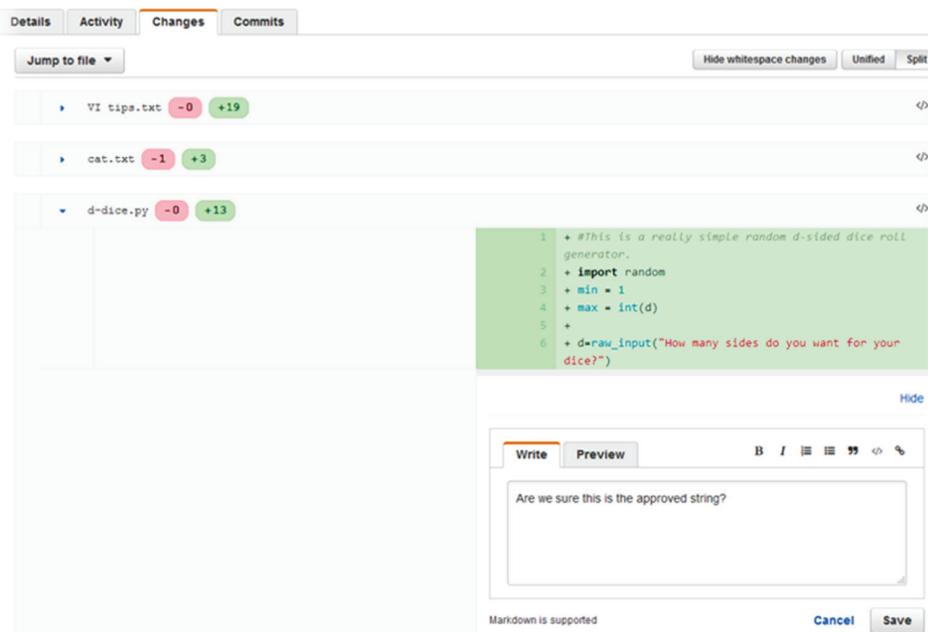
## Files

A *file* is a piece of data that is subject to version control by AWS CodeCommit. AWS CodeCommit tracks any modifications made to this file on a per-line level. You use the Git client to push changes in a file to the repository, where it tracks against other changes in previous commits.

## Pull Requests

*Pull requests* are the primary vehicle on which you review and merge code changes between branches. Unlike branch merging, pull requests allow multiple users to comment on changes before they merge with the destination branch. The typical workflow of a pull request is as follows:

1. Create a new branch off the default branch for the feature or bug fix.
2. Make changes to the branch files, commit, and push to the remote repository.
3. Create a pull request for the changes to integrate them with the default branch, as shown in Figure 7.8.
4. Other users can review the changes in the pull request and provide comments, as shown in Figure 7.9.
5. You can push any additional changes from user feedback to the same branch to include them in the pull request.
6. Once all reviewers provide approval, the pull request merges into the default branch and closes. You can close pull requests when you merge the branches locally or when you close the request via the AWS CodeCommit console or the AWS CLI.

**FIGURE 7.8** Creating a pull request**FIGURE 7.9** Reviewing changes

## Commits

*Commits* are point-in-time changes to contents of files in a repository. A commit is not a new copy of the file, but it is instead a way to track changes in the line(s) in a file, by whom, and when. When you push a commit to the repository, AWS CodeCommit tracks the following file changes:

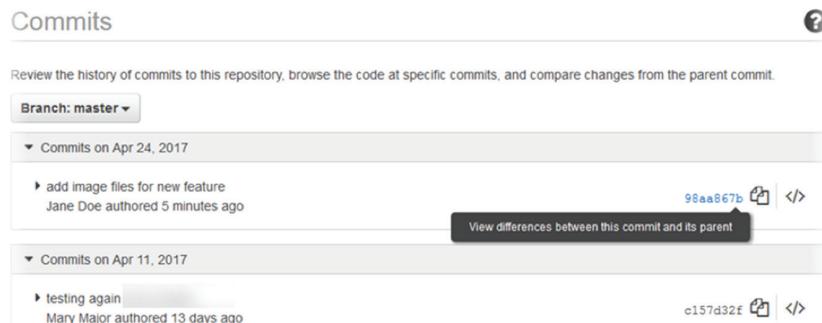
- Author Name
- Author Email
- Commit Message

Commits to a repository in AWS CodeCommit can be made in one of two ways. The most common workflow is to use the Git CLI and update the repository using `git push`. The AWS CLI supports the `aws codecommit put-file` action, which allows you to update a file on the repository with a local copy and specify a branch, parent commit, and message.

```
aws codecommit put-file --repository-name MyDemoRepo \
 --branch-name feature-branch \
 --file-content file://MyDirectory/ExampleFile.txt \
 --file-path /solutions/ExampleFile.txt \
 --parent-commit-id 11112222EXAMPLE \
 --name "Developer" \
 --email developer@myexamplesite.com \
 --commit-message "Fixed a bug"
```

The AWS CodeCommit console supports viewing differences between commits. To view differences between a commit and its parent, open the Commits pane on the repository dashboard and then select the commit ID, as shown in Figure 7.10.

**FIGURE 7.10** Selecting the commit ID



After doing so, you can view changes in this commit either side by side (*Split view*) or in the same pane (*Unified view*), as shown in Figure 7.11.

**FIGURE 7.11** Split view

The screenshot shows a commit detail page for commit 7e9fd309. The commit message is "Fix incorrect variable name". It shows a diff between two versions of a file named `ahs_count.py`. The left pane shows the initial state with a red highlight on line 6: `- ess = s.count('s')`. The right pane shows the updated state with a green highlight on line 6: `+ ess = ess.count('s')`. The bottom right of the diff area has buttons for "Hide whitespace changes", "Unified", and "Split".

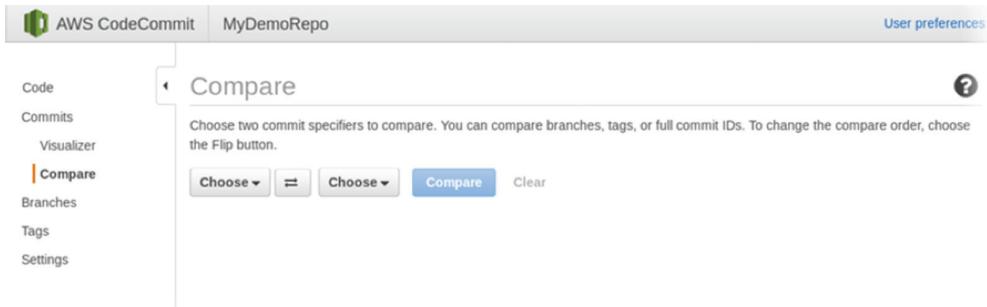
```

Commit 7e9fd309
Mary Major authored 6 minutes ago
Parent: 7aa87a03
Commit message
Fix incorrect variable name
▶ 1 changed file from 7aa87a03
ahs_count.py
@@ -3,7 +3,7 @@
 5 s = raw_input('Type a sentence or a string of
 letters: ')
 6 s = s.lower()
 7 - ess = s.count('s')
 8 z = z.count('z')
 9
10 total = (ess + z)
@@ -3,7 +3,7 @@
 5 s = raw_input('Type a sentence or a string of
 letters: ')
 6 s = s.lower()
 7 + ess = ess.count('s')
 8 z = z.count('z')
 9
10 total = (ess + z)

```

You can also view differences between arbitrary commit IDs in the same repository. In the Compare window of the repository dashboard, you can choose two commit IDs for comparison, as shown in Figure 7.12.

**FIGURE 7.12** Select and compare



After you select two commit IDs, click the Compare button. This will provide a similar split or unified view of changes.

## Branches

*Branches* are ways to separate and organize groups of commits. This allows developers to organize work in a meaningful fashion, separating changes into logical groups based on the feature or bug-fix being developed. For example, as you can see in Figure 7.13, a single repository may have branches for each environment: dev, test, and prod. Or, individual features and bug fixes can have separate branches.

**FIGURE 7.13** Branch view

Name	Last commit date	Commit message	Actions
master		add image files for new feature	
jane-branch		Added another comment	Compare Create pull request ...
new-branch		Added a basic Vi tutorial	Compare Create pull request ...
prepod	a year ago	...	Compare Create pull request ...
working-branch		Merge branch 'jane-branch' into working...	Compare Create pull request ...

A *default branch* is the base when you clone the repository. When you clone a repository to your local machine, the default branch (such as “master” or “prod”) clones. You cannot delete the default branch until a new branch is set as default, or you will delete the entire repository.



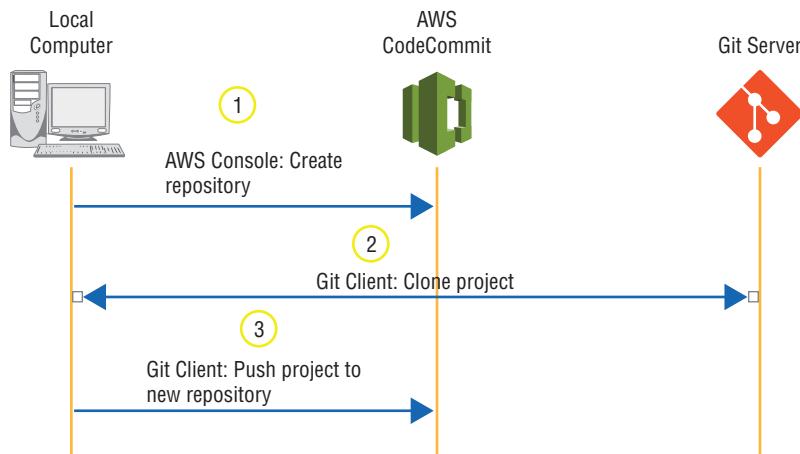
You can change the default branch for a repository, but first the new default branch must exist in the remote repository.

## Migrate to AWS CodeCommit

This section details how to migrate a Git repository, unversioned files, or another repository type, and it is important when you migrate a high volume of or large files.

### Migrate a Git Repository

You use an AWS CodeCommit to migrate from a Git repository, as shown in Figure 7.14.

**FIGURE 7.14** Migrating from a Git repository

The first step is to create the AWS CodeCommit repository (via either the AWS Management Console or the AWS CLI or AWS SDK). After you create the repository, clone the project to a local workstation. To push this repository to AWS CodeCommit, set the repository's remote to the AWS CodeCommit repository's HTTPS or SSH URL.

```
git push \
 https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyClonedRepository \
 --all
```

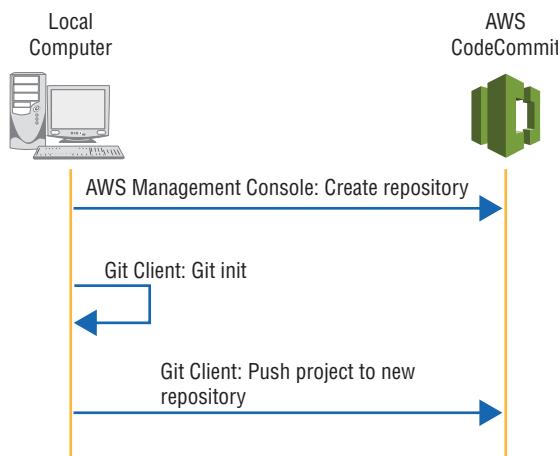
If you need to push any tags to the new repository, run the following code:

```
git push \
 https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyClonedRepository \
 --tags
```

### Migrate Unversioned Content

You can migrate any local or unversioned content to AWS CodeCommit in a similar manner, as if the content exists in another Git-based repository service. The primary difference is that you set up a new repository instead of cloning an existing one to migrate. Refer to Figure 7.15.

First create the AWS CodeCommit repository (either via the AWS Management Console or via the AWS CLI or AWS SDK). Next, create a local directory with the files to migrate, and run `git init` from the command line or terminal in that directory. This will initialize the directory to work with Git so that any file changes are tracked. After the directory initializes, run `git add .` to add all current files to Git. Run `git commit -m 'Initial Commit'` to generate a commit. Lastly, push the commit to AWS CodeCommit with `git push https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyFirstRepo --all`.

**FIGURE 7.15** Migrating unversioned content

### Migrate Incrementally

For large repositories, you can migrate in incremental steps and push many smaller files. This prevents any network issues that may cause the entire push to fail. If any smaller commit fails, it is a trivial matter to restart it when you compare it to a single, monolithic commit.

Additionally, when you push large repositories, AWS recommends that you use SSH over HTTPS, as there is a chance that the HTTPS connection may terminate because of various network or firewall issues.

## AWS CodeCommit Service Limits

AWS CodeCommit enforces the service limits in Table 7.2. An asterisk (\*) indicates limits that require you to submit a request to AWS Support to increase the limits.

**TABLE 7.2** AWS CodeCommit Service Limits

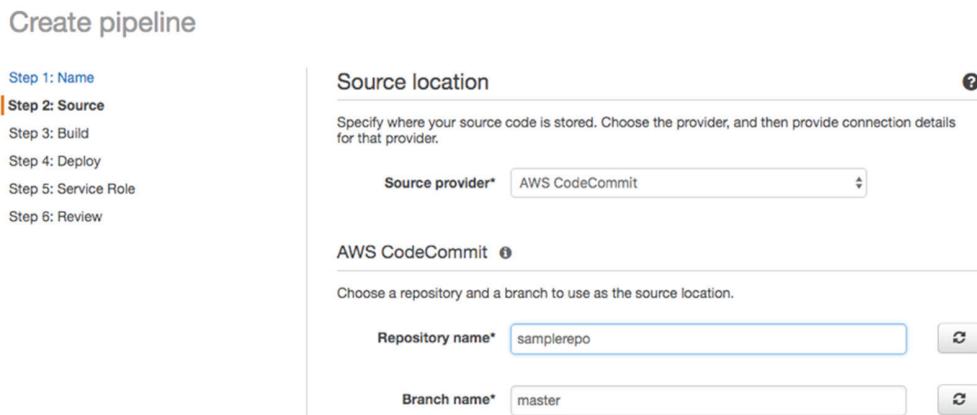
Limit	Value
Repositories per account*	1,000
References per push	4,000
Triggers per repository	10
Git blob size	2 GB

## Using AWS CodeCommit with AWS CodePipeline

You can use AWS CodeCommit as a source action in your pipeline. This allows you to utilize a highly available, redundant version control system as the initialization point of your CI/CD pipeline.

When you select AWS CodeCommit as the source provider, you must provide a repository name and branch. If you use AWS CodeCommit, it creates an Amazon CloudWatch Events rule and an IAM role to monitor the repository and branch for changes, as shown in Figure 7.16.

**FIGURE 7.16** Source location



One issue that can arise is if you store large binary files. Because of the system Git uses to track file changes, Git creates a new copy of every modification to a binary file within the repository. Over time, this can cause repositories to grow rapidly in size. Instead of storing binary files in AWS CodeCommit, add an additional Amazon S3 source action to the pipeline. If you store large binary files in Amazon S3, you can reduce the overall cost and development time because of the reduction of time it takes to push/pull commits. Since Amazon S3 already supports versioning (and requires it for use with AWS CodePipeline), changes to binary objects will still be tracked so that rollbacks are straightforward.

## Using AWS CodeBuild to Create Build Artifacts

AWS *CodeBuild* is a fully managed build service that compiles source code, runs tests, and produces software packages that are ready to deploy. With AWS CodeBuild, you do not need to provision, manage, and scale your own build servers. AWS CodeBuild scales continuously and processes multiple builds concurrently, so your builds do not wait in a queue. AWS

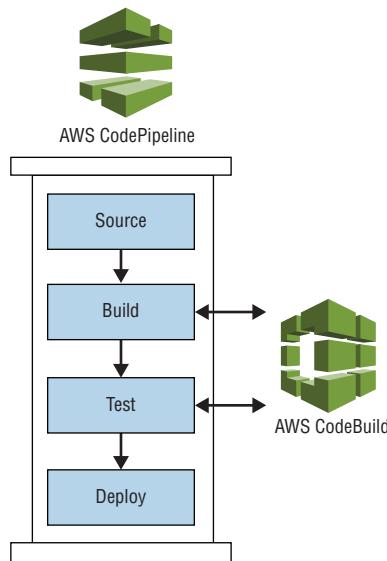
CodeBuild has prepackaged build environments, or you can create custom build environments that use your own build tools. With AWS CodeBuild, AWS charges by the minute for the compute resources you use.

## What Is AWS CodeBuild?

AWS CodeBuild enables you to define the build environment to perform build tasks and the actual tasks that it will perform. AWS CodeBuild comes with prepackaged build environments for most common workloads and build tools (Apache Maven, Gradle, and others), and it allows you to create custom environments for any custom tools or processes.

AWS CodePipeline includes built-in integration with AWS CodeBuild, which can act as a provider for any build or test actions in your pipeline, as shown in Figure 7.17.

**FIGURE 7.17** Using AWS CodeBuild in AWS CodePipeline



## AWS CodeBuild Concepts

AWS CodeBuild initiates build tasks inside a build project, which defines the environmental settings, build steps to perform, and any output artifacts. The build container's operating system, runtime, and build tools make up the build environment.

### Build Projects

*Build projects* define all aspects of a build. This includes the environment in which to perform builds, any tools to include in the environment, the actual build steps to perform, and outputs to save.

## Create a Build Project

When you create a build project, you first select a source provider. AWS CodeBuild supports AWS CodeCommit, Amazon S3, GitHub, and BitBucket as source providers. When you use GitHub or BitBucket, a separate authentication flow will be invoked. This allows access to the source repository from AWS CodeBuild. GitHub source repositories also support webhooks to trigger builds automatically any time you push a commit to a specific repository and branch.

After AWS CodeBuild successfully connects to the source repository or location, you select a *build environment*. AWS CodeBuild provides preconfigured build environments for some operating systems, runtimes, and runtime versions, such as Ubuntu with Java 9.

Next, you will configure the *build specification*. This can be done in one of two ways. You can insert build commands in the console or specify a `buildspec.yml` file in your source code. Both options are valid, but if you use a `buildspec.yml` file, you will see additional configuration options.

If your build creates artifacts you would like to use in later steps of your pipeline/process, you can specify *output artifacts* to save to Amazon S3. Otherwise, you can choose not to save any artifacts. You will need to specify individual filename(s) for AWS CodeBuild to save on your behalf.

AWS CodeBuild supports caching, which you can configure in the next step. Caching saves some components of the build environment to reduce the time to create environments when you submit build jobs.

Every build project requires an IAM service role that is accessible by AWS CodeBuild. When you create new projects, you can automatically create a service role that you restrict to this project only. You can update service roles to work with up to 10 build projects at a time.

Lastly, you can configure AWS CodeBuild to create build environments with connectivity to an Amazon Virtual Private Cloud (Amazon VPC) in your account. To do so, specify the Amazon VPC ID, subnets, and security groups to assign to the build environment. You can configure other settings when you create the build, such as to run the Docker daemon in privileged mode to build Docker images.

After you set the build project properties, you can select the compute type (memory and vCPU settings), any environment variables to pass to the build container, and tags to apply to the project.



When you set environment variables, they will be visible in plain text in the AWS CodeBuild console and AWS CLI or SDK. If there is sensitive information that you would like to pass to build jobs, consider using the *AWS Systems Manager Parameter Store*. This will require the build project's IAM role to have permissions to access the parameter store.

## Build Specification (`buildspec.yml`)

The `buildspec.yml` file can provide the build specification to your build projects in the AWS CodeBuild console, the AWS CLI, or the AWS SDK when you create the build project, or as part of your source repository in a YAML-formatted `buildspec.yml` file. You can supply only one build specification to a build project. A build specification's format is as follows:

```
version: 0.2

env:
 variables:
 key: "value"
 parameter-store:
 key: "value"

phases:
 install:
 commands:
 - command
 pre_build:
 commands:
 - command
 build:
 commands:
 - command
 post_build:
 commands:
 - command
artifacts:
 files:
 - location
discard-paths: yes
base-directory: location
cache:
 paths:
 - path
```

### Version

AWS supports multiple build specification versions; however, AWS recommends you use the latest version whenever possible.

## Environment Variables (env)

You can add optional environment variables to build jobs. Any key/value pairs that you provide in the variables section are available as environment variables in plain text.



Any environment variables that you define here will overwrite those you define elsewhere in the build project, such as those in the container itself or by Docker.

The parameter-store mapping specifies parameters to query in AWS Systems Manager Parameter Store.

## Phases

The phases mapping specifies commands to run at each stage of the build job. When you specify build settings in the AWS CodeBuild console, AWS CLI, or AWS SDK, you are not able to separate commands into phases. With a build specifications file, you can separate commands into phases.

**install** Commands to execute during installation of the build environment.

**pre\_build** Commands to be run before the build begins.

**build** Commands to be run during the build.

**post\_build** Commands to be run after the build completes.



If a command fails in any stage, subsequent stages will not run.

## Artifacts

The artifacts mapping specifies where AWS CodeBuild will place output artifacts, if any. This is required only if your build job produces actual outputs. For example, unit tests would not produce output artifacts for later use in a pipeline. The files list specifies individual files in the build environment that will act as output artifacts. You can specify individual files, directories, or recursive directories. You can use discard-paths and base-directory to specify a different directory structure to package output artifacts.

## Cache

If you configure caching for the build project, the cache map specifies which files to upload to Amazon S3 for use in subsequent builds.

