

AWS Certified Developer – Associate Guide

Second Edition

Your one-stop solution to passing the AWS developer's 2019
(DVA-C01) certification



Packt

www.packt.com

Vipul Tankariya and Bhavin Parmar

AWS Certified Developer –

Associate Guide

Second Edition

Your one-stop solution to passing the AWS developer's
2019 (DVA-C01) certification

Vipul Tankariya
Bhavin Parmar

Packt

BIRMINGHAM - MUMBAI

AWS Certified Developer – Associate Guide

Second Edition

Copyright © 2019 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the authors, nor Packt Publishing or its dealers and distributors, will be held liable for any damages caused or alleged to have been caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

Commissioning Editor: Vijn Boricha

Acquisition Editor: Heramb Bhavsar

Content Development Editor: Abhishek Jadhav

Technical Editor: Swathy Mohan

Copy Editor: Safis Editing

Project Coordinator: Jagdish Prabhu

Proofreader: Safis Editing

Indexer: Priyanka Dhadke

Graphics: Jisha Chirayil

Production Coordinator: Jyoti Chauhan

First published: September 2017

Second edition: May 2019

Production reference: 1310519

Published by Packt Publishing Ltd.

Livery Place

35 Livery Street

Birmingham

B3 2PB, UK.

ISBN 978-1-78961-731-3

www.packtpub.com



mapt.io

Mapt is an online digital library that gives you full access to over 5,000 books and videos, as well as industry leading tools to help you plan your personal development and advance your career. For more information, please visit our website.

Why subscribe?

- Spend less time learning and more time coding with practical eBooks and Videos from over 4,000 industry professionals
- Improve your learning with Skill Plans built especially for you
- Get a free eBook or video every month
- Mapt is fully searchable
- Copy and paste, print, and bookmark content

Packt.com

Did you know that Packt offers eBook versions of every book published, with PDF and ePUB files available? You can upgrade to the eBook version at www.packt.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at customercare@packtpub.com for more details.

At www.packt.com, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on Packt books and eBooks.

Contributors

About the authors

Vipul Tankariya has a broad range of experience in cloud consulting, development, and training. He has worked with a number of customers across the globe, solving real-life business problems in terms of technology and strategy. He is also a public speaker at various AWS events and meetups. He has not only extensively worked on AWS, but is also certified in five AWS certifications. He is an accomplished senior cloud consultant and technologist with more than 21 years of experience. He is focused on strategic thought leadership concentrated around next-generation cloud-based solutions. He has a lot of experience in working on DevOps, CI/CD, and automation at each level of the delivery lifecycle of products, solutions, and services on the cloud.

Bhavin Parmar has a broad range of experience in cloud consulting, development, and training. He actively participates in solving real-life business problems. He has not only extensively worked on AWS, but he is also certified in AWS and Red Hat. This book combines his AWS experience in solving real-life business problems with his hands-on deployment and development experience. Bhavin is an accomplished technologist and senior cloud consultant with more than 11 years of experience. He is focused on strategic thought leadership concentrated around next-generation cloud-based and DevOps solutions. He has also been instrumental in setting up cloud migration strategies for customers, building enterprise-class cloud solutions, and AWS training.

About the reviewers

Amado Gramajo is a passionate technologist with over 15 years of experience working for Fortune 100 companies and leading virtualization, platform migrations, and, currently, cloud migrations. He holds multiple professional and specialty AWS certifications.

Gajanan Chandgadkar has more than 12 years of IT experience. He has spent six years in the US, helping large enterprises architect, migrate, and deploy applications in AWS and Azure. He's been running production workloads on AWS for over six years, and on Azure for the past year. He is a certified solutions architect professional and a certified DevOps professional with seven certifications in trending technologies. Gajanan is also a technology enthusiast who has extensive interest and experience in different topics, such as application development, container technology, and CD. Currently, he is working with Happiest Minds Technologies as a DevOps architect, and has also worked with the Wipro Technologies Corporation in the past.

Adrin Mukherjee is an AWS and Google Cloud-certified architect and loves anything and everything related to cloud computing. He has over 14 years of IT experience and is currently working as a senior architect for a Fortune 500 company.

Packt is searching for authors like you

If you're interested in becoming an author for Packt, please visit authors.packtpub.com and apply today. We have worked with thousands of developers and tech professionals, just like you, to help them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Table of Contents

Preface	<hr/> 1
Chapter 1: Overview of AWS Certified Developer - Associate Certification	8
Frequently asked questions about the exam	11
Chapter 2: Understanding the Fundamentals of Amazon Web Services	13
Examples of cloud services	14
The evolution of cloud computing	16
More about AWS	18
The benefits of using AWS over a traditional data center	18
Comparing AWS cloud and on-premises data centers	19
Total cost of ownership versus return on investment	20
TCO	21
ROI	21
Accessing AWS services	21
An overview of AWS	22
AWS' global infrastructure	22
Regions and AZs	22
What are SaaS, PaaS, and IaaS?	24
Understanding virtualization	25
Virtualization types based on virtualization software	26
Virtualization types based on virtualization methods	26
Elasticity versus scalability	28
Creating a new AWS account	29
AWS' free tier	31
Root user versus non-root user	32
Deleting an AWS account	33
Understanding the AWS dashboard	34
Components of the AWS dashboard	34
Core AWS services	35
AWS compute services	36
AWS storage services	36
AWS database services	37
AWS networking and content delivery services	37
AWS migration services	38
AWS developer tools	38
AWS management tools	39
AWS security, identity, and compliance services	40

AWS analytics services	41
AWS machine learning services	42
AWS IoT services	43
AWS game development services	43
AWS mobile services	43
AWS application integration services	44
AWS desktop and app streaming services	45
AWS business productivity services	45
AWS customer engagement services	46
AWS media services	46
The shared security responsibility model	47
AWS soft limits	49
DR with AWS	50
The backup and restore DR model	52
The pilot light DR model	53
The warm standby DR model	56
The multi-site DR model	57
Summary	59
Chapter 3: Identity and Access Management (IAM)	60
Understanding the AWS root user	61
Elements of IAM	62
Users	62
Access key ID and secret key	64
Password policies	65
Multi-factor authentication	66
Security token-based MFA	67
Steps for enabling a virtual MFA device for a user	67
Creating an AWS IAM user using the AWS dashboard	72
Introducing the AWS CLI	75
Installing the AWS CLI	75
Getting an AWS user access key and secret key	76
Configuring the AWS CLI	76
AWS CLI syntax	77
Getting AWS CLI help	77
Creating an IAM user using the AWS CLI	77
Groups	78
Creating a new IAM group	80
Creating an IAM group using the CLI	81
Adding existing users to a group	81
IAM role	81
Creating roles for an AWS service	86
Creating IAM roles using the AWS CLI	90
Policy	90
Managed policies	92
Inline policies	92
Resource-based policies	93
IAM policy simulator	95

Active Directory Federation Service (AD FS)	96
Integration between AD FS and the AWS console	97
Web identity federation	98
Security Token Service (STS)	100
AWS account ID and alias	101
AWS account IDs	101
AWS account aliases	101
Controlling user access to the AWS Management Console	102
IAM best practices	103
Exam tips	105
Summary	107
Chapter 4: Virtual Private Clouds	109
 Introduction to VPCs	110
Subnets	113
Private subnets	113
Public subnets	114
IP addressing	115
Private IPs	115
Public IPs	116
Elastic IP addresses	118
 Creating a VPC	119
VPCs with a single public subnet	120
VPCs with private and public subnets	126
VPCs with public and private subnets and hardware VPN access	129
VPCs with a private subnet only and hardware VPN access	132
 Security	135
Security groups	137
NACLs	139
Security groups versus NACLs	140
Flow logs	140
Controlling access	142
 VPC networking components	143
ENI	143
Route tables	144
IGWs	146
Egress-only IGWs	147
NATs	148
Comparison of NAT instances and NAT gateways	151
DHCP option sets	152
DNS	153
VPC peering	154
VPC endpoints	155
ClassicLink	156
 VPC best practices	157
 Summary	159

Chapter 5: Getting Started with Elastic Compute Cloud (EC2)	161
Introducing EC2	162
Pricing for EC2	163
On-demand	164
Spot instances	164
Reserved instances	165
Scheduled reserved instances	166
Dedicated hosts	167
EC2 instance life cycle	168
Instance launch	169
Instance stop and start	169
Instance reboot	170
Instance retirement	170
Instance termination	170
Amazon Machine Images (AMIs)	172
Root device types	174
EC2 instance virtualization types	175
Creating an EC2 instance	176
Changing the EC2 instance type	182
Connecting to the EC2 instance	183
Connecting to a Linux EC2 instance from a Microsoft Windows system	184
Converting a PEM file to a private key (PPK)	185
Connecting to an EC2 instance using a PuTTY session	186
Troubleshooting SSH connection issues	189
EC2 instance metadata and user data	189
Placement groups	191
Introducing EBS	191
Types of EBS	192
General Purpose SSD (gp2)	192
Provisioned IOPS SSD (io1)	193
Throughput optimized HDD (st1)	193
Cold HDD (sc1)	194
Encrypted EBS	194
Monitoring EBS volumes with CloudWatch	195
Snapshots	196
EBS-optimized EC2 instances	198
EC2 best practices	198
Summary	200
Chapter 6: Handling Application Traffic with ELB	202
Introducing ELB	203
Benefits of using ELB	204
Types of ELB	205
Classic Load Balancer	205
Creating a Classic Load Balancer	205
Application Load Balancer	212
Network Load Balancer	212