## Build Project Cache

This example sets the JAVA\_HOME and LOGIN\_PASSWORD environment variables (the latter is retrieved from AWS Systems Manager Parameter Store), installs updates in the build environment, runs a Maven installation, and saves the .jar output to Amazon S3 as a build artifact. For future builds, the content of the /root/.m2 directory (and any subdirectories) is cached to Amazon S3.

```
version: 0.2

env:
 variables:
 JAVA_HOME: "/usr/lib/jvm/java-8-openjdk-amd64"
 parameter-store:
 LOGIN_PASSWORD: "dockerLoginPassword"

phases:
 install:
 commands:
 - echo Entered the install phase...
 - apt-get update -y
 - apt-get install -y maven
 pre_build:
 commands:
 - echo Entered the pre_build phase...
 - docker login -u User -p $LOGIN_PASSWORD
 build:
 commands:
 - echo Entered the build phase...
 - echo Build started on `date`
 - mvn install
 post_build:
 commands:
 - echo Entered the post_build phase...
 - echo Build completed on `date`
artifacts:
 files:
 - target/messageUtil-1.0.jar
 discard-paths: yes
cache:
 paths:
 - '/root/.m2/**/*'
```

## Build Environments

A *build environment* is a Docker image with a preconfigured operating system, programming language runtime, and any other tools that AWS CodeBuild uses to perform build tasks and communicate with the service, along with other metadata for the environment, such as the compute settings. AWS CodeBuild maintains its own repository of preconfigured build environments. If these environments do not meet your requirements, you can use public Docker Hub images. Alternatively, you can use container images in Amazon Elastic Container Registry (Amazon ECR).

### AWS CodeBuild Environments

AWS CodeBuild provides build environments for Ubuntu and Amazon Linux operating systems, and it supports the following:

- Android
- Docker
- Golang
- Java
- Node.js
- PHP
- Python
- Ruby
- .NET Core



Not all programming languages support both Ubuntu and Amazon Linux build environments.

### Compute Types

Table 7.3 lists the memory, virtual central processing unit (vCPU), and disk space configurations for build environments.

**TABLE 7.3** Compute Configurations for Build Environments

Compute Type	Memory	vCPUs	Disk Space
BUILD_GENERAL1_SMALL	3 GB	2	64 GB
BUILD_GENERAL1_MEDIUM	7 GB	4	128 GB
BUILD_GENERAL1_LARGE	15 GB	8	128 GB

## Environment Variables

AWS CodeBuild provides several environment variables by default, such as AWS\_REGION, CODEBUILD\_BUILD\_ID, and HOME.



When you create your own environment variables, AWS CodeBuild reserves the CODEBUILD\_ prefix.

## Builds

When you initiate a build, AWS CodeBuild copies the input artifact(s) into the build environment. AWS CodeBuild uses the build specification to run the build process, which includes any steps to perform and outputs to provide after the build completes. Build logs are made available to Amazon CloudWatch Logs for real-time monitoring.

When you run builds manually in the AWS CodeBuild console, AWS CLI, or AWS SDK, you have the option to change several properties before you run a build job.

- Source version (Amazon S3)
- Source branch, version, and Git clone depth (AWS CodeCommit, GitHub, and Bitbucket)
- Output artifact type, name, or location
- Build timeout
- Environment variables

## AWS CodeBuild Service Limits

AWS CodeBuild enforces service limits in Table 7.4. An asterisk (\*) indicates that you can increase limits if you submit a request to AWS Support.

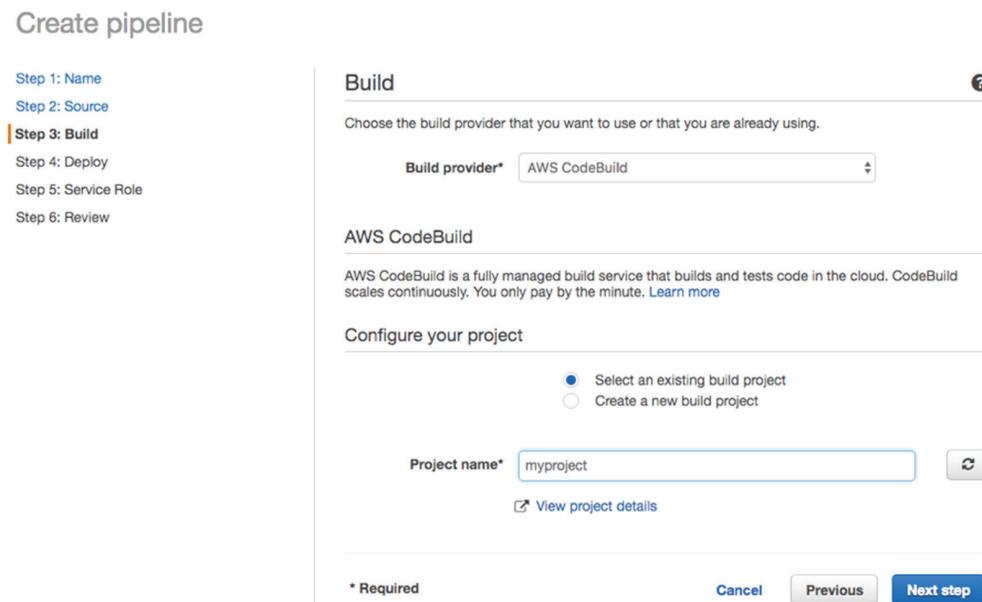
**TABLE 7.4** AWS CodeBuild Service Limits

Limit	Value
Build projects per region per account*	1,000
Build timeout	8 hours
Concurrently running builds*	20

## Using AWS CodeBuild with AWS CodePipeline

AWS CodePipeline enables you to build jobs for both build and test actions. Both action types require exactly one input artifact and may return zero or one output artifacts. When you create a build or test actions in your pipeline with your build projects, the only input that you require is the build project name. The AWS CodePipeline console also has the option to create new build projects when you create the action, as shown in Figure 7.18.

**FIGURE 7.18** Build provider



## Using AWS CodeDeploy to Deploy Applications

*AWS CodeDeploy* is a service that automates software deployments to a variety of compute services, such as Amazon EC2, AWS Lambda, and instances running on-premises. AWS CodeDeploy makes it easier for you to release new features rapidly, helps you avoid downtime through application deployment, and handles the complexity to update your applications. You can use AWS CodeDeploy to automate software deployments and eliminate the need for error-prone manual operations. The service scales to match your deployment needs, from a single AWS Lambda function to thousands of Amazon EC2 instances.

## What Is AWS CodeDeploy?

AWS CodeDeploy standardizes and automates deployments of any types of content or configuration to Amazon EC2 instances, on-premises servers, or AWS Lambda functions. Because of its flexibility, it is not restricted to deploy only application code, and it can perform various administrative tasks that are part of your deployment process. Additionally, you can create custom deployment configurations tailored to your specific infrastructure needs.

### AWS CodeDeploy and NGINX

You can install and enable NGINX as part of a deployment of configuration files to reverse proxy instances. The service itself does not involve any changes to your current source code, and it only requires you to install a lightweight agent on any managed instances or on-premises servers.

Should deployments fail in your environment, you can configure AWS CodeDeploy with a predetermined failure tolerance. Once this tolerance is breached, deployment will automatically roll back to the last version that works.

You can automate deployment of AWS CodeDeploy with AWS Lambda functions through traffic switching. When updates to functions deploy, AWS CodeDeploy will create new versions of each updated function and gradually route requests from the previous version to the updated function. AWS Lambda functions also support custom deployment configurations, which can specify the rate and percentage of traffic to switch.

## AWS CodeDeploy Concepts

When you deploy to Amazon EC2 on-premises instances, a revision occurs.

### Revision

A *revision* is an artifact that contains both application files to deploy and an AppSpec configuration file. Application files can include compiled libraries, configuration files, installation packages, static media, and other content. The AppSpec file specifies what steps AWS CodeDeploy will follow when it performs deployments of an individual revision.

A revision must contain any source files and scripts to execute on the target instance inside a root directory. Within this root directory, the `appspec.yml` file must exist at the topmost level and *not* in any subfolders.

```
/tmp/ or c:\temp (root folder)
|--content (subfolder)
| |--myTextFile.txt
| |--mySourceFile.rb
| |--myExecutableFile.exe
| |--myInstallerFile.msi
```

```
| |--myPackage.rpm
| |--myImageFile.png
|--scripts (subfolder)
| |--myShellScript.sh
| |--myBatchScript.bat
| |--myPowerShellScript.ps1
|--appspec.yml
```

When you deploy to AWS Lambda, a revision contains only the AppSpec file. It contains information about the functions to deploy, as well as the steps to validate that the deployment was successful.

In either case, when a code revision is ready to deploy, you package it into an archive file and store it in one of these three repositories:

- Amazon S3
- GitHub
- Bitbucket

When you use GitHub or Bitbucket, the source code does not need to be a .zip archive, as AWS CodeDeploy will package the repository contents on your behalf. Amazon S3, however, requires a .zip archive file.



AWS Lambda deployments support only Amazon S3 buckets as a source repository.

## Deployments

A *deployment* is the process of copying content and executing scripts on instances in your deployment group. To accomplish this, AWS CodeDeploy performs the tasks outlined in the AppSpec configuration file. For both Amazon EC2 on-premises instances and AWS Lambda functions, the deployment succeeds or fails based on whether individual AppSpec tasks complete successfully. *There are two types of deployments supported by AWS CodeDeploy: in-place and blue/green.*

### In-Place Deployments

In *in-place* deployments, revisions deploy to new infrastructure instead of an existing one. After deployment completes successfully, the new infrastructure gradually replaces old code in a phased rollout. After all traffic routes to the new infrastructure, you can keep the old code for review or discard it.



On-premises instances do not support blue/green deployments.

## Blue/Green Deployments

When you deploy to AWS Lambda functions, *blue/green deployments* publish new versions of each function, after which traffic shifting routes requests to the new function versions according to the deployment configuration that you define.

## Stop Deployments

You can stop deployments via the AWS CodeDeploy console or AWS CLI. If you stop deployments to Amazon EC2 on-premises instances, this can result in some deployment groups being left in an undesired deployment state. For example, when you deploy to instances in an Auto Scaling group, if you stop the deployment, it may result in some instances having different application versions. In situations where this occurs, you can configure the application to roll back to the last valid deployment automatically. To do this, you submit a new deployment to the instances with the previous revision, and they appear as a new deployment in the console.



Some instances that fail the most recent deployment may still have scripts run or files placed that are part of the failed deployment. If you configure automatic rollbacks, AWS CodeDeploy will attempt to remove any successfully created files.

## Rollbacks

AWS CodeDeploy achieves automatic rollbacks by redeploying the last working revision to any instances in the deployment group (this will generate a new deployment ID). If you do not configure automatic rollbacks for the application, you can perform a manual rollback by redeploying a previous revision as a new deployment. This will accomplish the same result as an automatic rollback.

During the rollback process, AWS CodeDeploy will attempt to remove any file(s) that were created on the instance during the failed deployment. A record of the created files is kept in the location on your instances.

Linux: /opt/codedeploy-agent/deployment-root/deployment-instructions/[deployment-group-id]-cleanup

Windows: C:\ProgramData\Amazon\CodeDeploy\deployment-instructions\[deployment-group-id]-cleanup

The *AWS CodeDeploy agent* that runs on the instance will reference this cleanup file as a record of what files were created during the last deployment.



By default, AWS CodeDeploy will not overwrite any files that were not created as part of a deployment. You can override this setting for new deployments.

AWS CodeDeploy tracks cleanup files; however, script executions are not tracked. Any configuration or modification to the instance that is done by scripts run on your instance cannot be rolled back automatically by AWS CodeDeploy. As an administrator, you will be responsible for implementing logic in your deployment scripts to ensure that the desired state is reached during deployments and rollbacks.

### Test Deployments Locally

If you would like to test whether a revision will successfully deploy to an instance you are able to access, you can use the `codedeploy-local` command in the AWS CodeDeploy agent. This command will search the execution path for an AppSpec file and any content to deploy. If this is found, the agent will attempt a deployment on the instance and provide feedback on the results. This provides a useful alternative to executing the full workflow when you want simply to validate the deployment package.

The following example command attempts to perform a local deployment of an archive file located in Amazon S3:

```
codedeploy-local --bundle-location s3://mybucket/bundle.tgz --type tgz
```



The `codedeploy-local` command requires the AWS CodeDeploy agent that you install on the instance or on-premises server where you execute the command.

### Deployment Group

A *deployment group* designates the Amazon EC2 on-premises instances that a revision deploys. When you deploy to AWS Lambda functions, this specifies what functions will deploy new versions. Deployment groups also specify alarms that trigger automatic rollbacks after a specified number or percentage of instances, or functions fail their deployment.

For Amazon EC2 on-premises deployments, you can add instances to a deployment group based on tag name/value pairs or Amazon EC2 Auto Scaling group names. An individual application can have one or more deployment groups defined. This allows you to separate groups of instances into environments so that changes can be progressively rolled out and tested before going to production. You can identify instances by individual tags or tag groups. If an instance matches one or more tags in a tag group, it is associated with the deployment group. If you would like to require that an instance match multiple tags, each tag must be in a separate tag group. A single deployment group supports up to 10 tags in up to three tag groups.

In Figure 7.19, if tags Environment, Region, and Type are present in tag groups 1, 2, and 3 respectively, then instances must have at least one tag in each tag group to identify with the deployment group.

**FIGURE 7.19** Selecting instances with multiple tags

Environment configuration

Specify any combination of Auto Scaling groups, Amazon EC2 instances, and on-premises instances to add instances to this deployment group.

Auto Scaling groups    Amazon EC2 instances **On-premises instances**

You can add up to three groups of tags for EC2 instances to this deployment group. [Learn more](#)

**One tag group**: Any instance identified by the tag group will be deployed to.

**Multiple tag groups**: Only instances identified by all the tag groups will be deployed to.

**Tag group 1**

	Key	Value	Instances	
1	Environment	Staging	4	
2	Environment	Beta	6	
3				

**Tag group 2**

	Key	Value	Instances	
1	Region	South	2	
2	Region	North	2	
3	Region	East	2	
4				

Remove tag group

**Tag group 3**

	Key	Value	Instances	
1	Type	t2.medium	3	
2	Type	t2.large	4	
3				

Remove tag group

When you create deployment groups, you can also configure the following:

**Amazon SNS notifications** Any recipients that subscribe to the topic will receive notifications when deployment events occur. You must create the topic before you configure this notification, and the AWS CodeDeploy service role must have permission to publish messages to the topic.

**Amazon CloudWatch alarms** You can configure alarms to trigger cancellation and rollback of deployments whenever the metric has passed a certain threshold. For example, you could configure an alarm to trigger when CPU utilization exceeds a certain percentage for instances in an AWS Auto Scaling group. If this alarm triggers, the deployment automatically rolls back. For AWS Lambda deployments, you can configure alarms to monitor function invocation errors.

**Automatic rollbacks** You can configure rollbacks to initiate automatically when a deployment fails or based on Amazon CloudWatch alarms. To test deployments, you can disable automatic rollbacks when you create a new deployment.

### On-Premises Instances

You can host instances for an Amazon EC2 on-premises deployment group in either an AWS account or your own data center. To configure an on-premises instance to work with AWS CodeDeploy, you must complete several tasks. Before you begin, you need to ensure that the instance has the ability to communicate with AWS CodeDeploy service endpoints over HTTPS (port 443). You will also need to create an IAM user that the instance assumes and has permissions to interact with AWS CodeDeploy.

1. Install the AWS CLI on the instance.
2. Configure the AWS CLI with an IAM user. Call the `aws configure` command, and specify the secret key ID and secret access key of the IAM user.
3. Register the instance with AWS CodeDeploy. Call the `aws codedeploy register` AWS CLI command from the on-premises instance. Provide a unique name with the `--instance-name` property. When you execute this command, include an IAM user to associate with the instance and tags to apply.

```
aws deploy register --instance-name AssetTag12010298EX \
--iam-user-arn arn:aws:iam::8039EXAMPLE:user/CodeDeployUser-OnPrem \
--tags Key=Name,Value=CodeDeployDemo-OnPrem \
--region us-west-2
```

4. Register the instance with AWS CodeDeploy. Install the AWS CodeDeploy agent. Run the `aws codedeploy install` AWS CLI command. By default, it will install a basic configuration file with preconfigured settings. If you would like to override this, you can provide your own configuration file with the `--config-file` parameter. If you specify the `--override-config` parameter, this will override the current configuration file on the instance.

```
aws deploy install --override-config \
--config-file /tmp/codedeploy.onpremises.yml \
--region us-west-2
```

After you complete the previous steps, the instance will be available for deployments to the deployment group(s).

## Deploy to Amazon EC2 Auto Scaling Groups

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

## Deployment Configuration

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

## Amazon EC2 On-Premises Deployment Configurations

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

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

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

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

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

### **CodeDeployDefault.AllAtOnce**

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

requires that at least one instance succeed for the deployment to be successful. If all instances fail the deployment itself fails.

For blue/green deployments, AWS CodeDeploy will attempt to deploy to the entire set of replacement instances at the same time and follows the same success criteria as in-place deployments. Once deployment to the replacement instances succeeds (at least one instance deploys successfully), traffic routes to all replacement instances at the same time. The deployment fails only if all traffic routing to replacement instances fails.

### **CodeDeployDefault.HalfAtATime**

For in-place deployments, up to half of the instances in the deployment group deploy at the same time (rounded down). Success criteria for this deployment configuration requires that at least half of the instances (rounded up) deploy successfully.

Blue/green deployments use the same rules for the replacement environment, with the exception that the deployment will fail if less than half of the instances in the replacement environment successfully handle rerouted traffic.

### **CodeDeployDefault.OneAtATime**

For in-place and blue/green deployments, this is the most stringent of the built-in deployment configurations, as it requires all instances to deploy the new application revision successfully, with the exception of the final instance in the deployment. For deployment groups with only one instance, the instance must complete successfully for the deployment to complete.

For blue/green deployments, the same rule applies for traffic routing. If all but the last instance registers successfully, the deployment is successful (with the exception of single-instance environments, where it must register without error).

### **CodeDeployDefault.AllAtOnce**

For in-place deployments, AWS CodeDeploy will attempt to deploy to all instances in the deployment group at the same time. The success criteria for this deployment configuration requires that at least one instance succeed for the deployment to be successful. If all instances fail the deployment, then the deployment itself fails.

For blue/green deployments, AWS CodeDeploy will attempt to deploy to the entire set of replacement instances at the same time and follows the same success criteria as in-place deployments. Once deployment to the replacement instances succeeds (at least one instance deploys successfully), traffic routes to all replacement instances at the same time. The deployment fails only if all traffic routing to replacement instances fails.

### **CodeDeployDefault.HalfAtATime**

For in-place deployments, up to half of the instances in the deployment group deploy at the same time (rounded down). Success criteria for this deployment configuration requires that at least half of the instances (rounded up) deploy successfully.

Blue/green deployments use the same rules for the replacement environment, with the exception that the deployment will fail if less than half of the instances in the replacement environment successfully handle rerouted traffic.

### **CodeDeployDefault.OneAtATime**

This is the most stringent of the built-in deployment configurations, as it requires that all instances successfully deploy the new application revision (both in-place and blue/green deployments), with the exception of the final instance in the deployment. For deployment groups with only one instance, the instance must complete successfully for the deployment to complete.

For blue/green deployments, the same rule applies for traffic routing. If all but the last instance registers successfully, the deployment is successful (with the exception of single-instance environments, where it must register without error).

## **AWS Lambda Deployment Configurations**

AWS CodeDeploy handles updates to AWS Lambda functions differently than to Amazon EC2 or on-premises instances. When you deploy to AWS Lambda, the deployment configuration specifies the traffic switching policy to follow, which stipulates how quickly to route requests from the original function versions to the new versions. You can configure AWS CodeDeploy to deploy instances only in a blue/green fashion. AWS Lambda does not support in-place deployments. This is because AWS CodeDeploy will deploy updates to new functions.

AWS CodeDeploy supports three methods for handling traffic switching in an AWS Lambda environment.

### **Canary**

Traffic shifts in two percentage-based increments. The first increment routes to the new function version, and it is monitored for the number of minutes you define. After this time period, the remainder of traffic routes to the new version if the initial increment of request executes.

AWS CodeDeploy provides a number of built-in canary-based deployment configurations, such as `CodeDeployDefault.LambdaCanary10Percent15Minutes`. If you use this deployment configuration, 10 percent of traffic shifts in the first increment and is monitored for 15 minutes. After this time period, the 90 percent of traffic that remains shifts to the new function version. You can create additional configurations as needed.

### **Linear**

Traffic can be shifted in a number of percentage-based increments, with a set number of minutes between each increment. During the waiting period between each increment, the requests routed to the new function versions must complete successfully for the deployment to continue.

AWS CodeDeploy provides a number of built-in linear deployment configurations, such as `CodeDeployDefault.LambdaLinear10PercentEvery1Minute`. With this configuration, 10 percent of traffic is routed to the new function version every minute, until all traffic is routed after 10 minutes.

### **All-at-Once**

All traffic is shifted at once to the new function versions.

## Application

An *application* is a logical grouping of a deployment group, revision, and deployment configuration. This serves as a reference to the entire set of objects needed to complete a deployment to your instances or functions.

## AppSpec File

The AppSpec configuration file is a JSON or YAML file that manages deployments on instances or functions in your environment. The actual format and purpose of an AppSpec file differs between Amazon EC2/on-premises and AWS Lambda deployments.

### Amazon EC2 On-Premises AppSpec

For Amazon EC2 on-premises deployments, the AppSpec file must be YAML formatted and follow the YAML specifications for spacing and indentation. You place the AppSpec file (`appspec.yml`) in the root of the revision's source code directory structure (it cannot be in a subfolder).

When you deploy to Amazon EC2 on-premises instances, the AppSpec file defines the following:

- A mapping of files from the revision and location on the instance
- The permissions of files to deploy
- Scripts to execute throughout the lifecycle of the deployment

The AppSpec file specifies scripts to execute at each stage of the deployment lifecycle. These scripts must exist in the revision for AWS CodeDeploy to call them successfully; however, they can call any other scripts, commands, or tools present on the instance. The AWS CodeDeploy agent uses the hooks section of the AppSpec file to reference which scripts must execute at specific times in the deployment lifecycle. When the deployment is at the specified stage (such as `ApplicationStop`), the AWS CodeDeploy agent will execute any scripts in that stage in the hooks section of the AppSpec file. *All scripts must return an exit code of 0 to be successful.*

For any files to place on the instance, the AWS CodeDeploy agent refers to the `files` section of the AppSpec file, where a mapping of files and directories in the revision dictates where on the instance these files reside and with what permissions. Here's an example of an `appspec.yml` file:

```
version: 0.0
os: linux
files:
 - source: /
 destination: /var/www/html/WordPress
hooks:
 BeforeInstall:
 - location: scripts/install_dependencies.sh
 timeout: 300
 runas: root
```

```
AfterInstall:
 - location: scripts/change_permissions.sh
 timeout: 300
 runas: root
ApplicationStart:
 - location: scripts/start_server.sh
 - location: scripts/create_test_db.sh
 timeout: 300
 runas: root
ApplicationStop:
 - location: scripts/stop_server.sh
 timeout: 300
 runas: root
```

In the previous example, the following events occur during deployment:

- During the install phase of the deployment, all files from the revision (source: /) are placed on the instance in the /var/www/html/WordPress directory.
- The `install_dependencies.sh` script (located in the `scripts` directory of the revision) executes during the `BeforeInstall` phase.
- The `change_permissions.sh` script executes in the `AfterInstall` phase.
- The `start_server.sh` and `create_test_db.sh` scripts execute in the `ApplicationStart` phase.
- The `stop_server.sh` script executes in the `ApplicationStop` phase.

The high-level structure of an Amazon EC2 on-premises AppSpec file is as follows:

```
version: 0.0
os: operating-system-name
files:
 source-destination-files-mappings
permissions:
 permissions-specifications
hooks:
 deployment-lifecycle-event-mappings
```

**version** Currently the only supported version number is 0.0.

**os** The `os` section defines the target operating system of the deployment group. Either `windows` or `linux` (Amazon Linux, Ubuntu, or Red Hat Enterprise Linux) is supported.

**files** The `files` section defines the mapping of revision files and their location to deploy on-instance during the install lifecycle event. This section is not required if no files are being

copied from the revision to your instance. The `files` section supports a list of source/destination pairs.

```
files:
 - source: source-file-location
 destination: destination-file-location
```

The `source` key refers to a file or a directory's local path within the revision (use `/` for all files in the revision). If `source` refers to a file, the file copies to `destination`, specified as the fully qualified path on the instance. If `source` refers to a directory, the directory contents copy to the instance.

**permissions** For any deployed files or directories, the `permissions` section specifies the permissions to apply to files and directories on the target instance. You can also apply permissions to files on the instance by AWS CodeDeploy using the `files` directive of the AppSpec configuration.

```
permissions:
 - object: object-specification
 pattern: pattern-specification
 except: exception-specification
 owner: owner-account-name
 group: group-name
 mode: mode-specification
 acls:
 - acls-specification
 context:
 user: user-specification
 type: type-specification
 range: range-specification
 type:
 - object-type
```

Each object specification includes a set of files or directories to which the permissions will apply. You can select files based on a `pattern` expression and ignore them with a comma-delimited list in the `except` property. The `owner`, `group`, and `mode` properties correspond to their Linux equivalents. You can apply access control lists with the `acls` property, providing a list of user/group permissions assignments (such as `u:ec2-user:rw`). The `context` property is reserved for SELinux-enabled instances. This property corresponds to a set of context labels to apply to objects. Lastly, you use the `type` property to specify to which types of objects (file or directory) the specified permissions will apply.



Windows instances do not support permissions.

**hooks** The hooks section specifies the scripts to run at each lifecycle event and under what user context to execute them.

One or more scripts can execute for each *lifecycle hook*.

**ApplicationStop** Before the application revision downloads to the instance, this lifecycle event can stop any running services on the instance that would be affected by the update. It is important to note that, since the revision has not yet been downloaded, the scripts execute from the previous revision. Because of this, the ApplicationStop hook does not run on the first deployment to an instance.

**DownloadBundle** The AWS CodeDeploy agent uses this lifecycle event to copy application revision files to a temporary location on the instance.

**Linux** /opt/codedeploy-agent/deployment-root/[deployment-group-id]/[deployment-id]/deployment-archive

**Windows** C:\ProgramData\Amazon\CodeDeploy\[deployment-group-id]\[deployment-id]\deployment-archive

This event cannot run custom scripts, as it is reserved for the AWS CodeDeploy agent.

**BeforeInstall** Use this event for any pre-installation tasks, such as to clear log files or to create backups.

**Install** This event is reserved for the AWS CodeDeploy agent.

**AfterInstall** Use this event for any post-installation tasks, such as to modify the application configuration.

**ApplicationStart** Use this event to start any services that were stopped during the ApplicationStop event.

**ValidateService** Use this event to verify deployment completed successfully.

If your deployment group is registered with a load balancer, additional lifecycle events become available. These can be used to control certain behaviors as the instance is registered or deregistered from the load balancer.

**BeforeBlockTraffic** Use this event to run tasks before the instance is deregistered from the load balancer.

**BlockTraffic** This event is reserved for the AWS CodeDeploy agent.

**AfterBlockTraffic** Use this event to run tasks after the instance is deregistered from the load balancer.

**BeforeAllowTraffic** Similar in concept to BeforeBlockTraffic, this event occurs before instances register with the load balancer.

**AllowTraffic** This event is reserved for the AWS CodeDeploy agent.

**AfterAllowTraffic** Similar in concept to AfterBlockTraffic, this event occurs after instances register with the load balancer.

hooks:

```
deployment-lifecycle-event-name:
 - location: script-location
 timeout: timeout-in-seconds
 runas: user-name
```

In the hooks section, the lifecycle name must match one of the previous event names, which are not reserved for the AWS CodeDeploy agent. The location property refers to the relative path in the revision archive where the script is located. You can configure an optional timeout to limit how long a script can run before it is considered failed. (Note that this does not stop the script's execution.) *The maximum script duration is 1 hour (3,600 seconds) for each lifecycle event.* Lastly, the runas property can specify the user to execute the script. This user must exist on the instance and cannot require a password.

Figure 7.20 displays lifecycle hooks and their availability for in-place deployments with and without a load balancer.

**FIGURE 7.20** Lifecycle hook availability with load balancer

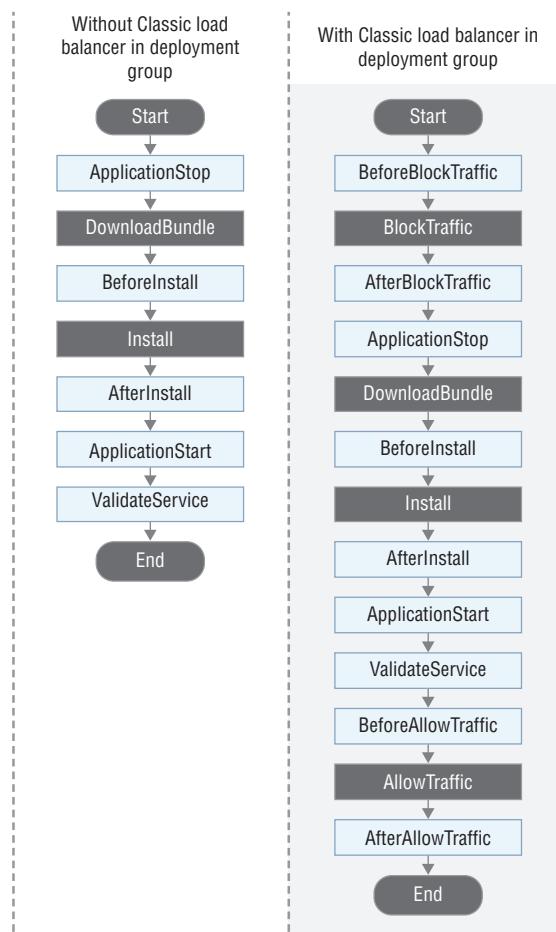
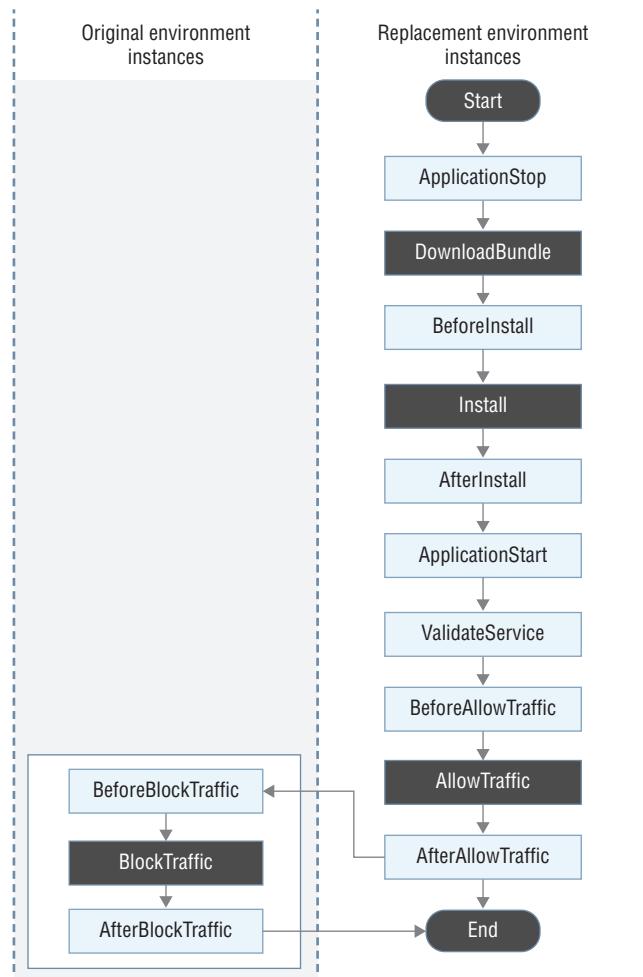


Figure 7.21 displays lifecycle hooks and their availability for blue/green deployments.

**FIGURE 7.21** Lifecycle hook availability with blue/green deployments



### AWS Lambda AppSpec

When you deploy to AWS Lambda functions, the AppSpec file can be in JSON or YAML format, and it specifies the function versions to deploy as well as other functions to execute for validation testing.



AWS Lambda deployments do not use the AWS CodeDeploy agent.

The high-level structure of an AWS Lambda deployment AppSpec file is as follows:

```
version: 0.0
resources:
 lambda-function-specifications
hooks:
 deployment-lifecycle-event-mappings
```

**version** Currently the only supported version number is 0.0.

**resources** The resources section defines the AWS Lambda functions to deploy.

```
resources:
- name-of-function-to-deploy:
 type: "AWS::Lambda::Function"
 properties:
 name: name-of-lambda-function-to-deploy
 alias: alias-of-lambda-function-to-deploy
 currentversion: lambda-function-version-traffic-currently-points-to
 targetversion: lambda-function-version-to-shift-traffic-to
```

Name each function in the **resources** list both as the list item name and in the **name** property. The **alias** property specifies the function alias, which maps from the version specified in **currentversion** to the version specified in **targetversion** after the update deploys.

**hooks** The hooks section specifies the additional AWS Lambda functions to run at specific stages of the deployment lifecycle to validate success. The following lifecycle events support hooks in AWS Lambda deployments:

**BeforeAllowTraffic** For running any tasks prior to traffic shifting taking place

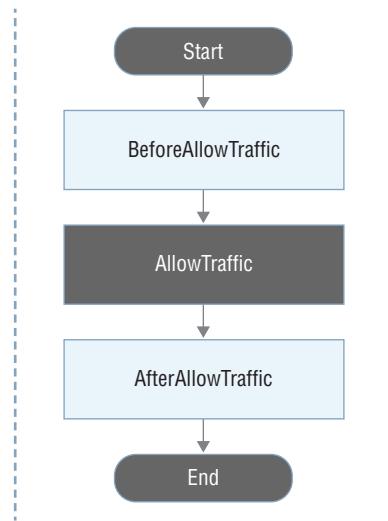
**AfterAllowTraffic** For any tasks after all traffic shifting has completed

```
hooks:
- BeforeAllowTraffic: BeforeAllowTrafficHookFunctionName
- AfterAllowTraffic: AfterAllowTrafficHookFunctionName
```

Figure 7.22 displays the lifecycle hook availability for AWS Lambda deployments.



AWS CodeDeploy reserves the Start, AllowTraffic, and End lifecycle events.

**FIGURE 7.22** Lifecycle hook availability for AWS Lambda deployments

For any functions in the hooks section, the function is responsible for notifying AWS CodeDeploy of success or failure with the `PutLifecycleEventHookExecutionStatus` call API from within your validation function. Here's an example for Node.js:

```
CodeDeploy the prepared validation test results.
codedeploy.putLifecycleEventHookExecutionStatus(params, function(err, data) {
 if (err) {
 // Validation failed.
 callback('Validation test failed');
 } else {
 // Validation succeeded.
 callback(null, 'Validation test succeeded');
 }
});
```

## AWS CodeDeploy Agent

The *AWS CodeDeploy agent* is responsible for driving and validating deployments on Amazon EC2 on-premises instances. The agent currently supports Amazon Linux (Amazon EC2 only), Ubuntu Server, Microsoft Windows Server, and Red Hat Enterprise Linux, and it is available as an open source repository on GitHub (<https://github.com/aws/aws-codedeploy-agent>).

When the agent installs, a `codedeployagent.yml` configuration file copies to the instances. You can use this file to adjust the behavior of the AWS CodeDeploy agent on instances throughout various deployments. This configuration file is stored in `/etc/codedeploy-agent/conf` on Linux instances and `C:\ProgramData\Amazon\AWS CodeDeploy` on Windows Server instances.

The most common settings are as follows:

**max\_revisions** Use this to configure how many application revisions to archive on an instance. If you are experiencing storage limitations on your instances, turn this value down and release some storage space consumed by the agent.

**root\_dir** Use this to change the default storage location for revisions, scripts, and archives.

**verbose** Set this to true to enable verbose logging output for debugging purposes.

**proxy\_url** For environments that use an HTTP proxy, this specifies the URL and credentials to authenticate to the proxy and connect to the AWS CodeDeploy service.

## AWS CodeDeploy Service Limits

AWS CodeDeploy enforces the service limits, as shown in Table 7.5. An asterisk (\*) indicates limits that you can increase with a request to AWS Support.

**TABLE 7.5** AWS CodeDeploy Service Limits

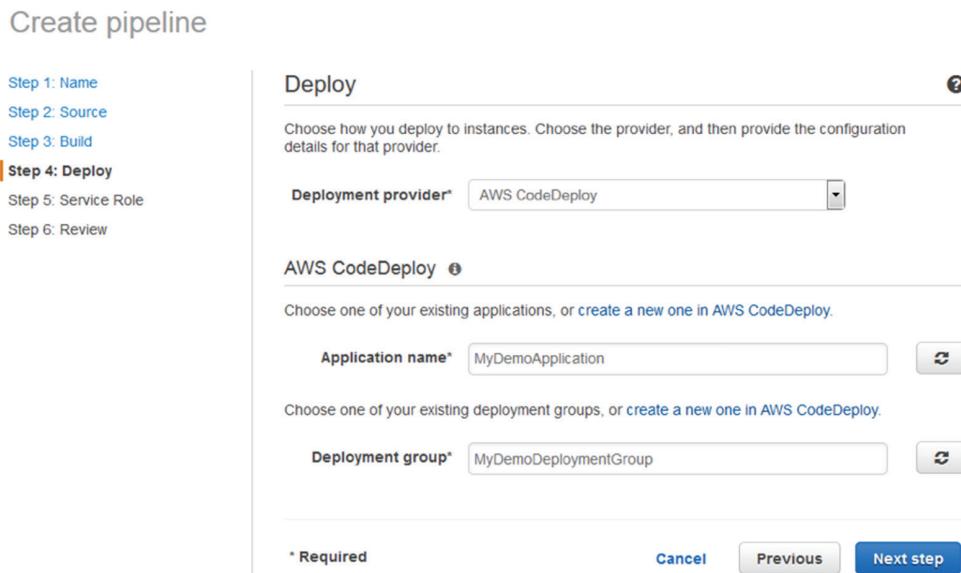
Limit	Value
Applications per account per region	100
Allowed revision file types	.zip, .tar, .tar, and.gz
Concurrent deployments per deployment group	1
Concurrent deployments per account	100
Maximum deployment lifecycle event duration	3600 seconds
Custom deployment configurations per account	25
Deployment groups per application*	100
Tags per deployment group	10
Auto Scaling groups per deployment group	10
Instances per deployment	500

## Using AWS CodeDeploy with AWS CodePipeline

AWS CodeDeploy can integrate automatically with AWS CodePipeline as a deployment action to deploy changes to Amazon EC2 on-premises instances or AWS Lambda functions. You can configure applications, deployment groups, and deployments directly in the AWS CodePipeline console when you create or edit a pipeline, or you can do this ahead of time with the AWS CodeDeploy console or the AWS CLI or AWS SDK.

After you define the deployment provider, application name, and deployment group in the AWS CodePipeline console, the pipeline will automatically configure to pass a pipeline artifact to AWS CodeDeploy for deployment to the specified application/group, as shown in Figure 7.23.

**FIGURE 7.23** Deployment provider



AWS CodeDeploy monitors the progress of any revisions to deploy and report success or failure to AWS CodePipeline.

## Summary

In this chapter, you learned about these deployment services:

- AWS CodePipeline
- AWS CodeCommit
- AWS CodeBuild
- AWS CodeDeploy

AWS CodePipeline drives application deployments starting with a source repository (AWS CodeCommit), performing builds with AWS CodeBuild, and finally deploying to Amazon EC2 instance or AWS Lambda functions using AWS CodeDeploy. You can use AWS CloudFormation to provision and manage infrastructure in your environment. By integrating this with AWS CodePipeline, you can automate the entire process of creating development, testing, and production environments into a fully hands-off process. In a fully realized enterprise as code, a single commit to a source repository can kick off processes such as those shown in Figure 7.1.

## Exam Essentials

**Know the difference between continuous integration, continuous delivery, and continuous deployment.** *Continuous integration* is the practice where all code changes merge into a repository. *Continuous delivery* is the practice where all code changes are prepared for release. *Continuous deployment* is the practice where all code is prepared for release and automatically released to production environments.

**Know the basics of AWS CodePipeline.** AWS CodePipeline contains the steps in the *continuous integration and deployment pipeline* (CI/CD) workflow, driving automation between different tasks after assets have been committed to a repository or saved in a bucket. AWS CodePipeline uses stages, which correspond to different steps in a workflow. Within each stage, different actions can perform tasks in series or in parallel. Transitions between stages can be automatic or require manual approval by an authorized user.

**Understand how revisions can move through a pipeline.** Revisions move automatically between stages in a pipeline, provided that all actions in the preceding stage complete. If a manual approval is required, the revision will not proceed until an authorized user allows it to do so. When two changes are pushed to a source repository in a short time span, the latest of the two changes will proceed through the pipeline.

**Know the different pipeline actions that are available.** A pipeline stage can include one or more actions: build, test, deploy, and invoke. You can also create custom actions.

**Know how to deploy a cross-account pipeline.** The account containing the pipeline must create a KMS key that can be used by both AWS CodePipeline and the other account. The pipeline account must also specify a bucket policy on the assets bucket that the pipeline uses, which allows the second account to access assets. The AWS CodePipeline service IAM role must include a policy that allows it to assume a role in the second account. The second account must have a role that can be assumed by the pipeline account, which allows the pipeline account to deploy resources and access the assets bucket.

**Know the basic concepts of AWS CodeCommit.** AWS CodeCommit is a Git-based repository service. It is fully compatible with existing Git tooling. AWS CodeCommit provides various benefits, such as encryption in transit and at rest; automatic scaling to handle increases in activity; access control using IAM users, roles, and policies; and HTTPS/SSH connectivity. AWS CodeCommit supports normal Git workflows, such as pull requests.

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

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

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

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

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

## Resources to Review

What is DevOps?

<https://aws.amazon.com/devops/what-is-devops/>

AWS DevOps Blog:

<https://aws.amazon.com/blogs/devops/>

Introduction to DevOps on AWS:

[https://d1.awsstatic.com/whitepapers/AWS\\_DevOps.pdf](https://d1.awsstatic.com/whitepapers/AWS_DevOps.pdf)

Practicing Continuous Integration and Continuous Delivery on AWS:

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

AWS CodePipeline User Guide:

<https://docs.aws.amazon.com/codepipeline/latest/userguide/welcome.html>

Set Up a CI/CD Pipeline on AWS:

<https://aws.amazon.com/getting-started/projects/set-up-ci-cd-pipeline/>

AWS CodePipeline:

<https://aws.amazon.com/codepipeline/>

AWS CodeCommit:

<https://aws.amazon.com/codecommit/>

AWS CodeBuild:

<https://aws.amazon.com/codebuild/>

AWS CodeDeploy:

<https://aws.amazon.com/codedeploy/>

## Exercises

### EXERCISE 7.1

#### Create an AWS CodeCommit Repository and Submit a Pull Request

This exercise demonstrates how to use AWS CodeCommit to submit and merge pull requests to a repository.