Features of ELB	212
How ELB works	220
The working of the Classic Load Balancer	221
The working of the Application Load Balancer	223
ELB best practices	225
Summary	225
Chapter 7: Monitoring with CloudWatch	227
Introducing CloudWatch	227
How Amazon CloudWatch works	228
Elements of Amazon CloudWatch	229
Namespaces	229
Metrics	230
Dimensions	231
Statistics	232
Percentile	233
Alarms	233
Creating a CloudWatch alarm	234
Billing alerts	241
CloudWatch dashboards	243
Monitoring types – basic and detailed	243
CloudWatch best practices	246
Summary	247
Chapter 8: Simple Storage Service, Glacier, and CloudFront	249
Introducing Amazon S3	252
Creating a bucket	256
Bucket restrictions and limitations	261
Bucket access control	261
Bucket policy	262
User policies	263
Transfer Acceleration	264
Enabling Transfer Acceleration	265
Requester Pays model	266
Enabling Requester Pays on a bucket	266
Understanding objects	267
Object keys	268
Object key naming guide	268
Object metadata	271
System-defined metadata	271
User-defined metadata	273
Versioning	273
Enabling versioning on a bucket	274
Object tagging	275
S3 storage classes	276
S3 Standard storage	276

S3-IA storage	277
S3 One Zone-IA	278
S3 RRS	278
S3 Intelligent-Tiering	278
Glacier	279
Comparison of S3 storage classes and Glacier	280
Life cycle management	281
Life cycle configuration use cases	281
Defining a life cycle policy for a bucket	282
Hosting a static website on S3	288
Cross-origin resource sharing (CORS)	290
Using CORS in different scenarios	290
Configuring CORS on a bucket	291
Enabling CORS on a bucket	292
Cross-region replication	293
Enabling cross-region replication	294
CloudFront	299
CloudFront regional edge caches	301
Setting up CloudFront content and delivery	302
Summary	304
Chapter 9: Other AWS Storage Options	306
Storage and backup services provided by AWS	307
Amazon EFS	312
AWS Storage Gateway	314
File gateways	315
Volume gateways	315
Gateway-cached volumes	316
Gateway-stored volumes	317
Tape-based storage solutions	319
VTL	319
AWS Snowball	322
AWS Snowmobile	323
Summary	324
Chapter 10: AWS Relational Database Service	326
Introducing RDS	327
Amazon RDS components	328
DB instances	328
Regions and AZs	329
Security groups	329
DB parameter groups	330
DB option groups	330
RDS engine types	330
Amazon Aurora DB	331
Comparing Amazon RDS Aurora to Amazon RDS MySQL	334

MariaDB	335
Microsoft SQL Server	336
MySQL	337
Oracle	339
PostgreSQL	340
Creating an Amazon RDS MySQL DB instance	341
Monitoring RDS instances	349
Creating a snapshot	351
Restoring a DB from a snapshot	352
Changing an RDS instance type	353
Amazon RDS and VPC	355
Amazon RDS and high availability	355
Connecting to an Amazon RDS DB instance	356
Connecting to an Amazon Aurora DB cluster	357
Connecting to a MariaDB instance	357
Connecting to a MySQL instance	358
Connecting to an Oracle instance	358
RDS best practices	359
Summary	359
Chapter 11: AWS DynamoDB - A NoSQL Database Service	361
Understanding RDBMSes	362
Understanding SQL	363
Understanding NoSQL	363
Key-value pair databases	364
Document databases	364
Graph databases	365
Wide column databases	365
Using NoSQL databases	366
SQL versus NoSQL	366
Introducing DynamoDB	367
DynamoDB components	367
Primary key	369
Secondary indexes	370
DynamoDB Streams	371
Read consistency model	373
Naming rules and data types	374
Naming rules	374
Data types	375
Scalar data types	376
Document types	377
Set types	378
Creating a DynamoDB table	378
Adding a sort key while creating a DynamoDB table	380
Using advanced settings while creating a DynamoDB table	381