1. Create an AWS CodeCommit repository with a name and description. You do not need to configure email notifications for repository events.
2. In the AWS CodeCommit console, select **Create File** to add a simple markdown file to test the repository.
3. Clone the repository to your local machine with HTTPS or SSH authentication.
4. Create a file locally, commit it to the repository, and push it to test the AWS CodeCommit.
5. Create a feature branch from the master branch in the repository.
6. Edit the file and commit the changes to the feature branch.
7. Use the AWS CodeCommit console to create a pull request. Use the master branch of the repository as the destination and the feature branch as the source.
8. After the pull request successfully creates, merge the changes from the feature branch with the master branch.

The pull request has been merged with the master branch, which can be confirmed by viewing the source code of the markdown file in the master branch.

**EXERCISE 7.2****Create an Application in AWS CodeDeploy**

This exercise demonstrates how to use AWS CodeDeploy to perform an in-place deployment to Amazon EC2 instances in your account.

1. Create a new application in the AWS CodeDeploy console.  
For the compute platform type, select **EC2 On-premises**.
2. Create a new deployment group for your application. Specify the following values:

**Deployment type** In-place

**Environment configuration** Amazon EC2 instances

**Tag group** Create a tag group that is easy to identify, such as a “Name” for the key, and “CodeDeployInstance” as the value.

**Load balancer** Clear the **Enable load balancing** check box.

3. Launch new **Amazon EC2 instance**.

Make sure to specify the tag value chosen in the previous step.

4. Download the **sample application bundle** to your local machine for future updates.

Sample application bundles for each operating system can be found using the following links:

**Windows Server** <https://docs.aws.amazon.com/codedeploy/latest/userguide/tutorials-windows.html>

**Amazon Linux or Red Hat Enterprise Linux (RHEL)** <https://docs.aws.amazon.com/codedeploy/latest/userguide/tutorials-wordpress.html>

5. Create a deployment group, and verify that the sample application **deploys**.

6. **Update** the application code, and **submit** a new deployment to the deployment group.

7. Verify your changes after the deployment completes.
- 

**EXERCISE 7.3****Create an AWS CodeBuild Project**

This exercise demonstrates how to use AWS CodeBuild to perform builds and the compilation of artifacts prior to deployment to Amazon EC2 instances.

1. Create an Amazon S3 bucket to hold artifacts.
  2. Upload two or more arbitrary files to the bucket.
- 

*(continued)*

**EXERCISE 7.3 (continued)**

3. Use the AWS CodeBuild console to create a build project with the following settings:

**Project name** Provide a name of your choice.

**Source** Use Amazon S3.

**Bucket** Provide the name of the bucket you created.

**S3 object key** Provide the name of one of the objects you uploaded.

**Environment image** Select the **Managed Image** type.

**Operating system** Use **Ubuntu**.

**Runtime** Use **Python**.

**Runtime version** Select a version of your choice.

**Service role** Select **New Service Role**.

**Role name** Provide a name for your service role.

**Build specifications** Select **Insert Build Commands**.

**Build commands** Select **Switch To Editor** and enter the following. Replace the Amazon S3 object paths with paths to the objects you uploaded to your bucket.

```
version: 0.2
```

```
phases:
```

```
 build:
```

```
 commands:
```

```
 - aws s3 cp s3://yourbucket/file1 /tmp/file1
 - aws s3 cp s3://yourbucket/file2 /tmp/file2
```

```
artifacts:
```

```
 files:
```

```
 - /tmp/file1
 - /tmp/file2
```

**Artifact Type** Use **Amazon S3**.

**Bucket name** Select your Amazon S3 bucket.

**Artifacts packaging** Select **Zip**.

4. Save your build project.

5. Run your build project, and observe the output archive file created in your Amazon S3 bucket.
-

# Review Questions

1. You have two AWS CodeDeploy applications that deploy to the same Amazon EC2 Auto Scaling group. The first deploys an e-commerce app, while the second deploys custom administration software. You are attempting to deploy an update to one application but cannot do so because another deployment is already in progress. You do not see any instances undergoing deployment at this time. What could be the cause of this?
  - A. If both deployment groups reference the same Auto Scaling group, a failure of the first group's deployment can block the second until the deployment times out. Since the instance that failed deployment has been terminated from the Auto Scaling group, the AWS CodeDeploy agent is unable to provide results to the service.
  - B. The AWS CodeDeploy agent is not installed on the instances as part of the launch configuration user data script.
  - C. If both deployment groups reference the same Auto Scaling group, a failure of the first group's deployment can block the second until the deployment times out. Since the instance that failed deployment has been terminated from the Auto Scaling group, the AWS CodeDeploy service is unable to request status updates from the Amazon EC2 API.
  - D. The AWS CodeDeploy agent is not installed in the Amazon Machine Image (AMI) being used.
2. If you specify a hook script in the `ApplicationStop` lifecycle event of an AWS CodeDeploy `appspec.yml`, will it run on the first deployment to your instance(s)?
  - A. Yes
  - B. No
  - C. The `ApplicationStop` lifecycle event does not exist.
  - D. It will run only if your application is running.
3. If a single pipeline contains multiple sources, such as an AWS CodeCommit repository and an Amazon S3 archive, under what circumstances will the pipeline be triggered?
  - A. When either a commit is pushed to the repository or the archive is updated, regardless of timing.
  - B. When a commit is pushed to the repository and the archive is updated at the same time.
  - C. When either a commit is pushed to the repository or the archive is updated, but not when both are updated at the same time.
  - D. AWS CodePipeline does not support multiple sources in the same pipeline.
4. If you want to implement a deployment pipeline that deploys both source files and large binary objects to instance(s), how would you best achieve this while taking cost into consideration?
  - A. Store both the source files and binary objects in AWS CodeCommit.
  - B. Build the binary objects into the AMI of the instance(s) being deployed. Store the source files in AWS CodeCommit.
  - C. Store the source files in AWS CodeCommit. Store the binary objects in an Amazon S3 archive.

- D. Store the source files in AWS CodeCommit. Store the binary objects on an Amazon Elastic Block Store (Amazon EBS) volume, taking snapshots of the volume whenever a new one needs to be created.
  - E. Store the source files in AWS CodeCommit. Store the binary objects in Amazon S3 and access them from an Amazon CloudFront distribution.
5. Your team is building a deployment pipeline to a sensitive application in your environment using AWS CodeDeploy. The application consists of an Amazon EC2 Auto Scaling group of instances behind an Elastic Load Balancing load balancer. The nature of the application requires 100 percent availability for both successful and failed deployments. The development team want to deploy changes multiple times per day.

How would this be achieved at the lowest cost and with the fastest deployments?

- A. Rolling deployments with an additional batch
  - B. Rolling deployments without an additional batch
  - C. Blue/green deployments
  - D. Immutable updates
6. What would cause an access denied error when attempting to download an archive file from Amazon S3 during a pipeline execution?
- A. Insufficient user permissions for the user initiating the pipeline
  - B. Insufficient user permissions for the user uploading the Amazon S3 archive
  - C. Insufficient role permissions for the Amazon S3 service role
  - D. Insufficient role permissions for the AWS CodePipeline service role
7. How do you output build artifacts from AWS CodeBuild to AWS CodePipeline?
- A. Write the outputs to STDOUT from the build container.
  - B. Specify artifact files in the `buildspec.yml` configuration file.
  - C. Upload the files to Amazon S3 from the build environment.
  - D. Output artifacts are not supported with AWS CodeBuild.
8. What would be the most secure means of providing secrets to an AWS CodeBuild environment?
- A. Create a custom build environment with the secrets included in configuration files.
  - B. Upload the secrets to Amazon S3 and download the object when the build job runs. Protect the bucket and object with an appropriate bucket policy.
  - C. Save the secrets in AWS Systems Manager Parameter Store and query them as needed. Encrypt the secrets with an AWS Key Management Service (AWS KMS) key. Include appropriate AWS KMS permissions to your build environment's IAM role.
  - D. Include the secrets in the source repository or archive.
9. In which of the pipeline actions can you execute AWS Lambda functions?
- A. Invoke
  - B. Deploy

- C. Build
  - D. Approval
  - E. Test
10. In what ways can pipeline actions be ordered in a stage? (Select TWO.)
- A. Series
  - B. Parallel
  - C. Stages support only one action each
  - D. First-in-first-out (FIFO)
  - E. Last-in-first-out (LIFO)
11. If you would like to delete an AWS CloudFormation stack before you deploy a new one in your pipeline, what would be the correct set of actions?
- A. One action that specifies “Create or update a stack.”
  - B. Two actions: the first specifies “Create or update a stack,” and the second specifies “Delete a stack.”
  - C. Three actions: the first specifies “Delete a stack,” the second specifies “Create or update a stack,” and the third specifies “Replace a failed stack.”
  - D. Two actions: the first specifies “Delete a stack,” and the second specifies “Create or update a stack.”
12. How can you connect to an AWS CodeCommit repository without Git credentials?
- A. It is not possible.
  - B. HTTPS
  - C. SSH
  - D. AWS CodeCommit credential helper
13. Of the following, which event cannot be used to generate notifications to an Amazon Simple Notification Service (SNS) topic from AWS CodeCommit without using a trigger?
- A. Pull Request Creation
  - B. Commit Comments
  - C. Commit Creation
  - D. Pull Request Comments
14. Which pipeline actions support AWS CodeBuild projects? (Select TWO.)
- A. Invoke
  - B. Deploy
  - C. Build
  - D. Approval
  - E. Test

15. Can data passed to build projects using environment variables be encrypted or protected?
  - A. Yes, this is supported natively by AWS CodeBuild.
  - B. No, it is not supported.
  - C. No, but this can be enabled in the console.
  - D. No, but this can be supported using other AWS products and services.
16. What is the only deployment type supported by on-premises instances?
  - A. In-place
  - B. Blue/green
  - C. Immutable
  - D. Progressive
17. If your AWS CodeDeploy configuration includes creation of a file, nginx.conf, but the file already exists on the server (prior to the use of AWS CodeDeploy), what is the default behavior that will occur during deployment?
  - A. The file will be replaced.
  - B. The file will be renamed nginx.conf.bak, and the new file will be created.
  - C. The deployment will fail.
  - D. The deployment will continue, but the file will not be modified.
18. How does AWS Lambda support in-place deployments?
  - A. Function versions are overwritten during the deployment.
  - B. New function versions are created, and then version numbers are switched.
  - C. AWS Lambda does not support in-place deployments.
  - D. Function aliases are overwritten during the deployment.
19. What is the minimum number of stages required by a pipeline in AWS CodePipeline?
  - A. 0
  - B. 1
  - C. 2
  - D. 3
20. If an instance is running low on storage, and you find that there are a large number of deployment revisions stored by AWS CodeDeploy, what can be done to free up this space permanently?
  - A. Delete the old revisions.
  - B. Add an additional Amazon EBS volume.
  - C. Configure the AWS CodeDeploy agent to store fewer revisions.
  - D. Delete all of the revisions, and push all new code.

# Chapter 8



AWS® Certified Developer Official Study Guide  
By Nick Alteen, Jennifer Fisher, Casey Gerena, Wes Gruver, Asim Jalil,  
Heiwad Osman, Marife Pagan, Santosh Patlolla and Michael Roth  
Copyright © 2019 by Amazon Web Services, Inc.

# Infrastructure as Code

---

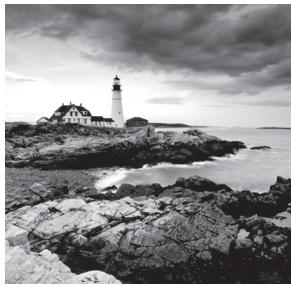
**THE AWS CERTIFIED DEVELOPER –  
ASSOCIATE EXAM TOPICS COVERED IN  
THIS CHAPTER MAY INCLUDE, BUT ARE  
NOT LIMITED TO, THE FOLLOWING:**

**Domain 1: Deployment**

- ✓ 1.1 Infrastructure as Code (IaC).
- ✓ 1.2 Use AWS CloudFormation to Deploy Infrastructure.

**Domain 5: Monitoring and Troubleshooting**

- ✓ 5.1 Custom Resource Success/Failure.



## Introduction to Infrastructure as Code

Chapter 7 covered deployment tools, processes, and methodologies in AWS services. These services can leverage and be read by *AWS CloudFormation* to provision and manage AWS infrastructure from Amazon Elastic Compute Cloud (Amazon EC2) instances to Amazon API Gateway REST APIs. For all intents and purposes, if you provision and update code with an AWS API, you can use AWS CloudFormation to move this process entirely to template code updates.



If you create an AWS Auto Scaling group of instances with the AWS Management Console, you must perform a number of steps. You can launch and test multiple instances of the user data script with Amazon EC2 launch configurations, you can use Amazon CloudWatch alarms to scale your application, and finally you can implement the AWS Auto Scaling group itself. A better solution is to use AWS CloudFormation to create and manage all of the aforementioned resources over time with a simple, declarative template syntax.

## Infrastructure as Code

Using an *infrastructure as code* (IaC) model, instead of manually provisioning or using scripting languages, helps remove the dependency on human intervention when you create and manage infrastructure over time. You can use tools such as AWS CloudFormation to deploy infrastructure from a declarative template syntax. For example, a typical provisioning script that uses the *AWS Command Line Interface* (AWS CLI) includes many procedural steps that are prone to error because of invalid inputs, incorrect command syntax, and resource dependency conflicts. AWS CloudFormation templates provide the ability to validate inputs and automatically detect dependencies between resources.

Provisioning infrastructure with AWS CloudFormation templates provides some built-in benefits, such as the ability to track changes with a “source of truth,” such as a Git-based repository. Since repositories track changes over time, you can roll back an undesired change by resubmitting the last working version of the template(s). This can significantly reduce the time needed to roll back undesired changes.

You can view users' resources with appropriate permissions within an AWS account. An issue can arise where, as your infrastructure grows over time, it can be difficult to determine what resources belong to what functional group, application, team, and so on. Use of tags can alleviate this somewhat, but this is not possible for resources that do not yet support tags. AWS CloudFormation organizes resources into stacks, which you describe in the AWS Management Console, the AWS CLI, or AWS software development kits (AWS SDKs). AWS CloudFormation stacks provide a comprehensive list of any infrastructures in a functional group.

## Using AWS CloudFormation to Deploy Infrastructure

*AWS CloudFormation provides a common language for you to describe and provision all of the infrastructure resources in your cloud environment.*

AWS CloudFormation allows you to use a simple text file to model and provision, in an automated and secure manner, all of the resources for your applications across all regions and accounts. This file serves as the single source of truth for your cloud environment.

AWS CloudFormation is available at no additional charge, and you pay only for the AWS resources required to run your applications.

### What Is AWS CloudFormation?

Before you deploy any application code, the first requirement is that infrastructure exists where you will deploy the code. AWS CloudFormation aims to alleviate previous deployment issues with the use of a service that allows you to describe your infrastructure with standardized JSON or YAML template syntax. The template contains the infrastructure that AWS will deploy and all the related configuration properties. When you submit this template to the AWS CloudFormation service, it creates a stack, which is a logical group of resources that the template describes.

When you manually create resources with the AWS Management Console or AWS CLI or AWS SDK, you cannot easily define relationships between resources.



If you manually create an AWS Auto Scaling Group (ASG) and attach this to an Elastic Load Balancing (ELB) load balancer, it requires several API calls or console actions—one for each resource and one to attach the ASG to the ELB. With AWS CloudFormation, you define the resources and any relationships in one location for easy deployment and updates over time.

Two key benefits of AWS CloudFormation over procedural scripting or manual console actions are that your infrastructure is now *repeatable* and that it is *versionable*.

Any template that you deploy one time in an account you can deploy again (either in the same account and/or region or in others). This offers you an opportunity for dynamically provisioning short-lived environments to test or roll over to a new production environment (blue/green deployment). Since templates describe your infrastructure, you check the templates themselves into a source code repository. With this, you can track changes over time, and updates roll back when they revert commits and redeploy the previous template(s). Over time, this creates self-documenting infrastructure that shows changes over the life-cycle of an environment.

## AWS CloudFormation Concepts

This section details AWS CloudFormation concepts, such as stacks, change sets, permissions, templates, and instinct functions.

### Stacks

A *stack* represents a collection of resources to deploy and manage by AWS CloudFormation. When you submit a template, the resources you configure are provisioned and then make up the stack itself. Any modifications to the stack affect underlying resources. For example, if you remove an `AWS::EC2::Instance` resource from the template and update the stack, AWS CloudFormation causes the referred instance to terminate.



AWS CloudFormation manages all of the resources you declare in a stack when the stack updates. If you manually update the resource outside of AWS CloudFormation, the result will be inconsistencies between the state AWS CloudFormation expects and the actual resource state. This can cause future stack operations to fail.

### Change Sets

There may be times where you would like to see what changes will occur to resources when you update a template, before the update occurs. Instead of submitting the update directly, you can generate a change set. A *change set* is a description of the changes that will occur to a stack, should you submit the template. If the changes are acceptable, the change set itself can execute on the stack and implement the proposed modifications. This is especially important in situations where there is a potential for data loss.

#### Amazon Relational Database Service Instances

There are several properties in Amazon Relational Database Service (Amazon RDS) instances that AWS CloudFormation modifies and requires replacement in the underlying database instance resource. If backups are not being taken, data loss will occur. You use a change set to preview the replacement event, make the necessary backups, and take the required precautions before you update the resources.

## Permissions

AWS CloudFormation, unless otherwise specified, functions within the context of the IAM user or AWS role to invoke a stack action. This means that if you submit a template that creates an Amazon EC2 instance (or instances), AWS CloudFormation will fail unless your IAM user or AWS role has permissions to create instances. Any action that AWS CloudFormation performs is done on your behalf, with your authorizations. With this, you can control what stack actions perform (create, update, or delete) and what actions are performed on the underlying resources.

If there is a need to restrict what permissions a single IAM user or AWS role can have, you can provide a service role the stack uses for the create, update, or delete actions. When the role passes to AWS CloudFormation, it will use the role's credentials to determine what operations it performs. To create an AWS CloudFormation service role, make sure that the role as a trust policy allows `cloudformation.amazonaws.com` to assume the role.

As a user, your IAM credentials will need to include the ability to pass the role to AWS CloudFormation, using the `iam:PassRole` permission. An additional benefit when you use a service role is that it will extend the default timeout for stack create, update, and delete actions. This is especially important when you work with resources that take a longer time because of their size or distribution. Certain services can time out in AWS CloudFormation, returning a Resource failed to stabilize error.

- `AWS::AutoScaling::AutoScalingGroup`
- `AWS::CertificateManager::Certificate`
- `AWS::CloudFormation::Stack`
- `AWS::ElasticSearch::Domain`
- `AWS::RDS::DBCluster`
- `AWS::RDS::DBInstance`
- `AWS::Redshift::Cluster`



After a service role passes to AWS CloudFormation, other users with the ability to perform updates will be able to do so with the same role, regardless of whether they have the ability to pass it. Make sure that the service role follows least-privilege practices.

When you assign permissions for IAM users or AWS roles, you have the ability to specify conditions to control whether policies are in effect. For example, you can allow your users to create stacks only with certain names. However, do not use the `aws:SourceIp` condition. This is because AWS CloudFormation actions originate from AWS IP addresses, not the IP address of the request.

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

- `cloudformation:CreateUploadBucket`
- `s3:PutObject`
- `s3>ListBucket`
- `s3:GetObject`
- `s3>CreateBucket`

## Template Structure

AWS CloudFormation uses specific template syntax in JSON or YAML. (The primary difference is YAML's support of comments using the `#` symbol.) The high-level structure of a template is as follows:

```
{
 "AWSTemplateFormatVersion": "2010-09-09",
 "Description": "String Description",
 "Metadata": { },
 "Parameters": { },
 "Mappings": { },
 "Conditions": { },
 "Transform": { },
 "Resources": { },
 "Outputs": { }
}
```

Of the previous properties, *AWS CloudFormation requires only the Resources section*. Each property can be in any order, with the exception that `Description` must follow the `AWSTemplateFormatVersion` command.

### **AWSTemplateFormatVersion**

`AWSTemplateFormatVersion` corresponds to the template version to which this template adheres. Do not confuse this with an API version or the version of the developer's template draft. Currently, AWS CloudFormation only supports the value `"2010-09-09"`, which you must provide as a literal string.

### **Description**

The `Description` section allows you to provide a text explanation of the template's purpose or other arbitrary information. The maximum length of the `Description` field is 1,024 bytes. Similar to the `AWSTemplateFormatVersion` section, `Description` supports only literal text.

## Metadata

The *Metadata* section of a template allows you to provide structured details about the template. For example, you can provide Metadata about the overall infrastructure to deploy and which sections correspond to certain environments, functional groups, and so on. The Metadata you provide is made available to AWS CloudFormation for reference in other sections of a template or on Amazon EC2 instances being provisioned by AWS CloudFormation.

### Updating the Metadata Section of a Template

You cannot update template metadata by itself; you must perform an update to one or more resources when you update the Metadata section of a template.

```
"Metadata": {
 "ApplicationLayer": {
 "Description": "Information about resources in the app layer."
 },
 "DatabaseLayer": {
 "Description": "Information about resources in the DB layer."
 }
}
```

In the *Metadata* section of the template, you have the ability to specify properties that affect the behavior of different components of the AWS CloudFormation service, such as how template parameters display in the AWS CloudFormation console.

## Parameters

You can use *Parameters* to provide inputs into your template, which allows for more flexibility in how this template behaves when you deploy it. Parameter values can be set either when you create the stack or when you perform updates.

The *Parameters* section must include a unique logical ID (in the next example, `InstanceTypeParameter`). A parameter must include a value, either a default or one that you provide. Lastly, you cannot reference parameters outside a single template.

### AllowedValues Error

This example defines a String parameter named `InstanceTypeParameter` with a default value of `t2.micro`. The parameter allows `t2.micro`, `m1.small`, or `m1.large`. The AllowedValues section specifies what options you can select for this parameter in the AWS CloudFormation console. AWS CloudFormation will throw an error if you add a value not in AllowedValues.

```
"Parameters": {
 "InstanceTypeParam": {
 "Type": "String",
```

*(continued)*

(continued)

```
"Default": "t2.micro",
"AllowedValues": ["t2.micro", "m1.small", "m1.large"],
"Description": "Enter t2.micro, m1.small, or m1.large. Default is t2.micro."
}
}
```

Once you specify a parameter, you can use it within the template using the Ref intrinsic function. When AWS CloudFormation evaluates it, the Ref statement converts it to the value of the parameter.

```
"EC2Instance": {
 "Type": "AWS::EC2::Instance",
 "Properties": {
 "InstanceType": { "Ref": "InstanceTypeParam" },
 "ImageId": "ami-12345678"
 }
}
```

AWS CloudFormation supports the following parameter types:

- String
- Number
- List of numbers
- Comma-delimited list
- AWS parameter types
- AWS Systems Manager Parameter Store (Systems Manager) parameter types

If a parameter value is sensitive, you can add the NoEcho property. When this is set, the parameter value displays as asterisks (\*\*\* ) for any `cloudformation:Describe*` calls. Within the template itself, the value will resolve to the actual input when making Ref calls.

**AWS parameter types** When you use AWS *parameter types*, AWS CloudFormation automatically queries existing properties and values within your AWS account. This can include information such as Amazon EC2 key pair names, IDs of resources, AWS regions/availability zones, or other properties of your account. These input values must exist in your account and are validated to ensure that they are correct. For example, you can use the `AWS::EC2::KeyPair::KeyName` parameter type to require a valid Amazon EC2 key pair. This way, there is reduced risk that a user will input an incorrect value that results in improper stack behavior.

**AWS System Manager parameter types** AWS *Systems Manager parameter types* can reference parameters that exist in the AWS Systems Manager Parameter Store. If you specify a parameter key, AWS CloudFormation will search your Systems Manager Parameter Store for the correct value and input this into the stack. When you perform stack updates, AWS CloudFormation queries the same key again and could result in a new value for the AWS CloudFormation parameter.

## Mappings

You can use the *Mappings* section of a template to create a rudimentary lookup tables that you can reference in other sections of your template when you create the stack.

A common example of mappings usage is to look up Amazon EC2 instance AMI IDs based on the region and architecture type. Note in the following example that mappings entries may contain only string values. (*Mappings* does not support parameters, conditions, or intrinsic functions.)

```
"Mappings" : {
 "RegionMap" : {
 "us-east-1" : { "32" : "ami-6411e20d", "64" : "ami-7a11e213" },
 "us-west-1" : { "32" : "ami-c9c7978c", "64" : "ami-cfc7978a" },
 "eu-west-1" : { "32" : "ami-37c2f643", "64" : "ami-31c2f645" },
 "ap-southeast-1" : { "32" : "ami-66f28c34", "64" : "ami-60f28c32" },
 "ap-northeast-1" : { "32" : "ami-9c03a89d", "64" : "ami-a003a8a1" }
 }
}
```

After you declare the *Mappings* section, you can query the values within the mapping with the `Fn::FindInMap` intrinsic function. The example shows an `Fn::FindInMap` call that queries the AMI ID based on region and architecture type (32- or 64-bit). If the region was `us-east-1`, for example, the previous template snippet would resolve to `ami-6411e20d`.

### Pseudo Parameter: AWS::Region

The `AWS::Region` reference is a pseudoparameter; that is, it's a parameter that AWS defines automatically on your behalf. The `AWS::Region` parameter, for example, resolves to the region code where the stack is being deployed (such as `us-east-1`).

```
"Resources" : {
 "myEC2Instance" : {
 "Type" : "AWS::EC2::Instance",
 "Properties" : {
 "ImageId" : { "Fn::FindInMap" : ["RegionMap", { "Ref" : "AWS::Region" },
 "32"] },
 "InstanceType" : "m1.small"
 }
 }
}
```

## Conditions

You can use *Conditions* in AWS CloudFormation templates to determine when to create a resource or when a property of a resource is defined (either in the *Resources* or *Outputs* section of the stack). Conditional statements make use of intrinsic functions to evaluate multiple inputs against one other.

A common use case for this would be to conditionally set an Amazon EC2 instance to use a larger instance type if the environment to which you deploy is prod versus dev. The environment type is input as a template parameter, EnvType, which the conditional statement, CreateProdResources, uses. The conditional statement decides whether to create an additional Amazon Elastic Block Store (Amazon EBS) volume and mount it to the instance with the Condition property of the resource.



A single condition can reference input parameters, mappings, or other conditions to determine whether the final value is true or false.

```
{
 "AWSTemplateFormatVersion" : "2010-09-09",
 "Mappings" : {
 "RegionMap" : {
 "us-east-1" : { "AMI" : "ami-7f418316", "TestAz" : "us-east-1a" },
 "us-west-1" : { "AMI" : "ami-951945d0", "TestAz" : "us-west-1a" },
 "us-west-2" : { "AMI" : "ami-16fd7026", "TestAz" : "us-west-2a" }
 }
 },
 "Parameters" : {
 "EnvType" : {
 "Description" : "Environment type.",
 "Default" : "test",
 "Type" : "String",
 "AllowedValues" : ["prod", "test"]
 }
 },
 "Conditions" : {
 "CreateProdResources" : {"Fn::Equals" : [{"Ref" : "EnvType"}, "prod"]}
 },
 "Resources" : {
 "EC2Instance" : {
 "Type" : "AWS::EC2::Instance",
 "Properties" : {
 "ImageId" : { "Fn::FindInMap" : ["RegionMap", { "Ref" : "AWS::Region" }, "AMI"] }
 }
 },
 "MountPoint" : {
 "Type" : "AWS::EC2::VolumeAttachment",
 "Condition" : "CreateProdResources",
 "Properties" : {
 "Device" : "/dev/sdh",
 "InstanceId" : {"Ref" : "EC2Instance"},
 "VolumeId" : { "Fn::FindInMap" : ["RegionMap", { "Ref" : "AWS::Region" }, "VolumeId"] }
 }
 }
 }
}
```

```
"Properties" : {
 "InstanceId" : { "Ref" : "EC2Instance" },
 "VolumeId" : { "Ref" : "NewVolume" },
 "Device" : "/dev/sdh"
}
},
"NewVolume" : {
 "Type" : "AWS::EC2::Volume",
 "Condition" : "CreateProdResources",
 "Properties" : {
 "Size" : "100",
 "AvailabilityZone" : { "Fn::GetAtt" : ["EC2Instance",
"AvailabilityZone"] }
 }
}
}
```

You can also use Conditions to declare different resource properties based on whether the condition evaluates to true with the Fn::If intrinsic function. The following example uses the UseDBSnapshot condition to determine whether to pass a value to the DBSnapshotIdentifier property of an AWS::RDS::DBInstance resource. You use the AWS::NoValue pseudoparameter in place of a null value in AWS CloudFormation templates. When you provide it as a value to a resource property, AWS::NoValue removes that property declaration.

```
"MyDB" : {
 "Type" : "AWS::RDS::DBInstance",
 "Properties" : {
 "AllocatedStorage" : "5",
 "DBInstanceClass" : "db.m1.small",
 "Engine" : "MySQL",
 "EngineVersion" : "5.5",
 "MasterUsername" : { "Ref" : "DBUser" },
 "MasterUserPassword" : { "Ref" : "DBPassword" },
 "DBParameterGroupName" : { "Ref" : "MyRDSPParamGroup" },
 "DBSnapshotIdentifier" : {
 "Fn::If" : [
 "UseDBSnapshot",
 {"Ref" : "DBSnapshotName"}, {"Ref" : "AWS::NoValue"}
]
 }
 }
}
```

## Transforms

As templates grow in size and complexity, there may be situations where you use certain components repeatedly across multiple templates, such as common resources or mappings. Transforms allow you to simplify the template authoring process through a powerful set of macros you use to reduce the amount of time spent in the authoring process. AWS CloudFormation transforms first create a change set for the stack. Transforms are applied to the template during the change set creation process.



Once a change set is complete, the template updates with output of the executed macros. The finalized template deploys to AWS CloudFormation, not the original with the transform declarations. This can cause confusion, as the original template will not be available via the console or AWS CLI or AWS SDK actions.

There are two types of supported transforms.

**AWS::Include Transform** AWS::Include Transform acts as a tool to import template snippets from Amazon S3 buckets into the template being developed. When the template is evaluated, a change set is created, and the template snippet is copied from its location and is added to the overall template structure. You can use this transform anywhere in a template, except the Parameters and AWSTemplateFormatVersion sections.

When you use the AWS::Include Transform at the top level of a template, the syntax must match the example. (Note that the transform is declared as Transform.) This is especially useful if there is a set of common mappings that you use across multiple teams or template authors, as they can share this set and update it in one location.

```
{
 "Transform" : {
 "Name" : "AWS::Include",
 "Parameters" : {
 "Location" : "s3://MyAmazonS3BucketName/MyFileName.json"
 }
 }
}
```

When you use a transform in nested sections of a template, such as the Properties section of an AWS::EC2::Instance resource, use the following syntax. (Note that this is now an intrinsic function call.)

```
{
 "Fn::Transform" : {
 "Name" : "AWS::Include",
 "Parameters" : {
 "Location" : "s3://MyAmazonS3BucketName/MyFileName.json"
 }
 }
}
```

When you process stack updates, the template snippets you reference in any transforms pull from their Amazon S3 locations. This means that if a snippet updates without your knowledge, the updated snippet will import into the template. We recommend that you create change sets first so that any accidental updates can be caught before you deploy.



AWS CloudFormation does not support nested transforms. If the snippet being imported into a template includes an additional transform declaration, the stack creation or update will fail.

**AWS::Serverless Transform** You can use the AWS::Serverless Transform to convert AWS Serverless Application Model (AWS SAM) templates to valid AWS CloudFormation templates for deployment. AWS SAM uses an abbreviated template syntax to deploy serverless applications with AWS Lambda, Amazon API Gateway, and Amazon DynamoDB.

The following example creates a function that uses the serverless transform. When AWS CloudFormation evaluates the transform, the transform expands the template to include an AWS Lambda function and its IAM execution role.

```
Transform: AWS::Serverless-2016-10-31
Resources:
 MyServerlessFunctionLogicalID:
 Type: AWS::Serverless::Function
 Properties:
 Handler: index.handler
 Runtime: nodejs4.3
 CodeUri: 's3://testBucket/mySourceCode.zip'
```

## Resources

The *Resources* section of an AWS CloudFormation template declares the actual AWS resources to be provisioned and their properties. *AWS CloudFormation requires this template section when you create stacks.* The Resources section follows a standard syntax, where a logical ID acts as the resource key and type/properties subkeys define the actual type of resource to deploy and what properties it should have.

The *logical ID* of the resource allows it to be referenced in other parts of a template. You can refer to Resources in other sections of a template, build relationships between interdependent resources, output property values of the resources, perform other useful functions. The Resource Type defines the actual type of resource being managed. For example, an Amazon S3 bucket type is AWS::S3::Bucket. There are too many resource types available to list in this book, and they are updated regularly. Check the AWS CloudFormation documentation for available resource types.

<https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-template-resource-type-ref.html>

The resource properties section defines what configuration a resource should have. In the same example, the AWS::S3::Bucket resource has an optional property called BucketName, which defines the name of the bucket to create.

```
{
 "Resources": {
 "MyBucket": {
 "Type": "AWS::S3::Bucket",
 "Properties": {
 "BucketName": "MyBucketName1234"
 }
 }
 }
}
```

Resource properties are either optional or required and may be any of the following types:

- String
- List of strings
- Boolean
- References to parameters or pseudoparameters
- Intrinsic functions

## Outputs

*Outputs* are values that can be made available to use outside a single stack. You can reference these values in a number of different ways, such as cross-stack references, nested stacks, describe-stack API calls, or in the AWS CloudFormation console. Outputs are useful in providing meaningful information after a stack has been created or updated successfully. For example, it would be helpful to output an Elastic Load Balancing load balancer URL to the user when a web application stack deploys successfully.

The basic structure for AWS CloudFormation outputs follows. Similar to resources, outputs must have a logical ID so that AWS CloudFormation can reference them. The Description field provides a friendly explanation of the purpose of the output, which can be useful to users of your template. The value being returned can be produced using intrinsic functions, or it can be a static string value. Lastly, the Export key (optional) creates cross-stack references.

Here is an example of outputting the ELB load balancer URL:

```
"Outputs" : {
 "BackupLoadBalancerDNSName" : {
 "Description": "The DNSName of the backup load balancer",
 "Value" : { "Fn::GetAtt" : ["BackupLoadBalancer", "DNSName"] }
 }
}
```

## Intrinsic Functions

Situations can occur where values input into a template cannot be determined until the stack or change set actually is created. If you create an Amazon RDS instance, which is referenced in a configuration file added to an Amazon EC2 instance in the same template, the actual database connection string cannot be determined until the database instance is created. Other attributes, settings, or values may need to be calculated from several inputs at once.

*Intrinsic functions* aim to resolve this issue by adding dynamic functionality into AWS CloudFormation templates. Multiple intrinsic functions are available to add significant power and flexibility to your templates.

### **Fn::Base64**

The *Fn::Base64* intrinsic function converts an input string into its Base64 equivalent. The primary purpose of this function is to pass instructions written in string format to an Amazon EC2 instance's `UserData` property.

```
{ "Fn::Base64": valueToEncode }
```

### **Fn::Cidr**

When you create Amazon VPCs and subnets, you must provide Classless Inter-Domain Routing (CIDR) blocks to map a group of IP addresses to the resource being created. The *Fn::Cidr* intrinsic function allows you to convert an IP address block, subnet count, and size mask (optional) into valid CIDR notation.

```
{ "Fn::Cidr": [ipBlock, count, sizeMask] }
```

### **Fn::FindInMap**

After you create mappings in AWS CloudFormation, you use the *Fn::FindInMap* intrinsic function to query information stored within the mapping table. Note that mappings have two key levels, and thus top-level and second-level keys must be supplied as inputs, along with the mapping name itself.

```
{ "Fn::FindInMap": ["MapName", "TopLevelKey", "SecondLevelKey"] }
```

Consider the following `Mappings` section. The `Fn::FindInMap` call would return `ami-c9c7978c`.

```
"Mappings" : {
 "RegionMap" : {
 "us-east-1" : { "32" : "ami-6411e20d", "64" : "ami-7a11e213" },
 "us-west-1" : { "32" : "ami-c9c7978c", "64" : "ami-cfc7978a" },
 "eu-west-1" : { "32" : "ami-37c2f643", "64" : "ami-31c2f645" }
 }
}

. . .

{ "Fn::FindInMap" : ["RegionMap", { "Ref" : "AWS::Region" }, "32"] }
```

### Fn::GetAtt

Resources you create in AWS CloudFormation contain information that you can query in other parts of the same template. For example, if you create an IAM role to use when log in to AWS CloudTrail events to Amazon CloudWatch Logs, you must provide the Amazon Resource Name (ARN) of the AWS role to the trail configuration. Since the ARN is not returned when you use the Ref intrinsic function (this returns the role name), you can use Fn::GetAtt to query additional resource properties. In this case, you would be able to use this intrinsic function to determine the ARN of the role.

```
{ "Fn::GetAtt" : ["logicalIDOfResource", "attributeName"] }
```

### Fn::GetAZs

For each AWS region, different availability zones (with different names) are available. The specific availability zones will not always match between different accounts (in fact, two accounts with the same availability zone by name may not use the same physical location). Because of this, it is not easy to determine which availability zones are usable when you create a stack. The Fn::GetAZs intrinsic function returns a list of availability zones for the account in which the stack is being created.

```
{ "Fn::GetAZs" : "region" }
```



Only availability zones where a default subnet exists will be returned by Fn::GetAZ.

To increase flexibility further and remove the need to hard-code a region in the template, you can use the AWS::Region pseudoparameter to return the list of availability zones for the region in which the stack is being created.

```
{ "Fn::GetAZs" : { "Ref": "AWS::Region" } }
```

### Fn::Join

In some situations, string values must be concatenated from multiple input strings, as is the case when building Java Database Connectivity (JDBC) connection strings. AWS CloudFormation supports string concatenation with the Fn::Join intrinsic function. You can join string values with a predefined delimiter, which you supply to the function along with a list of strings to join.

When you define the UserData for an AWS::EC2::Instance resource, it may be required that you add various parameters to commands being run on the instance.

#### Fn::Join Appending Data Dynamically

This example shows how you can use Fn::Join to append various data dynamically to create complex commands.

```
"Resources" : {
 "Ec2Instance" : {
 "Type" : "AWS::EC2::Instance",
```

```
"Properties" : {
 "ImageId" : "ami-12345678",
 "Tags" : [{"Key" : "Role", "Value" : "Test Instance"}],
 "UserData" : { "Fn::Base64" : { "Fn::Join" : ["", [
 "#!/bin/bash -ex", "\n",
 "echo deploying into region: ", { "Ref": "AWS::Region" }, "\n",
 "\n", "yum install ec2-net-utils -y", "\n",
 "ec2ifup eth1", "\n",
 "service httpd start"]] }
 }
}
}
}
```

## Fn::Select

If you pass a list of values into your template, there needs to be a way to select an item from the list based on what position (index) it is in the list. *Fn::Select* allows you to choose an item in a list based on the zero-based index.



The *Fn::Select* intrinsic function does not check for issues such as whether an index is out of bounds or whether the values in a list equal null. You need to verify that the input list does not contain null values and has a known length.

```
{ "Fn::Select" : [index, listOfObjects] }
```

## Fn::Split

Counter to the *Fn::Join* intrinsic function, you use *Fn::Split* to create a list of strings by separating a single string by a known delimiter. You can use *Fn::Select* to access the output list of strings and pass them to an index to select from different substrings.

```
{ "Fn::Split" : ["delimiter", "source string"] }
```

## Fn::Sub

If you need to build an input string with multiple variables determined at runtime, use the *Fn::Sub* function to populate a template string with input variables from a variable map.

This intrinsic function can also use parameters, resources, and resource attributes already present in your template. Note in the following example that two template values are present in the string, but only one mapping value is provided. This is because the

`AWS::AccountId` pseudoparameter will automatically resolve to the account ID where the stack is being created, and `AWS::Region` automatically resolves to the region ID.

```
{
 "Fn::Sub": ["arn:aws:ec2:${AWS::Region}:${AWS::AccountId}:vpc/${vpc}" , {
 "vpc": { "Ref": "MyVPC" }
 }
}
```

## Ref

You will use the `Ref` intrinsic function a lot within your template, especially when multiple resources have dependencies and relationships between one another (such as if you create an `AWS::EC2::VPC` resource with two `AWS::EC2::Subnet` resources). The behavior of the `Ref` function can differ slightly depending on the resource type being referenced. In some cases, such as with `AWS::S3::Bucket` or `AWS::AutoScaling::AutoScalingGroup` resources, you use `Ref` to return the resource name (in this situation, either the bucket or AWS Auto Scaling group name). In other cases, different properties such as the resource ARN or physical ID returns. Make sure to check the documentation for the resource type being referenced to verify what data returns.

```
{ "Ref" : "logicalName" }
```

## Condition Functions

*Condition functions* are special intrinsic functions for which you can optionally create resources or set resource properties, depending on whether the condition evaluates to true or false. Other than `Fn::If`, you must use all other condition functions within the `Conditions` section of a template. The `Fn::If` intrinsic function allows you to pass different data to resource properties depending on the state of the referenced condition.

### FN::AND

Returns true only if all contained conditions evaluate to true; otherwise, false returns.

```
"Fn::And": [{condition}, {...}]
```

### FN::EQUALS

Returns true if both compared values are equal; otherwise, false returns.

```
"Fn::Equals" : ["value_1", "value_2"]
```

### FN::IF

Returns one of two values, depending on whether the specified condition evaluates to true or false. If you would like to return a null value, pass a reference the `AWS::NoValue` pseudoparameter with the `Ref` intrinsic function.

```
"Fn::If": [condition_name, value_if_true, value_if_false]
```

### FN::NOT

Acts as a negation, returning the opposite of the evaluated condition.

```
"Fn::Not": [{condition}]
```

**Fn::Or**

Returns true if any of the provided conditions are true. Otherwise, false returns.

```
"Fn::Or": [{condition}, {...}]
```

## Built-in Metadata Keys

This section details built-in metadata keys for AWS::CloudFormation::Init, AWS::CloudFormation::Interface, and AWS::CloudFormation::Designer.

### AWS::CloudFormation::Init

This section defines what operations the `cfn-init` helper script performs on Amazon EC2 instances being provisioned by AWS CloudFormation (either as stand-alone instances or in AWS Auto Scaling groups). This metadata key allows you to develop a more declarative infrastructure configuration, instead of having to procedurally script every individual action (such as installing packages, which can vary based on the instance's operating system).

This Metadata section is organized by config keys, which contain a list of configurations to apply.