Creating secondary indexes	381
Read/write capacity mode	383
Provisioned capacity	383
Auto Scaling	383
Encryption at rest	385
Methods of accessing DynamoDB	385
DynamoDB console	385
DynamoDB CLI	386
Working with APIs	388
DynamoDB provisioned throughput	389
Read capacity units	389
Write capacity units	389
Calculating table throughput	390
Examples for understanding throughput calculation	390
Example 1	390
Example 2	391
Example 3	391
Example 4	392
Partitions and data distribution	392
Data distribution – partition key	393
Data distribution – partition key and sort key	395
GSI and LSI	396
The difference between GSIs and LSIs	397
DynamoDB Query	398
Query with AWS CLI	400
DynamoDB Scan	401
Reading an item from a DynamoDB table	402
Writing an item to a DynamoDB table	403
PutItem	403
UpdateItem	404
DeleteItem	405
Conditional writes	405
User authentication and access control	409
Managing policies	410
DynamoDB API permissions	411
DynamoDB best practices	412
Summary	413
Chapter 12: Amazon Simple Queue Service (SQS)	416
Why use SQS?	417
How do queues work?	420
Main features of SQS	421
Types of queues	422
Standard queues and FIFO queues	422
Dead Letter Queue (DLQ)	423

Queue attributes	423
Operations in a queue	425
Creating a queue	425
Sending a message in a queue	429
Viewing/deleting a message from a queue	432
Purging a queue	434
Deleting a queue	436
Subscribing a queue to a topic	437
Adding user permissions to a queue	439
SQS limits	442
Queue monitoring and logging	443
CloudWatch metrics available for SQS	443
Logging SQS API actions	445
SQS security	446
Authentication	446
Server-Side Encryption (SSE)	446
Summary	447
Chapter 13: Simple Notification Service (SNS)	449
Introducing Amazon SNS	450
Amazon SNS fanout	452
Application and system alerts	453
Mobile device push notifications	453
Push emails and text messaging	453
Creating an Amazon SNS topic	454
Subscribing to an SNS topic	458
Publishing a message to an SNS topic	460
Deleting an SNS topic	467
Managing access to Amazon SNS topics	469
When to use access control	469
Key concepts	470
Architectural overview	472
Accessing request evaluation logic	473
Invoking the Lambda function using SNS notifications	475
Sending Amazon SNS messages to Amazon SQS queues	477
Monitoring SNS with CloudWatch	482
SNS best practices	484
Summary	485
Chapter 14: AWS Simple Workflow Service (SWF)	486
When to use Amazon SWF	487
Workflow	487
Example workflow	488
Workflow history	488
How workflow history helps	489

Actors	489
Workflow starter	490
Decider	491
Activity worker	491
Tasks	492
SWF domains	492
Object identifiers	493
Task lists	493
Workflow-execution closure	494
Life cycle of a workflow execution	495
Polling for tasks	498
SWF endpoints	498
Managing access with IAM	499
SWF – IAM policy examples	499
Summary	501
Chapter 15: CloudFormation Overview	503
Understanding templates	504
Understanding a stack	506
The template structure	508
AWSTemplateFormatVersion	510
Description	510
Metadata	511
Parameters	511
AWS-specific parameters	512
Mappings	521
Conditions	523
Transform	526
Resources	527
Outputs	528
A sample CloudFormation template	530
CloudFormer	530
Rolling updates for auto scaling groups	531
CloudFormation best practices	531
Summary	532
Chapter 16: Understanding Elastic Beanstalk	534
Introduction to Elastic Beanstalk	535
Elastic Beanstalk components	536
Elastic Beanstalk environment tiers	537
The web server environment tier	537
The worker environment tier	539
Elastic Beanstalk-supported platforms	540
Creating a web application source bundle	541
Getting started using Elastic Beanstalk	541

Step 1 – signing in to the AWS account	541
Step 2 – creating an application	542
Step 3 – viewing information about the recently created environment	548
Step 4 – deploying a new application version	549
Step 5 – changing the configuration	551
Verifying the changes on the load balancer	553
Step 6 – cleaning up	554
The version life cycle	554
Deploying web applications to Elastic Beanstalk environments	555
Monitoring the web application environment	557
Elastic Beanstalk best practices	558
Summary	559
Chapter 17: Overview of AWS Lambda	561
Introducing AWS Lambda	562
Understanding a Lambda function	562
The Lambda function invocation types	564
Writing a Lambda function	565
A Lambda function handler in Node.js	565
A Lambda function handler in Java	566
A Lambda function handler in Python	567
A Lambda function handler in C#	568
Deploying a Lambda function	569
AWS Lambda function versioning and aliases	570
Environment variables	571
Tagging Lambda functions	572
Lambda functions over VPC	572
Building applications with AWS Lambda	573
Event source mapping for AWS services	573
Event source mapping for AWS stream-based services	575
Event source mapping for custom applications	576
AWS Lambda best practices	577
Summary	578
Chapter 18: Key Management Services	580
Introducing encryption	580
Symmetric encryption	581
Asymmetric encryption	581
How does KMS work?	582
Types of keys	584
Different types of CMKs	586
Creating a CMK	587
Viewing existing keys	592
Modifying existing CMKs	592
Updating the administrators or users of a key	593

Tagging a key	594
Enabling or disabling keys	594
AWS services supported by KMS	595
Summary	596
Chapter 19: Working with AWS Kinesis	597
Kinesis Video Streams	597
The Kinesis Video Streams API	600
The producer API	600
The consumer APIs	601
Kinesis Data Streams	603
Architecture	604
Kinesis Data Streams terminology	605
Kinesis Data Firehose	608
Kinesis Data Firehose – key concepts	609
Kinesis Data Firehose – data flow	610
Kinesis Data Analytics	612
Kinesis Data Analytics for SQL applications	612
Kinesis Data Analytics for Java applications	614
Summary	616
Chapter 20: Working with AWS CodeBuild	618
Introducing AWS CodeBuild	619
Understanding AWS CodeBuild	620
Working with AWS CodeBuild	621
Configuring a build project in AWS CodeBuild	623
Project configuration	624
Source code	625
Environment	625
Additional configuration	631
Buildspec	633
Artifacts	635
Logs	637
Summary	638
Chapter 21: Getting Started with AWS CodeDeploy	640
The need for CodeDeploy	640
Introducing CodeDeploy	641
Components of CodeDeploy	642
Summary	648
Chapter 22: Working with AWS CodePipeline	649
Introducing CodePipeline and workflows	650
AWS CodePipeline usages	651
AWS CodePipeline – a higher-level view	651
A high-level view of the input and output artifacts at each stage of the pipeline	652

AWS CodePipeline concepts	654
CI with AWS CodePipeline	654
Continuous delivery with AWS CodePipeline	654
Working with CodePipeline	655
Summary	659
Chapter 23: CI/CD on AWS	660
Understanding CI/CD	660
CI	661
CD	662
Continuous deployment	663
AWS tools for CI/CD	664
Summary	668
Chapter 24: Serverless Computing	669
Recapping AWS Lambda	670
An overview of API Gateway	670
Things that API Gateway can do for you	671
How API Gateway works	672
Understanding step functions	673
The difference between Step Functions and a Lambda function	674
The difference between Step Functions and SWF	674
How Step functions works	675
Understanding states	675
Commonly used state fields	676
Tasks	677
Creating a state machine	677
Amazon Cognito	679
Cognito user pool	680
Cognito identity pool	681
Common Amazon Cognito applications	682
Amazon Cognito Sync	686
Summary	686
Chapter 25: Amazon Route 53	687
Introduction to Route 53	687
Working with Route 53	688
Hosted zones	692
DNS record types	693
A record type	693
AAAA record type	693
CAA record type	693
CNAME record type	694
MX record type	694
NAPTR record type	695
NS record type	695

Table of Contents

PTR record type	696
SOA record type	696
SPF record type	696
SRV record type	697
TXT record type	697
Routing policies	697
Health checking	699
Summary	701
Chapter 26: ElastiCache Overview	703
Introduction to ElastiCache	703
ElastiCache engine types	705
Amazon ElastiCache for Memcached	705
Amazon ElastiCache for Redis	708
Designing the right cache for your workload	711
Summary	715
Chapter 27: Mock Tests	717
Mock test 1	717
Mock test 2	733
Appendix A: Assessments	751
Mock Test 1	751
Mock Test 2	751
Another Book You May Enjoy	752
Index	754

Preface

This book will focus on the revised version of the AWS Certified Developer Associate exam. The June 2019 version of this exam guide includes all the recent services and offerings from Amazon that benefit developers.