#### AWS::CloudFormation::Init: Resource Metadata

Unless otherwise specified, AWS CloudFormation will look for config wherever the AWS::CloudFormation::Init Metadata section appears.

```
"Resources": {
 "MyInstance": {
 "Type": "AWS::EC2::Instance",
 "Metadata" : {
 "AWS::CloudFormation::Init" : {
 "config" : {
 "packages" : { },
 "groups" : { },
 "users" : { },
 "sources" : { },
 "files" : { },
 "commands" : { },
 "services" : { }
 }
 }
 },
 "Properties": { }
 }
}
```

## PACKAGES

The packages key allows installation of arbitrary packages on the system. Packages must be available to one of the supported package managers (yum, apt, python, and others). Packages nest under the supported package manager and include a package name followed by an optional version string (or list of versions). If you do not provide a version, the version installs. If the package is not available in the package manager repository, you must include a download URL.

```
"packages": {
 "rpm" : {
 "epel" : "http://download.fedoraproject.org/pub/epel/5/i386/
epel-release-5-4.noarch.rpm"
 },
 "yum" : {
 "httpd" : [],
 "php" : [],
 "wordpress" : []
 }
}
```



On Windows systems, the packages key only supports MSI installers.

## GROUPS

Use the groups key to generate Linux/UNIX groups on the target system. The name of the group is derived from the key name, and you can provide an optional group ID. For example, you create two groups with the following syntax. The first group, groupOne, randomly generates the gid value. The second group, groupTwo, will be assigned a gid of 45.

```
"groups" : {
 "groupOne" : {},
 "groupTwo" : { "gid" : "45" }
}
```



Windows systems do not support the groups key.

## USERS

The users key allows you to create Linux/UNIX users on your instance. By default, users you create with this key are noninteractive system users, and their default shell is set to /sbin/nologon. If you want to modify this behavior, you will have to issue a separate command on the system after the user generates.

```
"users" : {
 "myUser" : {
 "groups" : ["groupOne", "groupTwo"],
 "uid" : "50",
 "homeDir" : "/tmp"
 }
}
```



Windows systems do not support the users key.

## SOURCES

Similar in operation to the files key, you use the sources key to download files from remote locations. However, the sources key supports unpacking archives into target directories on the instance. For example, to download and unpack an archive hosted in a public Amazon S3 bucket, use this snippet:

```
"sources" : {
 "/etc/myapp" : "https://s3.amazonaws.com/mybucket/myapp.tar.gz"
}
```

## FILES

The files key creates files based on either inline content in the template or content from a remote location (URL). An example of inline file content written to /tmp/setup.mysql is as follows:

```
"files" : {
 "/tmp/setup.mysql" : {
 "content" : { "Fn::Join" : ["", [
 "CREATE DATABASE ", { "Ref" : "DBName" }, ";\\n",
 "CREATE USER '", { "Ref" : "DBUsername" }, "'@'localhost' IDENTIFIED BY
 '", { "Ref" : "DBPassword" }, "'\\n",
 "GRANT ALL ON ", { "Ref" : "DBName" }, ".* TO '", { "Ref" : "DBUsername" }
 ",'@'localhost';\\n",
 "FLUSH PRIVILEGES;\\n"
]] },
 "mode" : "000644",
 "owner" : "root",
 "group" : "root"
 }
}
```

Additional file options, such as symlinks and mustache templates, are also supported.

## COMMANDS

The commands key allows the execution of arbitrary commands on an Amazon EC2 instance, such as calling a custom application or script file.

### Command Order of Execution

The commands section processes commands in alphabetical order based on the command name key. In this snippet, the command test would be called before test2.

```
"commands" : {
 "test" : {
 "command" : "echo \\$MAGIC\\ > test.txt",
 "env" : { "MAGIC" : "I come from the environment!" },
 "cwd" : "~",
 "test" : "test ! -e ~/test.txt",
 "ignoreErrors" : "false"
 },
 "test2" : {
 "command" : "echo \\$MAGIC2\\ > test2.txt",
 "env" : { "MAGIC2" : "I come from the environment!" },
 "cwd" : "~",
 "test" : "test ! -e ~/test2.txt",
 "ignoreErrors" : "false"
 }
}
```

## SERVICES

The services key defines which services are enabled or disabled on the instance being configured. Linux systems utilize sysvinit to support the services key, while Windows systems use Windows Service Manager. Additionally, you can configure services to restart when dependencies update, such as files and packages. The following example enables nginx, configures it to run when the instance starts, and restarts whenever /var/www/html updates on the instance.

```
"services" : {
 "sysvinit" : {
 "nginx" : {
 "enabled" : "true",
 "ensureRunning" : "true",
 "files" : ["/etc/nginx/nginx.conf"],
 "sources" : ["/var/www/html"]
 }
 }
}
```

## CONFIGSETS

You can organize config keys into *configSets*, which allow you to call groups of configurations at different times during an instance's setup process and change the order in which configurations are applied. The following example shows two *configSets*, which reverse the order in which configurations execute.

```
"AWS::CloudFormation::Init" : {
 "configSets" : {
 "ascending" : ["config1" , "config2"],
 "descending" : ["config2" , "config1"]
 },
 "config1" : {
 "commands" : {
 "test" : {
 "command" : "echo \\$CFNTEST\\ > test.txt",
 "env" : { "CFNTEST" : "I come from config1." },
 "cwd" : "~"
 }
 }
 },
 "config2" : {
 "commands" : {
 "test" : {
 "command" : "echo \\$CFNTEST\\ > test.txt",
 "env" : { "CFNTEST" : "I come from config2" },
 "cwd" : "~"
 }
 }
 }
}
```

## ENFORCING AWS::CLOUDFORMATION::INIT METADATA

To enforce the `AWS::CloudFormation::Init` metadata, instances being provisioned in your template must call the `cfn-init` helper script as part of `UserData` execution (either in the `AWS::EC2::Instance` `UserData` property or in the same property of an `AWS::AutoScaling::LaunchConfiguration` resource). When doing so, you must provide the stack name and resource logical ID. Optionally, you can execute a `configSet` or list of `configSets` in the call.

You must pass `UserData` to instances in Base64 format. Thus, you call the `Fn::Base64` function to convert the text-based script to a Base64 encoding.

```
"UserData" : { "Fn::Base64" :
 { "Fn::Join" : ["", [
 "#!/bin/bash -xe\\n",
 "aws s3 cp s3://mybucket/test.txt /tmp/test.txt &> /dev/null",
 "cat /tmp/test.txt"
]] }
```

```
"# Install the files and packages from the metadata\n",
"/opt/aws/bin/cfn-init -v ",
" --stack ", { "Ref" : "AWS::StackName" },
" --resource WebServerInstance ",
" --configsets InstallAndRun ",
" --region ", { "Ref" : "AWS::Region" }, "\n"
]]}
}
```

### AWS::CloudFormation::Interface

This section details how to modify the ordering and presentation of parameters in the AWS CloudFormation console. Without this section, parameters display alphabetically without any additional clarification. This is especially useful when providing templates to other groups who are not familiar with the purpose of each input parameter.



This Metadata section is only for the visual appearance of parameters in the AWS CloudFormation console. metadata does not have change templates that you submit via the AWS CLI or AWS SDK.

The AWS::CloudFormation::Interface metadata key uses two child keys, ParameterGroups and ParameterLabels.

```
"Metadata" : {
 "AWS::CloudFormation::Interface" : {
 "ParameterGroups" : [ParameterGroup, ...],
 "ParameterLabels" : ParameterLabel
 }
}
```

### PARAMETERGROUPS

You use the ParameterGroups section to organize sets of parameters into logical groupings, which are then separated by horizontal lines in the console. Each entry in ParameterGroups is defined as an object with a Label key and Parameters key. The Label key contains a friendly text name for each grouping of parameters. The Parameters key contains a list of logical IDs for each parameter in the group.

```
"ParameterGroups" : [
 {
 "Label" : { "default" : "Network Configuration" },
 "Parameters" : ["VPCID", "SubnetId", "SecurityGroupID"]
 },
 {

```

```
 "Label" : { "default": "Amazon EC2 Configuration" },
 "Parameters" : ["InstanceType", "KeyName"]
}
]
```

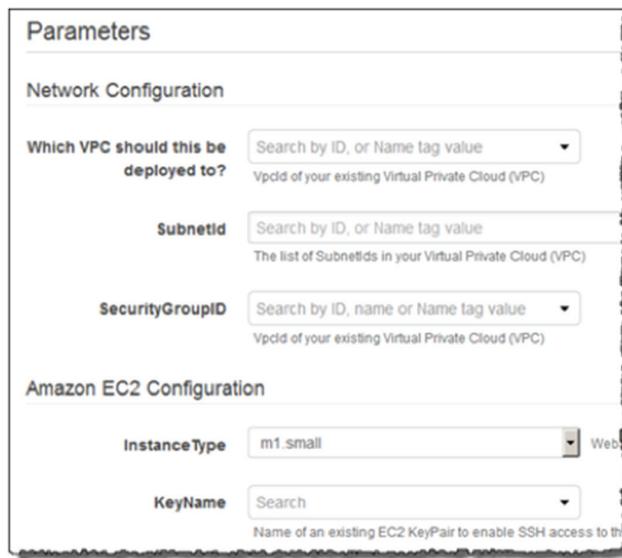
### PARAMETERLABELS

The ParameterLabels section lets you define friendly names for parameters in the console. A logical ID such as BastionSecurityGroupName may be confusing to consumers of your template, especially if the template is shared outside your organization or team. By providing a more human-readable name, template portability is increased. The ParameterLabels key takes a list of parameter logical IDs, each of which has a friendly description as a subkey.

```
"ParameterLabels" : {
 "VPCID" : { "default" : "Which VPC should this be deployed to?" }
}
```

The inclusion of the AWS::CloudFormation::Interface definition results in an easy-to-understand list of parameters that you can complete, as shown in Figure 8.1.

**FIGURE 8.1** AWS CloudFormation parameters



### AWS::CloudFormation::Designer

This Metadata section specifies the visual layout and representation of resources when you design templates in the AWS CloudFormation Designer. Since it is used by Designer, we do not recommend that you manually modify this section.

## AWS CloudFormation Designer

*AWS CloudFormation Designer* is a web-based graphical interface used to design and deploy AWS CloudFormation templates. You can design templates with a drag-and-drop interface of resource objects. You can create connections to make relationships between resources, which automatically update dependencies between them. When you are ready to deploy, you can submit the template directly to AWS CloudFormation or download it in JSON or YAML format.

AWS CloudFormation Designer keeps track of resource positions and relationships with metadata information in `AWS::CloudFormation::Designer`. Since no other service or component uses this information, it is safe to leave as is within your template.

## Custom Resources

Sometimes custom provisioning logic is required when creating resources in AWS. Common examples of this include managing resources not currently supported by AWS CloudFormation, interacting with third-party tools, or other situations where more complexity is involved in the provisioning process.

AWS CloudFormation uses *custom resource providers* to handle the provisioning and configuration of custom resources. Custom resource providers may be AWS Lambda functions or Amazon Simple Notification Service (Amazon SNS) topics. When you create, update, or delete a custom resource, either the AWS Lambda function is invoked or a message is sent to the Amazon SNS topic you configure in the resource declaration.

In the custom resource declaration, you must provide a service token along with any optional input parameters. The *service token* acts as a reference to where custom resource requests are sent. This can be either an AWS Lambda function or Amazon SNS topic. Any input parameters you include are sent with the request body. After the resource provider processes the request, a SUCCESS or FAILED result is sent to the presigned Amazon S3 URL you included in the request body. AWS CloudFormation monitors this bucket location for a response, which it processes once it is sent by the provider. Custom resources can provide outputs back to AWS CloudFormation, which are made accessible as properties of the custom resource. You can access these properties with the `Fn::GetAtt` intrinsic function to pass the logical ID of the resource and the attribute you desire to query.

### AWS Lambda Backed Custom Resources

Custom resources that are backed by AWS Lambda invoke functions whenever create, update, or delete actions are sent to the resource provider. This resource type is incredibly useful to reference other AWS services and resources that may not support AWS CloudFormation. Also, you can use them to look up data from other resources, such as Amazon EC2 instance IDs or entries in an Amazon DynamoDB table.

You can include the AWS Lambda function, which acts as a resource provider in the same AWS CloudFormation template that creates the custom resource and adds additional flexibility for stack update events. In this case, you can define the code for the

AWS Lambda function itself inline in the template or store it in a separate location such as Amazon S3. The following example demonstrates a custom resource, AMIInfo, which makes use of an AWS Lambda function, AMIInfoFunction, as the resource provider. Two additional properties, Region and OSName, provide inputs to the resource provider.

```
"AMIInfo": {
 "Type": "Custom::AMIInfo",
 "Properties": {
 "ServiceToken": { "Fn::GetAtt" : ["AMIInfoFunction", "Arn"] },
 "Region": { "Ref": "AWS::Region" },
 "OSName": { "Ref": "WindowsVersion" }
 }
}
```

For the AWS Lambda function to execute successfully, you must supply it with an IAM role. If the function will interact with other AWS services, you need the following permissions at minimum:

- logs:CreateLogGroup
- logs:CreateLogStream
- logs:PutLogEvents

### Custom Resources Associated with Amazon SNS

Although AWS Lambda functions are incredibly powerful and versatile, they have a limit of 5 minutes of execution time, at which point the function will exit prematurely. This may not be desirable, especially when custom resources take a long time to provision or update. In these situations, use *custom resources associated with Amazon SNS*.

With this resource type, notifications are sent to an Amazon SNS topic any time the custom resource triggers. As the developer you are responsible for managing the system that receives notifications and performs processing. For instance, transcoding of long video files may take a longer time than AWS Lambda allows. In these situations, you subscribe an Amazon EC2 instance to the Amazon SNS topic to listen for requests, consume the input request object, perform the transcoding work, and place an appropriate response.

### Custom Resource Success/Failure

For a custom resource to be successful in AWS CloudFormation, the resource provider must return a success response to the presigned Amazon S3 URL that you provide in the request. If you do not provide a response, the custom resource will eventually time out. This is especially important with regard to update and delete actions. The custom resource provider will need to respond appropriately to every action type (create, update, and delete) for both successful and unsuccessful attempts. If you do not provide a response to an update action, for example, the entire stack update will fail after the custom resource times out, and this results in a stack rollback.

## Resource Relationships

By default, AWS CloudFormation will track most dependencies between resources. There are, however, some exceptions to this process. For example, an application server may not function properly until the backend database is up and running. In this case, you can add a `DependsOn` attribute to your template to specify the order of creation. The `DependsOn` attribute specifies that creation of a resource should not begin until another completes. A resource can have a dependency on one or more other resources in a stack. The following code demonstrates that the resource `EC2Instance` has a dependency on `MyDB`, which means the instance resource will not begin creation until the database resource is in a `CREATE_COMPLETE` state.

```
{
 "Resources" : {
 "Ec2Instance" : {
 "Type" : "AWS::EC2::Instance",
 "Properties" : {
 "ImageId" : {
 "Fn::FindInMap" : ["RegionMap", { "Ref" : "AWS::Region" }, "AMI"]
 }
 },
 "DependsOn" : "myDB"
 },
 "myDB" : {
 "Type" : "AWS::RDS::DBInstance",
 "Properties" : {
 "AllocatedStorage" : "5",
 "DBInstanceClass" : "db.m1.small",
 "Engine" : "MySQL",
 "EngineVersion" : "5.5",
 "MasterUsername" : "MyName",
 "MasterUserPassword" : "MyPassword"
 }
 }
 }
}
```

## Creation Policies

There may be situations where a dependency is not enough, such as when you install and configure applications on an instance before you attach it to an elastic load balancer. In this case, you can use a `CreationPolicy`. A `CreationPolicy` instructs AWS CloudFormation not to mark a resource as `CREATE_COMPLETE` until the resource itself signals back to the service.

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

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

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

## Wait Conditions

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



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

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

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



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

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

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

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

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

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

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

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

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

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

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

## Stack Create, Update, and Delete Statuses

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

### **CREATE\_COMPLETE**

The stack has created successfully.

### **CREATE\_IN\_PROGRESS**

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

### **CREATE\_FAILED**

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

### **DELETE\_COMPLETE**

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

### **DELETE\_IN\_PROGRESS**

The stack is currently deleting.

### **DELETE\_FAILED**

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

### **ROLLBACK\_COMPLETE**

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

### **ROLLBACK\_IN\_PROGRESS**

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

### **ROLLBACK\_FAILED**

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

### **UPDATE\_COMPLETE**

The stack has updated successfully.

### **UPDATE\_IN\_PROGRESS**

The stack is currently performing an update.

### **UPDATE\_COMPLETE\_CLEANUP\_IN\_PROGRESS**

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

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

### **UPDATE\_ROLLBACK\_COMPLETE**

If a stack update fails, AWS CloudFormation will attempt to roll the stack back to the last working state. Once complete, the stack will enter the UPDATE\_ROLLBACK\_COMPLETE state.

### **UPDATE\_ROLLBACK\_IN\_PROGRESS**

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

### **UPDATE\_ROLLBACK\_COMPLETE\_CLEANUP\_IN\_PROGRESS**

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

### **UPDATE\_ROLLBACK\_FAILED**

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

## **Stack Updates**

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



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

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

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

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

**TABLE 8.1** AWS CloudFormation Update Types

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



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



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

## Update Policies

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

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

**TABLE 8.2** AWS Auto Scaling Update Types in AWS CloudFormation

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



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

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

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

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

### SuspendProcesses Attribute

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

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

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

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

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

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

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

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

## **Deletion Policies**

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

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

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

```
{
 "AWSTemplateFormatVersion" : "2010-09-09",
 "Resources" : {
 "myS3Bucket" : {
 "Type" : "AWS::S3::Bucket",
 "DeletionPolicy" : "Retain"
 }
 }
}
```

## Exports and Nested Stacks

Since AWS CloudFormation enforces limits on how large templates can grow and how many resources, outputs, and parameters you can declare in one template, situations can arise where you will need to manage more infrastructure than a single stack will allow. There are two approaches to manage relationships between multiple stacks. You use *stack exports* to share information between separate stacks or manage AWS CloudFormation stacks themselves as resources in a “parent” or “master” stack (a *nested stack* relationship).

### Export and Import Stack Outputs

You can export stack output values to import them into other stacks in the same account and region. This allows you to share data that generates in one stack out to other stacks in your account. If, for example, you create a networking infrastructure such as an Amazon VPC in one stack, you can export the IDs of such resources from this stack and import them into others at a later date.

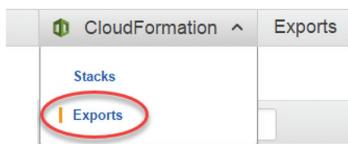
To export a stack value, update the Outputs section to include an Export declaration for every output you want to share.

```
"Outputs" : {
 "Logical ID" : {
 "Description" : "Information about the value",
 "Value" : "Value to return",
 "Export" : {
 "Name" : "Value to export"
 }
 }
}
```



Export values must have a unique name within the AWS account and AWS region.

After you declare the export and the stack creates or updates, it displays in the AWS CloudFormation console on the Exports tab, as shown in Figure 8.2.

**FIGURE 8.2** AWS CloudFormation Exports tab

To import this value into another stack, you use the *Fn::ImportValue* intrinsic function. This intrinsic function requires only the export name as an input parameter (the name present in the AWS CloudFormation console).



You cannot change export values after you import them into another stack. You must first modify the import stack so that it no longer uses the export. To list stacks that import an exported output, use the *ListImports* API action.

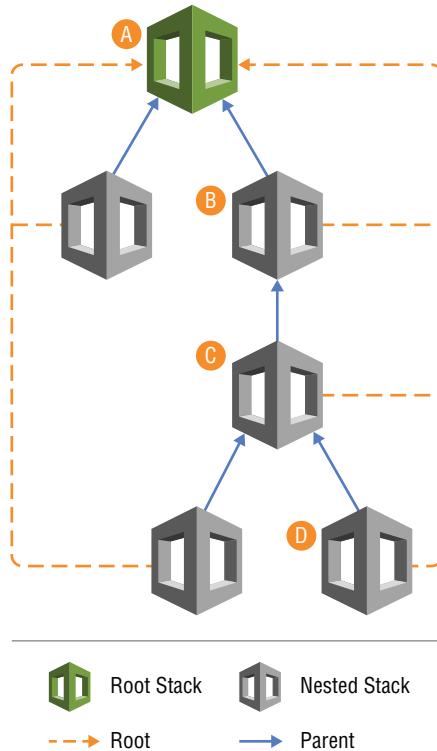
```
https://cloudformation.us-east-1.amazonaws.com/
?Action=ListImports
&ExportName=SampleStack-MyExportedValue
&Version=2010-05-15
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=[Access key ID and scope]
&X-Amz-Date=20160316T233349Z
&X-Amz-SignedHeaders=content-type;host
&X-Amz-Signature=[Signature]
```

### Nesting with the AWS::CloudFormation::Stack Resource

You can manage stacks as resources within the service in AWS CloudFormation. A single parent stack can create one or more AWS::CloudFormation::Stack resources, which act as child stacks that the parent manages. The direct benefits of this are as follows:

- You can work around template limits that AWS CloudFormation imposes.
- It provides the ability to separate resources into logical groups, such as network, database, and web application.
- It lets you separate duties. (Each team is responsible only for maintaining their respective child stack.)

You can increase the nesting levels, as shown in Figure 8.3, with the AWS::CloudFormation::Stack resources.

**FIGURE 8.3** Nested stack structure

From a workflow perspective, the “topmost” parent stack should manage all updates to child stacks. In Figure 8.3, if you need to update stack D, you perform the update on stack A, the topmost parent, to accomplish this.