AWS Certified Developer – Associate Guide, Second Edition starts with a quick introduction to AWS and its prerequisites to get you started. Then, this book will describe Identity and Access Management (IAM) and Virtual Private Cloud (VPC). Next, this book will teach you about microservices, serverless architecture, security best practices, advanced deployment methods, and more. Moving ahead, we will take you through AWS DynamoDB (a NoSQL database service), Amazon Simple Queue Service (Amazon SQS) and CloudFormation. Finally, this book will help you to understand Elastic Beanstalk and will also walk you through AWS Lambda.

By the end of this book, we will cover enough topics and tips and tricks, along with mock tests for you to be able to pass the AWS Certified Developer – Associate exam, and go on to develop and manage your applications on the AWS platform.

Who this book is for

This book is for IT professionals and developers looking to clear the 2019 AWS Certified Developer Associate exam. Developers looking to develop and manage their applications on the AWS platform will also find this book useful. No prior AWS experience is needed.

What this book covers

Chapter 1, *Overview of AWS Certified Developer – Associate Certification*, discusses the official blueprint that is published by Amazon for each certification exam. This blueprint explains the scope of the exam, the prerequisites to attend the exam, and the knowledge that is required to successfully complete the exam. This chapter outlines the AWS Certified Developer – Associate exam and highlights the critical aspects, knowledge area, and services covered in the blueprint.

Chapter 2, *Understanding the Fundamentals of Amazon Web Services*, discusses the fundamentals of AWS. The chapter starts with a basic understanding of what a cloud is and takes you through a brief journey of familiarizing you with the basic building blocks of AWS. It highlights some of the critical aspects of how AWS works and provides an overview of AWS' core infrastructure.

Chapter 3, *Identity and Access Management (IAM)*, discusses IAM in detail. IAM is one of the core and most critical services of AWS. IAM provides a very strong backbone to control the security of the user infrastructure. This chapter covers all the critical aspects of the IAM service and provides you with an understanding of how to work with various features and functionalities of the IAM service.

Chapter 4, *Virtual Private Clouds*, discusses VPCs in detail. VPCs form the basic building blocks of networking on the AWS cloud. They enable the user to create a private network on the AWS infrastructure. This chapter explains how you can create a VPC and start building a secure network with a number of components of AWS networking services.

Chapter 5, *Getting Started with Elastic Compute Cloud (EC2)*, discusses EC2, which is a part of AWS' compute services. This chapter describes what EC2 is and how you can start provisioning servers with various Windows and Linux operating system flavors. It also demonstrates how you can connect and work with these servers. At the end of the chapter, you should be able to work and manage EC2 instances and different types of Elastic Block Storage (EBS), which is attached as a volume to EC2 instances.

Chapter 6, *Handling Application Traffic with ELB*, discusses Elastic Load Balancing (ELB) in detail. An ELB is a load balancing service that distributes incoming application traffic across multiple EC2 instances and increases the fault tolerance of an application. This chapter describes how to create an ELB, how an ELB works, and what the critical aspects of the ELB service are.

Chapter 7, *Monitoring with CloudWatch*, discusses Amazon CloudWatch. This is a monitoring service for AWS cloud resources and the applications you run on AWS. This chapter describes how you can use Amazon CloudWatch to collect and track metrics, collect and monitor log files, set alarms, and automatically react to changes in your AWS resources.

Chapter 8, *Simple Storage Service, Glacier, and CloudFront*, provides an explanation of Amazon's Simple Storage Service (S3), Glacier, and CloudFront services. The chapter, after discussing S3, explains the cheaper, archival storage that is Glacier and, finally, takes you through CloudFront – a Content Distribution Network (CDN) service.

Chapter 9, *Other AWS Storage Options*, touches on the AWS Storage Gateway service, which is a network appliance or a server residing at a customer's premises. It provides an overview of AWS Snowball, which is a service that accelerates the transfer of large amounts of data into and out of AWS using physical storage appliances. It also provides a basic understanding of AWS Snowmobile, which is an Exabyte-scale data transfer service used to move extremely large amounts of data to and from AWS.

Chapter 10, *AWS Relational Database Services*, provides an understanding of AWS Relation Database Service (RDS). It explains different types of engines that are supported by AWS RDS and how to efficiently and effectively create and manage RDS instances on the AWS cloud.

Chapter 11, *AWS DynamoDB – A NoSQL Database Service*, explores Amazon DynamoDB, which is a fully-managed NoSQL database service that provides fast and predictable performance with seamless scalability. This chapter describes various components of DynamoDB, along with the best practices to manage it.

Chapter 12, *Amazon Simple Queue Service (SQS)*, examines SQS, which is a distributed message queuing service. This chapter provides understanding of what SQS is, and explains how you can create and manage it with relevant examples.

Chapter 13, *Simple Notification Service (SNS)*, describes SNS, which is a fully-managed messaging service that can be used to send messages, alarms, and notifications from various AWS services (such as Amazon RDS, CloudWatch, and S3) to other AWS services (such as SQS and Lambda).

Chapter 14, *Simple Workflow Service (SWF)*, examines Amazon SWF in detail. SWF is a web service that makes it easy to coordinate work across distributed application components. This chapter provides a basic understanding of SWF, its various components, and how to use them.

Chapter 15, *CloudFormation Overview*, provides an overview of the AWS CloudFormation service. CloudFormation is a service that helps you to model and set up your AWS resources. The CloudFormation template provides a simple and efficient way to manage your resources in the AWS cloud.

Chapter 16, *Understanding Elastic Beanstalk*, discusses Elastic Beanstalk – an orchestration service that makes it easier for developers to quickly deploy and manage applications in the AWS Cloud. This chapter offers an introduction to Elastic Beanstalk and describes how you can create and manage applications using the service.

Chapter 17, *Overview of AWS Lambda*, explores Lambda, which is an event-driven, serverless computing platform. This chapter provides an overview of Lambda and describes how it runs code in response to events and automatically manages the compute resources that are required by that code.

Chapter 18, *Key Management Service (KMS)*, describes AWS KMS in detail. KMS is a scalable encryption and key management service that is provided by Amazon. As the name suggests, KMS can be used for encrypting data and managing encryption keys. This chapter introduces you to KMS and explains how you can use it with other AWS services.

Chapter 19, *Working with AWS Kinesis*, discusses Kinesis—an easy-to-use real-time data collection, processing, analysis, and data streaming service.

Chapter 20, *Working with AWS CodeBuild*, explores CodeBuild, which is a fully managed build service by Amazon. It can compile and test source code, making it ready to deploy in your projects. This chapter explains what CodeBuild is and demonstrates how you can use it.

Chapter 21, *Getting Started with AWS CodeDeploy*, explores CodeDeploy, which is a fully managed build service by Amazon. It can be used to automate code deployment to any instance on an AWS EC2 or on-premises environment. This chapter introduces you to CodeDeploy and describes how to use CodeDeploy in your development projects.

Chapter 22, *Working with AWS CodePipeline*, describes CodePipeline, which can be used to facilitate Continuous Deployment (CD) on AWS. It can be used to automate the software deployment process, allowing a developer to quickly model, visualize, and deliver code for new feature updates. This chapter introduces you to CodePipeline and describes how you can use it in your development projects.

Chapter 23, *CI/CD on AWS*, examines Continuous Integration (CI) and CD. CI/CD is a mechanism that is used to optimize and automate the development life cycle. This chapter introduces you to CI/CD on AWS and describes how you can use it for your AWS workloads.

Chapter 24, *Serverless Computing*, explores serverless computing. This is a mechanism that is used to run applications without provisioning, maintaining, and administering the computer or storage resources to run the applications. This chapter introduces you to serverless computing, as well as a number of services that AWS provides in order to run your workloads serverlessly.

Chapter 25, *Amazon Route 53*, describes Amazon Route 53, which is a highly-available and scalable cloud Domain Name System (DNS) web service. This chapter introduces you to the service and describes various components of the service.

Chapter 26, *ElastiCache Overview*, examines ElastiCache, which is an AWS service that provides caching mechanisms using Redis and Memcached on the AWS environment. This chapter gives you an overview of ElastiCache and describes how you can use Redis and Memcached engine types.

Chapter 27, *Mock Tests*, contains two mock tests for you to test your knowledge. It tries to cover all the topics that can be expected in the exam in order to challenge your understanding of them. Each test contains 60 questions; you should try to complete each test in 90 minutes.

Chapter 28, *Exploring AWS CodeCommit*, explores CodeCommit in detail. CodeCommit provides a fully managed, scalable, and private Git repository service. Anything from code to binaries can be stored in the CodeCommit repositories. This chapter explains what CodeCommit is and demonstrates how you can use it for creating and managing code repositories in AWS. This chapter is available online at https://www.packtpub.com/sites/default/files/downloads/Exploring_AWS_CodeCommit.pdf.

To get the most out of this book

As the practical examples involve the use of AWS, an AWS account is required.