You can share data from each nested stack if you use a combination of stack outputs and the Fn::GetAtt function calls. If there is an output value from a nested stack that you would like to access from its parent, the following syntax will let you access stack outputs.

```
{ "Fn::GetAtt" : ["logicalNameOfChildStack", "Outputs.attributeName"] }
```



Outputs from stacks created by a nested stack (such as to access outputs in stack C from stack A, as shown in Figure 8.3) can be accessed from the parent stack(s). First, you will need to output the value in the originating stack and then its parent and finally access the output from the parent. To clarify, the output would originate in stack C and be added as an output to stack B, and then stack A references it.

## Stack Policies

Though you can assign resources to create, update, and delete policies to stacks directly, there may be situations where you will want to prevent certain types of updates to stacks themselves. By default, anyone with permissions to modify stacks can perform updates to all underlying stack resources (if they have permissions to modify the resources themselves, or the AWS CloudFormation service role attached to the stack has these permissions). You can assign a *stack policy* to a stack to allow or deny access to modify certain stack resources, which you can filter by the type of update. Stack policies apply to all users, regardless of their IAM permissions.

```
{
 "Statement" : [
 {
 "Effect" : "Allow",
 "Action" : "Update:*",
 "Principal": "*",
 "Resource" : "*"
 },
 {
 "Effect" : "Deny",
 "Action" : "Update:*",
 "Principal": "*",
 "Resource" : "LogicalResourceId/ProductionDatabase"
 }
]
}
```



Stack policies are not a replacement for appropriate access control from an IAM policy. Stack policies are an additional fail-safe to prevent accidental updates to critical resources.

Stack policies protect all resources by default with an implicit deny. To allow access to actions on stack resources, you must apply explicit allow statements to the policy. In the previous example, an explicit allow specifies that you can perform all updates on all resources in the stack. However, the explicit deny for the ProductionDatabase resource prevents update actions to this specific resource. You can specify allow and deny actions for

either resource logical IDs or generic resource types. To specify policies for generic resource types, use a condition statement as follows:

```
{
 "Statement" : [
 {
 "Effect" : "Deny",
 "Principal" : "*",
 "Action" : "Update:*",
 "Resource" : "*",
 "Condition" : {
 "StringEquals" : {
 "ResourceType" : ["AWS::EC2::Instance", "AWS::RDS::DBInstance"]
 }
 }
 }
]
}
```



Once you apply a stack policy, you cannot remove it. During future updates, the policy must be temporarily replaced.

You can allow or deny specific types of updates for resources in your stack. Action types include the following:

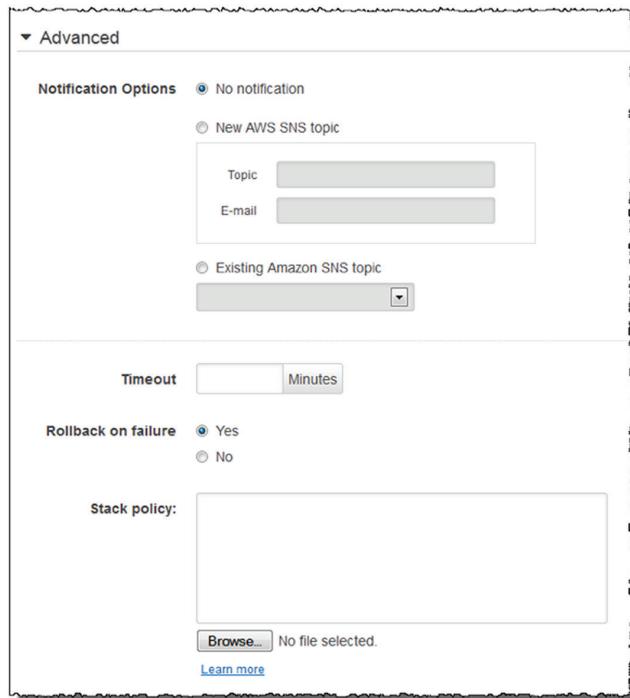
**Update:Modify** Update actions where resources will experience some or no interruption

**Update:Replace** Update actions where replacement resources create (the physical ID of the resource changes)

**Update:Delete** Update actions where resources delete from the stack

**Update:\*** All update actions

Once a stack policy has been set, it will need to be overridden during updates to protected resources. To do so, you supply a new, temporary stack policy. You add this stack policy in the console under the **Stack policy** property, as shown in Figure 8.4.

**FIGURE 8.4** AWS CloudFormation Stack Policy field

When you supply a stack policy during an update, it only modifies the policy for the duration of the update after which the original policy reinstates.

## AWS CloudFormation Command Line Interface

AWS CloudFormation provides several utility functions apart from the standard API-based component of the AWS CloudFormation CLI.

### Packaging Local Dependencies

When you develop templates locally, you may require additional files for your infrastructure that you do not want to define inline as part of the template syntax. For example, you may need to place configuration files on Amazon EC2 instances or AWS Lambda function code. You can use the `aws cloudformation package` command to upload local files and convert local references in your template to Amazon S3 URIs. Consider the following example:

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
```

Resources:

  MyFunction:

    Type: 'AWS::Serverless::Function'

    Properties:

      Handler: index.handler

      Runtime: nodejs4.3

      CodeUri: /home/user/code/lambdafunction

The CodeUri property refers to a local path on the user's workstation (/home/user/code/lambdafunction). To prepare this for deployment, you can run the following command:

```
aws cloudformation package --template /path_to_template/template.json --s3-bucket mybucket --output json > packaged-template.json
```

When you execute this command, the AWS CLI will package the contents of /home/user/code/lambdafunction into a .zip archive and upload it to the Amazon S3 bucket you specify in the --s3-bucket parameter. After doing so, the template updates to refer to the Amazon S3 URI for the archive file and generates the following:

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Resources:
 MyFunction:
 Type: 'AWS::Serverless::Function'
 Properties:
 Handler: index.handler
 Runtime: nodejs4.3
 CodeUri: s3://mybucket/lambdafunction.zip
```

## Deploy Templates with Transforms

Any time that you want to deploy an AWS CloudFormation template that contains transforms, you must first create a change set. The *change set* is responsible for executing the transform to generate a final template that you can deploy. If you would like to reduce this to a one-step process, the aws cloudformation deploy command will generate and execute the change set on your behalf. This is especially useful for rapid testing, as it eliminates the need to approve change sets manually.

When you use this command, you can override default parameters with the --parameter-overrides property.

```
aws cloudformation deploy --template /path_to_template/my-template.json --stack-name my-new-stack --parameter-overrides Key1=Value1 Key2=Value2
```

## AWS CloudFormation Helper Scripts

When you execute custom scripts on Amazon EC2 instances as part of your `UserData`, AWS CloudFormation provides several important helper scripts. You can use these to interact with the stack to query metadata, notify a `CreationPolicy` or `WaitCondition`, and process scripts when AWS CloudFormation detects metadata updates.

### cfn-init

You use this helper script to read `AWS::CloudFormation::Init` metadata from the `AWS::EC2::LaunchConfiguration` or `AWS::EC2::Instance` resource being declared. It is responsible for installing packages, adding files, creating users and groups, and any other configuration you specify in your `AWS::CloudFormation::Init` metadata.

`AWS::CloudFormation::Init` metadata is not enforced automatically. You must call the `cfn-init` helper script in your instances' `UserData`. The following example demonstrates a `cfn-init` call on an instance in an AWS CloudFormation stack. In this case, the `InstallAndRun` configuration set executes on the instance.

```
"UserData" : { "Fn::Base64" :
 { "Fn::Join" : ["", [
 "#!/bin/bash -xe\n",
 "# Install the files and packages from the metadata\n",
 "/opt/aws/bin/cfn-init -v ",
 " --stack ", { "Ref" : "AWS::StackName" },
 " --resource WebServerInstance ",
 " --configsets InstallAndRun ",
 " --region ", { "Ref" : "AWS::Region" }, "\n"
]]
}
```

### cfn-signal

After `cfn-init` has been called and the `AWS::CloudFormation::Init` metadata has been enforced successfully (or unsuccessfully), you can use `cfn-signal` to notify AWS CloudFormation that the instance has completed its configuration. For example, if your template contains a `CreationPolicy` or `WaitCondition` to prevent the setup of an `AWS::ElasticLoadBalancing::LoadBalancer` resource until instances in your `AWS::AutoScaling::AutoScalingGroup` have configured a custom application, `cfn-signal` performs the notification. The following `UserData` example demonstrates how to pass the result of `cfn-init` to `cfn-signal`:

```
"UserData": {
 "Fn::Base64": {
 "Fn::Join": [
 "", [
 "#!/bin/bash -x\n",
 "# Install the files and packages from the metadata\n",
 "# Signal the stack that configuration is complete\n",
 "cfn-signal -e Succeeded", { "Ref" : "AWS::StackName" }
]
]
 }
}
```

```
"/opt/aws/bin/cfn-init -v ",
" --stack ", { "Ref": "AWS::StackName" },
" --resource MyInstance ",
" --region ", { "Ref": "AWS::Region" },
"\n",
"# Signal the status from cfn-init\n",
"/opt/aws/bin/cfn-signal -e $? ",
" --stack ", { "Ref": "AWS::StackName" },
" --resource MyInstance ",
" --region ", { "Ref": "AWS::Region" },
"\n"
]
]
}
}
```

### cfn-get-metadata

If your template contains arbitrary metadata, use `cfn-get-metadata` to fetch this information for use on your instance(s). You can use this helper script to query either an entire metadata block or a subtree. AWS CloudFormation supports only top-level keys.

### cfn-hup

Since AWS CloudFormation executes `UserData` only on resource creation, instances will not detect changes to `AWS::CloudFormation::Init` metadata automatically. Unlike other helper scripts, you can configure `cfn-hup` to run as a daemon on instances. This script checks for changes to resource metadata, can execute custom scripts whenever they are detected, and allows you to perform configuration updates on instances in a stack.

The `cfn-hup` helper script requires you to perform several configuration steps before it detects updates.

#### DAEMON CONFIGURATION FILE

You must create the `cfn-hup.conf` configuration file on the instance, and it needs to contain the stack name. You can also use `cfn-hup.conf` to contain AWS credentials the daemon requires, though it can also leverage IAM instance profiles. Here's an example:

```
[main]
stack=<stack-name-or-id>
```

#### HOOKS CONFIGURATION FILE

Whenever AWS CloudFormation detects changes to instance metadata, user-defined actions are called based on settings in the `hooks.conf` configuration file. You can configure hooks to run on one or more resource actions (add, update, or remove) and can execute arbitrary commands. If there are scripts you want to call, you must add the scripts to the instance

before you execute the hook. If you require more than one configuration file, you can add `/etc/cfn/hooks.d/` on Linux instances. The `hooks.conf` file structure is as follows:

```
[hookname]
triggers=post.add or post.update or post.remove
path=Resources.<logicalResourceId> (.Metadata or
.PhysicalResourceId)(.<optionalMetadatapath>)
action=<arbitrary shell command>
runas=<runas user>
```

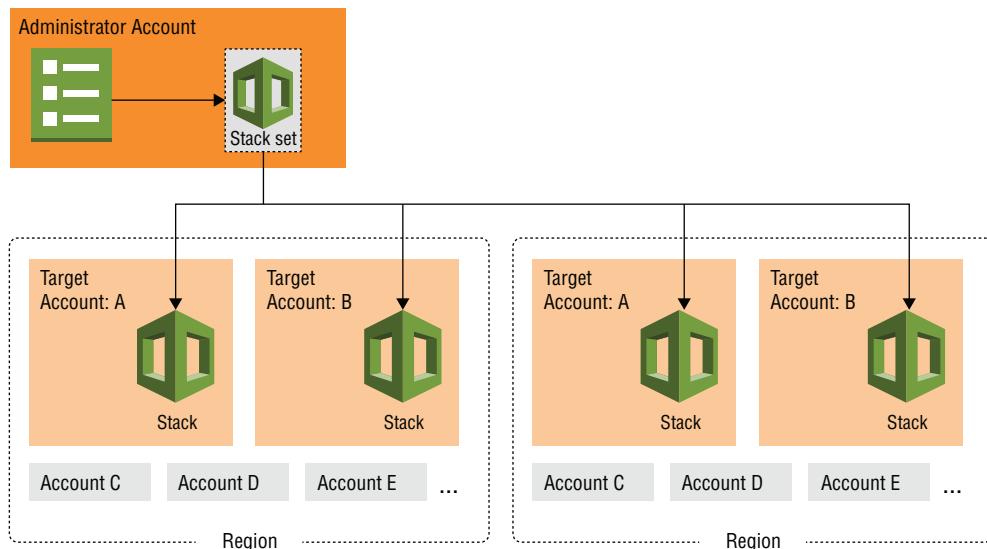
This template snippet demonstrates how to add a `cfn-hup` hook file to instances in an `AWS::AutoScaling::LaunchConfiguration` resource. This hook file will detect updates to the `LaunchConfig` resource and execute the `wordpress_install` config set you specify in the `AWS::CloudFormation::Init` metadata.

```
[hookname]
triggers=post.add or post.update or post.remove
path=Resources.<logicalResourceId> (.Metadata or
.PhysicalResourceId)(.<optionalMetadatapath>)
action=<arbitrary shell command>
runas=<runas user>
"LaunchConfig": {
 "Type" : "AWS::AutoScaling::LaunchConfiguration",
 "Metadata" : {
 "AWS::CloudFormation::Init" : {
 ...
 "/etc/cfn/hooks.d/cfn-auto-reloader.conf": {
 "content": { "Fn::Join": ["", [
 "[cfn-auto-reloader-hook]\n",
 "triggers=post.update\n",
 "path=Resources.LaunchConfig.Metadata.AWS::CloudFormation::Init\n",
 "action=/opt/aws/bin/cfn-init -v ",
 " --stack ", { "Ref" : "AWS::StackName" },
 " --resource LaunchConfig ",
 " --configsets wordpress_install ",
 " --region ", { "Ref" : "AWS::Region" }, "\n",
 "runas=root\n"
]]}},
 "mode" : "000400",
 "owner" : "root",
 "group" : "root"
 }
 }
```

## AWS CloudFormation StackSets

AWS CloudFormation *StackSets* gives users the ability to control, provision, and manage stacks across multiple accounts, as shown in Figure 8.5. From a centralized administrator account, you can develop a template as the basis for provisioning similar stacks across a fleet of accounts.

**FIGURE 8.5** AWS CloudFormation StackSets structure



### Stack Set

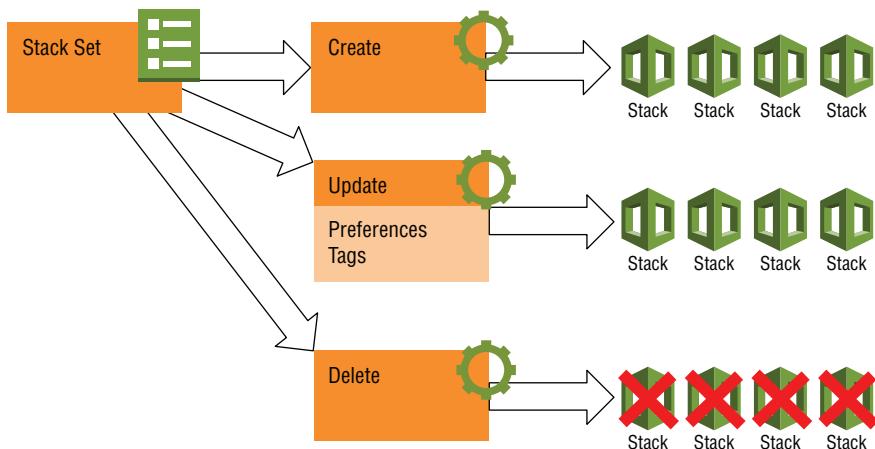
A *stack set* acts as a logical container for stack information in an administrator account. Each stack set will contain information about the stacks you deploy to a single target account in one or more regions. You can configure stack sets to deploy to regions in a specific order and how many unsuccessful deployments are required to fail the entire deployment.



Though a stack set allows you to deploy stacks to multiple regions, the stack set itself exists in one region, and you must manage it there.

### Stack Instance

*Stack instances* allow you to manage stacks in a target account, as shown in Figure 8.6. For example, if a stack set deploys to four regions in a target account, you create four stack instances. An update to a stack set propagates to all stack instances in all accounts and regions.

**FIGURE 8.6** AWS CloudFormation StackSet actions

### Stack Set Operations

When you perform operations on stack sets, you can configure how to control the flow of updates across accounts and regions. You can specify a maximum number or percentage of target accounts for concurrent deployment. Additionally, you can specify a maximum number or percentage of failures (per region). Lastly, you can configure delete operations to remove only the stack instances and stack set and leave the stack itself present in the target account. This option is useful when removing control from an administrator account for resources that need to remain operational in the target account.



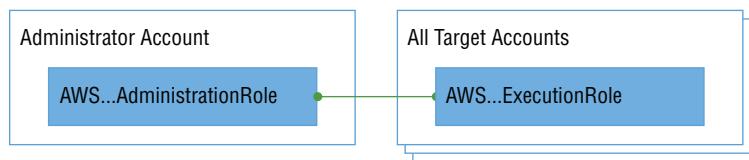
If you specify a maximum number of failures per region, stack updates will not progress to the next region when this threshold is breached. The stack set operation will stop completely.

### Stack Set Permissions

For an administrator account to deploy to any target accounts, you must create a trust relationship between the accounts. To do this, you create an IAM role in each account.

The administrator account requires an IAM service role with permissions to execute stack set operations and assume an execution role in any target accounts. This service role must have a trust policy that allows `cloudformation.amazonaws.com`.

Any target accounts will require an execution role that you create in the administrator account, which the service role can assume. This execution role will require AWS CloudFormation permissions and permissions to manage any resources you define in the template being deployed by the stack set, as shown in Figure 8.7.

**FIGURE 8.7** AWS CloudFormation StackSets permissions

### Target Account Gate

Before you create or update a stack set, evaluate potential blockers in target accounts. If a certain resource type is not available in different regions, for example, this can cause the stack set operation to fail. You can use a *target account gate* to perform evaluation tasks with AWS Lambda functions in the target account. Depending on the return value of the function, the stack set operation will either continue or stop. You can configure this so that account gate failures count toward the stack set's configured tolerance settings.

## AWS CloudFormation Service Limits

Important service limits for AWS CloudFormation are listed in Table 8.3. *You cannot raise template-specific limits through a support request.* You can raise some limits such as the number of stacks per account.

**TABLE 8.3** AWS CloudFormation Service Limits

Limit	Value
Mappings per template	100
Outputs per template	60
Parameters per template	60
Resources per template	200
Stacks per account	200
Template body size	51,200 B (local file) 460,800 B (S3)

## Using AWS CloudFormation with AWS CodePipeline

AWS CloudFormation has built-in integrations with AWS CodePipeline as a deployment provider. Refer to Figure 8.8. When a template revision passes through a pipeline, AWS CloudFormation can reference input parameters, stack policies, and other configuration data in the AWS CodePipeline deployment.

**FIGURE 8.8** CloudFormation as a deployment provider

Create pipeline

Step 1: Name  
Step 2: Source  
Step 3: Build  
**Step 4: Deploy**  
Step 5: Service Role  
Step 6: Review

Deploy

Choose how you deploy to instances. Choose the provider, and then provide the configuration details for that provider.

Deployment provider\* AWS CloudFormation

AWS CloudFormation (1)

Configure your action to create, update CloudFormation stacks or change sets. [Learn more](#)

Action mode\* Create or update a stack

Stack name\* mystack

Template file\* mytemplate.json

Configuration file myconfigfile.json

Capabilities CAPABILITY\_IAM

Role name\* cloudformation\_servicerole

\* Required Cancel Previous Next step

## Deployment Configuration Properties

This section details deployment configuration properties including the following: Action Mode, Stack or Change Set Name, Templates, Template Configurations, Capabilities, Role Names, Output File Names, and Parameter Overrides.

### Action Mode

You can use change sets in a pipeline to include a manual review step to ensure that the changes you deploy are valid and desired before they actually execute. AWS CodePipeline supports the following AWS CloudFormation actions:

- Create or replace a change set
- Create or update a stack
- Delete a stack
- Execute a change set
- Replace a failed stack

### Stack or Change Set Name

These refer to the new or existing stack or change set to be created, updated, or deleted.

## Template

This is the location of the template file to submit. Since AWS CodePipeline uses artifacts to pass files between stages, you must define this file within the artifact with the following:

```
ArtifactName::TemplateFileName
```

## Template Configuration

The template configuration is where you specify properties such as template parameters and the stack policy.



Do not commit sensitive information to your repository. If this file contains information such as passwords, restrict access and pull it into the artifact from another source, such as Amazon S3.

## Capabilities

You must specify any templates which create, update, or delete IAM resources with either the CAPABILITY\_IAM or CAPABILITY\_NAMED\_IAM within this property.

## Role Name

Unlike manually provisioned stacks, AWS CodePipeline requires a service role to assume when you perform actions in AWS CloudFormation.

## Output File Name

This is an optional output that you can add to the output artifact after the deploy action completes. This will add any stack outputs to the pipeline output artifact.

## Parameter Overrides

Though you can define parameters in the template configuration file, the parameter overrides section lets you specify a JSON input file to override any already-specified parameters. You can retrieve data from pipeline artifacts with the Fn::GetParam intrinsic function. The following example demonstrates how to specify a parameter override for ParameterName.

```
{
 "ParameterName" : {
 "Fn::GetParam" : ["ArtifactName", "config-file-name.json", "ParamName"]
 }
}
```



All parameters you specify in the parameter overrides or template configuration file must already exist in the Parameters section of the template you want to deploy.

Parameter overrides can leverage two intrinsic functions specific to AWS CodePipeline. These functions allow you to specify dynamic pipeline values and data from artifacts being passed through the pipeline.

**FN::GETARTIFACTATT**

You use `Fn::GetArtifactAtt` to query values of an input artifact attribute, such as the Amazon S3 bucket name where the artifact is stored. This function enables you to gather information about the artifact itself, not data within the artifact.

When you run a pipeline, AWS CodePipeline copies and writes files to the pipeline's artifact store (Amazon S3 bucket). AWS CodePipeline generates the filenames in the artifact store. These filenames are unknown before you run the pipeline. This attribute requires the Amazon S3 bucket name (`BucketName`), artifact object key (`ObjectKey`), and artifact URL (`URL`).

Use the following syntax to retrieve an attribute value of an artifact:

```
{ "Fn::GetArtifactAtt" : ["artifactName", "attributeName"] }
```

**FN::GETPARAM**

Complimentary to `Fn::GetArtifactAtt`, the `Fn::GetParam` function allows you to query information within an artifact. Any files in the artifact that you query must be in valid JSON format. For example, you can add outputs from a stack as a JSON file to the pipeline artifact, which you use `Fn::GetParam` to query.

```
{ "Fn::GetParam" : ["artifactName", "JSONFileName", "keyName"] }
```

# Summary

In this chapter, you became familiar with provisioning and managing AWS infrastructure using AWS CloudFormation. AWS CloudFormation allows you to describe an entire enterprise's infrastructure as one or more template files, achieving infrastructure as code (IaC) in an environment.

By leveraging AWS CloudFormation in a deployment pipeline, you can dynamically provision and update infrastructure over time by simply committing code to a Git-based repository (AWS CodeCommit). *You can use AWS CodePipeline to reliably automate complex deployment processes.*

AWS CloudFormation uses a declarative language (JSON or YAML template) to describe, model, and provision all infrastructure resources for your applications across all regions and accounts in your cloud environment in an automated and secure manner. This file serves as the single source of truth for your cloud environment. You pay only for the AWS resources you require to run your applications.

The template contains the infrastructure to where AWS will deploy and configuration properties. After you deploy a template in an account, you can redeploy it again in the same or different account and/or region.

A stack is a collection of resources that will be deployed and managed by AWS CloudFormation. When you submit a template, the resources you configure are provisioned and then make up the stack itself. Any modifications to the stack affect underlying resources. Stacks use the IAM user or AWS role authorizations to invoke an action. The template only requires the Resources section.

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

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

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

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

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

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

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

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

By default, AWS CloudFormation will track most dependencies between resources.

*A resource can have a dependency on one or more other resources in a stack, in which case you create a resource relationship to control the order of resource creation, updates, and deletion.*

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

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

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

You use stack exports to share information between separate stacks. Or, you can manage AWS CloudFormation stacks themselves as resources in a nested stack relationship. *You can export stack output values to import them into other stacks in the same account and region.* This allows you to share data that generates in one stack out to other stacks in your account.

You can assign a stack policy to a stack to allow or deny access to modify certain stack resources, which you can filter by the type of update. Stack policies protect all resources by default with an *implicit deny*. To allow access to actions on stack resources, you must apply *explicit allow* statements to the policy.

When you execute custom scripts on Amazon EC2 instances as part of your `UserData`, AWS CloudFormation provides several important helper scripts. You can use these to interact with the stack to query metadata, notify a `CreationPolicy` or `WaitCondition`, and process scripts when AWS CloudFormation detects metadata updates.

AWS CloudFormation *StackSets* give users the ability to control, provision, and manage stacks across multiple accounts and regions. A stack set as a logical container for stack information in an administrator account. Each stack set will contain information about stacks that you deploy to a single target account in one or more regions. Stack instances allow you to manage stacks in a target account. An update to a stack set propagates to all stack instances in all accounts and regions. When you perform operations on stack sets, you can configure how to control the flow of updates across accounts and regions. The administrator account requires an IAM service role with permissions to execute stack set operations and assume an execution role in any target account(s).

You can use a target account gate to perform evaluation tasks with AWS Lambda functions in the target account.

You cannot raise AWS CloudFormation template-specific limits through a support request. You can raise some limits, such as the number of stacks per account.

AWS CloudFormation has built-in integrations with AWS CodePipeline as a deployment provider. When a template revision passes through a pipeline, AWS CloudFormation can reference input parameters, stack policies, and other configuration data in the AWS CodePipeline deployment.

You can use change sets in a pipeline to include a manual review step to ensure that the changes you deploy are valid and desired before they actually execute with the use of the Action Mode.

## Exam Essentials

**Understand Infrastructure as Code (IaC).** You model infrastructure as code to automate the provisioning, maintenance, and retirement of complex infrastructure across an organization. The declarative syntax allows you to describe the resource state you desire, instead of the steps to create it. You can version and maintain IaC with the same development workflow as application and configuration code.

**Understand the purpose of change sets.** Change sets allow administrators to preview the changes that will take place when a given template deploys to the AWS CloudFormation.

This includes a description of resources that you will update or replace entirely. You create change sets to help prevent stack updates that could accidentally result in the replacement of critical resources, such as databases.

**Know the AWS CloudFormation permissions model.** When you create, update, or delete stacks, AWS CloudFormation will operate with the same permissions as the IAM user or IAM role that performs the stack action. For example, a user who deletes a stack that contains an Amazon EC2 instance must also have the ability to terminate instances; otherwise, the stack delete fails. AWS CloudFormation also supports service roles, which you can pass to the service when you perform stack actions. This requires that the user or role have permissions to pass the service role to perform the stack action.

**Know the AWS CloudFormation template structure.** You can use these AWS CloudFormation template properties: AWSTemplateFormatVersion, Description, Metadata, Parameters, Mappings, Conditions, Transform, Resources, and Outputs. Templates only require the Resources property, and you must define at least one resource in every template.

**Know how to use the intrinsic functions.** It is important to understand the AWS CloudFormation templates intrinsic functions.

- Fn::FindInMap
- Fn::GetAtt
- Fn::Join
- Fn::Split
- Ref

**Understand the purpose of AWS::CloudFormation::Init.** This template section defines the configuration tasks the cfn-init helper script will perform on instances that you create individually or as part of AWS Auto Scaling launch configurations. This metadata key allows you to define a more declarative syntax for configuration tasks compared to using procedural steps in the UserData property.

**Know the use cases for both custom resource types.** You can implement custom resources with AWS Lambda functions or Amazon SNS topics. The primary difference between each type is that AWS Lambda-backed custom resources have a maximum execution duration of 5 minutes. This may not work for custom resources that take a long time to provision or update. In those cases, Amazon SNS topics backed by Amazon EC2 instances would allow for long running tasks.

**Understand how AWS CloudFormation manages resource relationships.** AWS CloudFormation will automatically reorder resource provisioning and update steps based on known dependencies. For example, if a template declares an Amazon VPC and a subnet, the subnet will not create before the Amazon VPC (a subnet requires an Amazon VPC ID during creation). However, AWS CloudFormation is not aware of all possible relationships, so you must manually declare them with the DependsOn property. If a template declares an Amazon EC2 instance and AWS DynamoDB table, and the table is referenced inside the instance's UserData property, you must declare a DependsOn property that states the instance depends on the table.

**Understand wait conditions and creation policies.** In some cases, resources in a template should wait for other resources to provision and configure before starting their tasks. For example, you may want to prevent creation of a load balancer resource until instances in an AWS Auto Scaling group have installed a web application. In those cases, you can use either wait conditions or creation policies. Wait conditions require you to add two separate resources to the template (AWS::CloudFormation::WaitCondition and AWS::CloudFormation::WaitConditionHandle). The instance's UserData property references the wait condition handle, where a success or failure signal will be sent. A creation policy does not require the additional resources, and it allows for additional options such as timeouts and signal counts.

**Understand how stack updates affect resources.** When you update a stack, resources may behave differently when properties update. If an Amazon S3 bucket is created as part of a stack and later the bucket policy is updated, the resource will update with no interruption. However, if the bucket name later updates, you must replace the bucket. Resources can undergo one of three types of updates: update with no interruption, update with some interruption, and replace update.

**Know how to use exports and nested stacks to share stacks.** Stack exports allow you to access stack outputs in other stacks in the same region. Exports, however, come with some limitations. For example, you cannot delete stacks that export values until all other stacks that import the exported value have been modified to no longer include the import. Nested stacks make use for the AWS::CloudFormation::Stack resource type. This way, a single stack can create multiple “child” stacks, which can declare their own resources (including other stacks). This is a useful mechanism to work around some service limits such as the number of resources per template (200).

**Understand stack policies.** To prevent updates to critical stack resources, you implement stack policies. A stack policy declares what resources you can and cannot update and under what circumstances. A stack containing an Amazon RDS instance, for example, can include a stack policy that prevents updates that require replacement of the database instance.

## Resources to Review

AWS CloudFormation:

[Infrastructure as Code:](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide>Welcome.html</a></p></div><div data-bbox=)

<https://d1.awsstatic.com/whitepapers/DevOps/infrastructure-as-code.pdf>

AWS Quick Starts:

<https://aws.amazon.com/quickstart/>

Quick Start Builder's Guide:

<https://aws-quickstart.github.io/templates-examples.html>

Bootstrapping Applications via AWS CloudFormation:

<https://s3.amazonaws.com/cloudformation-examples/BoostrappingApplicationsWithAWSCloudFormation.pdf>

AWS CloudFormation Templates:

<https://aws.amazon.com/cloudformation/templates/>

AWS Resource Types Reference:

<https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-template-resource-type-ref.html>

# Exercises

## EXERCISE 8.1

### Write Your Own AWS CloudFormation Template

1. Create an Amazon S3 bucket with a static website configuration that includes references to index and error documents, such as `index.html` and `error.html`, respectively.
2. Create an output to the bucket's website URL.
3. Create an output that displays the bucket name.
4. Upload the code as `index.html` to the root of the bucket for the index document.

```
<html>
 <body>
 <h1>Hello, World!</h1>
 </body>
</html>
```

5. Upload the code as `error.html` to the root of the bucket for the error document.

```
<html>
 <body>
 <h1>Oops! Something went wrong.</h1>
 </body>
</html>
```

6. Access the URL the output provides from your AWS CloudFormation stack to verify the static website works.

**EXERCISE 8.2****Troubleshoot a Failed Stack Deletion**

1. Deploy the AWS CloudFormation code template, which provisions an Amazon S3 bucket in your account.

```
{
 "Resources" : {
 "ExampleBucket" : {
 "Type": "AWS::S3::Bucket"
 }
 },
 "Outputs" : {
 "BucketName" : {
 "Value": { "Ref": "ExampleBucket" }
 }
 }
}
```

2. Upload several files and objects to the Amazon S3 bucket that the template creates.
3. Delete the stack and monitor progress until it fails.

Note the error output by AWS CloudFormation when the stack reaches the DELETE\_FAILED state.

4. Delete all files from the Amazon S3 bucket.
5. Attempt to delete the stack again, but do not enable the option to retain the bucket.

**EXERCISE 8.3****Monitor Stack Update Activity**

1. Deploy the AWS CloudFormation code template, which provisions an Amazon S3 bucket in your account.

```
{
 "Resources" : {
 "ExampleBucket" : {
 "Type": "AWS::S3::Bucket"
 }
 },
 "Outputs" : {
```

```
 "BucketName" : {
 "Value": { "Ref": "ExampleBucket" }
 }
 }
}
```

2. After the stack is created, make note of the output value. This is the name of your Amazon S3 bucket.
3. Use the template code to update the stack, and replace BUCKET\_NAME with a name of your choice.

```
{
 "Resources" : {
 "ExampleBucket" : {
 "Type": "AWS::S3::Bucket",
 "Properties": {
 "BucketName": "BUCKET_NAME"
 }
 }
 },
 "Outputs" : {
 "BucketName" : {
 "Value": { "Ref": "ExampleBucket" }
 }
 }
}
```

Note that a new bucket is created, and the original bucket is deleted. This is because you cannot change bucket names after initial creation, so a replacement must be provisioned.

---

## Review Questions

1. Which of the AWS CloudFormation template sections is/are required?
  - A. AWSTemplateFormatVersion
  - B. Parameters
  - C. Metadata
  - D. Resources
  - E. All of the above
2. You are writing an AWS CloudFormation template and would like to create an output value corresponding to your application's website URL. The application is composed of two application servers in a private subnet behind an Elastic Load Balancing load balancer. The application servers read from the Amazon Relational Database Service (Amazon RDS) database instance. The logical IDs of the instances are AppServerA and AppServerB. The logical IDs of the load balancer and database are AppLB and AppDB, respectively.

```
"Outputs" : {
 "AppEndpoint" : {
 "Description" : "URL to access the application",
 "Value" : "Value to return"
 }
}
```

Which code correctly completes the previous output declaration?

- A. { "Fn::Join": [ "", [ https://, { "Ref": "AppLB" }, "/login.php" ] ] }
- B. { "Fn::Join": [ "", [ https://, { "Fn::GetAtt": [ "AppServerA", "PublicDNSName" ] }, "/login.php" ] ] }
- C. { "Fn::Join": [ "", [ https://, { "Ref": [ "AppLB", "DNSName" ] }, "/login.php" ] ] }
- D. { "Fn::Join": [ "", [ https://, { "Fn::GetAtt": [ "AppDB", "Endpoint.Address" ] }, "/login.php" ] ] }
- E. { "Fn::Join": [ "", [ https://, { "Fn::GetAtt": [ "AppLB", "DNSName" ] }, "/login.php" ] ] }
3. An AWS CloudFormation template you have written uses a `CreationPolicy` to ensure that video transcoding instances launch and configure before the application server instances so that they are available before users are able to access the website. However, you are finding that the stack always reaches the creation policy's timeout value before the transcoding instances complete setup.

Why could this be? (Select THREE.)

- A. The user data script does not include a call to `cfn-signal`.
- B. The instance could not be launched because of account limits.

- C. The user data script fails before reaching the cfn-signal step.
  - D. The instance cannot connect to the AWS CloudFormation endpoint when calling cfn-signal.
4. When you attempt to update an Amazon Relational Database Service (Amazon RDS) instance in your AWS CloudFormation stack, you experience a Resource failed to stabilize error, which causes the stack to roll back any changes you attempted.

What might be the cause of this error, and how could it be resolved?

- A. The database is corrupted and cannot be updated. Take a snapshot of the database, and use it to create a replacement.
  - B. The database took too long to update. Remove the database from the AWS CloudFormation stack by applying a DeletionPolicy of Retain, and manage the stack using the Amazon RDS console or AWS CLI.
  - C. The database took too long to update, and the session credentials used by AWS CloudFormation timed out. Use a service role to perform the update.
  - D. You have attempted to perform an update that is not supported by Amazon RDS. Review the specification documentation and attempt a valid update.
  - E. I/O has not been halted on the database before performing the update, and AWS CloudFormation timed out waiting for database transactions to halt. Temporarily block I/O and attempt the update again.
5. A custom resource associated with AWS Lambda in your stack creates successfully; however, it attempts to update the resource result in the failure message Custom Resource failed to stabilize in the expected time. After you add a service role to extend the timeout duration, the issue still persists.

What may also be the cause of this error?

- A. The custom resource defined a function for handling the CREATE action but did not do the same for the UPDATE action; thus, a success or failure signal was not sent to AWS CloudFormation.
  - B. The service role does not have appropriate permissions to invoke the custom resource function.
  - C. The custom resource function no longer exists.
  - D. All of the above.
6. After you deploy an AWS Serverless Application Model (AWS SAM) template to AWS CloudFormation, can you view the original template? Why or why not?
- A. No, after the template is submitted and the AWS::Serverless transform is executed, an AWS CloudFormation-supported template is generated.
  - B. Yes, the original template is saved and accessible using the get-stack-template AWS CLI command.
  - C. Yes, it is saved in the Amazon Simple Storage Service (Amazon S3) bucket created by AWS CloudFormation for AWS SAM templates.
  - D. No, AWS CloudFormation does not retain processed templates.

7. When defining an AWS Serverless Application Model (AWS SAM) template, how can you create an Amazon API Gateway as part of the stack?
  - A. By defining an `AWS::ApiGateway::RestApi` resource and any associated `AWS::ApiGateway::Method` resources
  - B. One will be created automatically for you whenever `AWS::Serverless::Function` resources are declared with one or more Events.
  - C. By defining an `AWS::Serverless::Api` and providing an inline or external Swagger definition
  - D. `AWS::ApiGateway::RestApi` resources are not supported in AWS SAM templates.
  - E. A, B, and C
8. Which of these helper scripts performs updates to OS configuration when an AWS CloudFormation stack updates?
  - A. `cfn-hup`
  - B. `cfn-init`
  - C. `cfn-signal`
  - D. `cfn-update`
9. Which of these options allows you to specify a required number of signals to mark the resource as `CREATE_COMPLETE`?
  - A. Wait Condition
  - B. Wait Condition Handler
  - C. `CreationPolicy`
  - D. `WaitCount`
10. How would you preview the changes a stack update will make without affecting any resources in your account?
  - A. Create a change set.
  - B. Perform the stack update, and then manually roll back.
  - C. Perform the stack update on a test stack.
  - D. Do a manual diff of both templates.
11. How would you access a property of a resource created in a nested stack?
  - A. This cannot be done.
  - B. In the child stack, declare the resource property as a stack output. In the parent stack, use `Fn::GetAtt` and pass in two parameters, the child stack logical ID and `Outputs.NestedStackOutputName`.
  - C. In the child stack, export the resource property. In the parent stack, import the exported value.
  - D. Use the cross-stack references.

- 12.** By default, with what permissions will AWS CloudFormation stack operations perform?
- A.** Full administrator
  - B.** The permissions of the user performing the operation
  - C.** The AWS CloudFormation service role
  - D.** The AWS CloudFormation does not use permissions
- 13.** An AWS CloudFormation template declares two resources: an AWS Lambda function and an Amazon DynamoDB table. The function code is declared inline as part of the template and references the table. In what order will AWS CloudFormation provision the two resources?
- A.** Amazon DynamoDB table, AWS Lambda function
  - B.** AWS Lambda function, Amazon DynamoDB table
  - C.** This cannot be determined ahead of time.
  - D.** This depends on the template.
- 14.** Which occurs during a replacing update?
- A.** The resource becomes unavailable.
  - B.** The resource physical ID changes.
  - C.** A new resource is created.
  - D.** The original resource is deleted during the cleanup phase.
  - E.** All of the above
- 15.** Which of the update types results in resource downtime? (Select TWO.)
- A.** Update with No Interruption
  - B.** Update with Some Interruption
  - C.** Replacing Update
  - D.** Update with No Data
  - E.** Static Update
- 16.** What must occur before a stack that exports an output can be deleted?
- A.** Any stacks importing the exported value must remove the import.
  - B.** The export must be removed from the stack.
  - C.** Nothing is required.
  - D.** The stack must be deleted.
- 17.** If an AWS CloudFormation stack is in UPDATE\_IN\_PROGRESS state, which of the states are possible transitions? (Select THREE.)
- A.** UPDATE\_ROLLBACK\_COMPLETE
  - B.** UPDATE\_FAILED
  - C.** UPDATE\_ROLLBACK\_FAILED
  - D.** UPDATE\_COMPLETE
  - E.** UPDATE\_COMPLETE\_CLEANUP\_IN\_PROGRESS

- 18.** What does it mean when an AWS CloudFormation stack is in UPDATE\_COMPLETE\_CLEANUP\_IN\_PROGRESS state?
- A.** The stack has failed to update, and it is removing newly created resources.
  - B.** The stack has successfully updated, and it is removing old resources.
  - C.** The stack has successfully updated, and it is removing new resources.
  - D.** The stack has failed to update, and it is removing old resources.
- 19.** Which of the formats are valid for an AWS CloudFormation template? (Select TWO.)
- A.** YAML
  - B.** XML
  - C.** JSON
  - D.** Markdown
  - E.** LaTeX
- 20.** What are some challenges to consider when using the AWS Command Line Interface (AWS CLI) or AWS software development kits (AWS SDKs) to provision and manage infrastructure compared to AWS CloudFormation?
- A.** Reduction of human error
  - B.** Repeatable infrastructure
  - C.** Reduced IAM permissions requirements
  - D.** Versionable infrastructure
  - E.** All of the above
- 21.** What does a service token represent in a custom resource declaration?
- A.** The AWS service that receives the request
  - B.** The Amazon Simple Notification Service (Amazon SNS) or AWS Lambda resource Amazon Resource Name (ARN) that receives the request
  - C.** The on-premises server IP address that receives the request
  - D.** The type of action to take
  - E.** The commands to execute for the custom resource
- 22.** You are creating a custom resource associated with AWS Lambda that will execute several database functions in an Amazon Relational Database Service (Amazon RDS) database instance. As part of this, the functions will return data you would like to use in other resources declared in your AWS CloudFormation template.

How would you best pass this data to the other resources declared in the template?

- A.** Store the data in a JSON file in an Amazon Simple Storage Service (Amazon S3) bucket, and use the AWS Command Line Interface (AWS CLI) to download the object.
- B.** Store the output data in AWS Systems Manager Parameter Store, and query the parameter store using the AWS CLI.
- C.** Use custom resource outputs to declare the returned data as resource properties. Then, query the properties using the Fn::GetAtt intrinsic function.
- D.** This cannot be accomplished.