Download the color images

We also provide a PDF file that has color images of the screenshots/diagrams used in this book. You can download it here: https://www.packtpub.com/sites/default/files/downloads/9781789617313_ColorImages.pdf.

Conventions used

There are a number of text conventions used throughout this book.

CodeInText: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "This specifies the action type, either `Allow` or `Deny` access."

A block of code is set as follows:

```
aws s3api put-bucket-tagging --bucket <Bucket> --tagging  
'TagSet=[{Key=<key>,Value=<value>} ]'
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
<!-- Sample policy -->  
<CORSConfiguration>  
<CORSRule>  
<AllowedOrigin>*</AllowedOrigin>  
<AllowedMethod>GET</AllowedMethod>  
<MaxAgeSeconds>3000</MaxAgeSeconds>  
<AllowedHeader>Authorization</AllowedHeader>  
</CORSRule>  
</CORSConfiguration>
```

Any command-line input or output is written as follows:

```
$ pip install --upgrade --user awscli
```

Bold: Indicates a new term, an important word, or words that you see onscreen. For example, words in menus or dialog boxes appear in the text like this. Here is an example: "Depending on your preference, you can select **Current version** or **Previous versions**, or both, as required."

Warnings or important notes appear like this.



Tips and tricks appear like this.



Get in touch

Feedback from our readers is always welcome.

General feedback: If you have questions about any aspect of this book, mention the book title in the subject of your message and email us at customercare@packtpub.com.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit www.packt.com/submit-errata, selecting your book, clicking on the Errata Submission Form link, and entering the details.

Piracy: If you come across any illegal copies of our works in any form on the Internet, we would be grateful if you would provide us with the location address or website name. Please contact us at copyright@packt.com with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit authors.packtpub.com.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions, we at Packt can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about Packt, please visit packt.com.

1

Overview of AWS Certified Developer - Associate Certification

First of all, congratulations on choosing this book and beginning your journey towards earning AWS Certified Developer – Associate certification. As the saying goes, a good beginning is half done. You have set a target and taken the first step towards it. If you follow the instructions in this book, it will certainly help you complete the certification exam.

To start with, let's first discuss a number of certifications offered by AWS and see where the AWS Certified Developer – Associate certification stands with respect to other AWS certifications.

AWS mainly provides certifications for four roles: AWS Cloud practitioner, architect, developer, and operations. As you can see in the *AWS certification path* section in the following diagram, AWS Cloud Practitioner is a foundation course for the architect, developer, and operations roles. The foundation course is the same for every role-based certification path. The AWS Certified Developer – Associate certification falls under the developer role. The AWS Cloud Practitioner certification is good to have, but it is not mandatory for either of the associate-level certifications. Each of the chapters in this book begins with foundational knowledge of the relevant service and subsequently gets into the details of all the topics that are required to pass the Developer Associate exam.

The following diagram describes the AWS certification path for all the certifications with AWS:

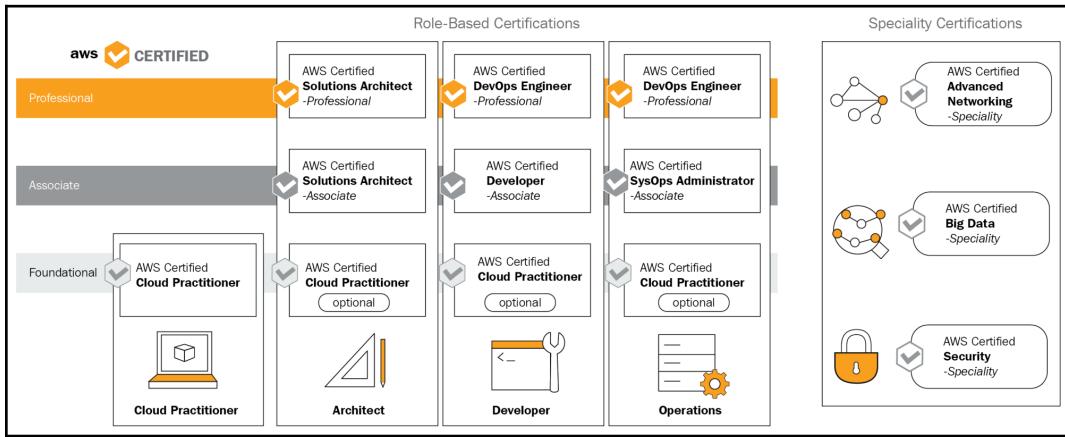


Figure 1.1: AWS certification path

After passing the AWS Certified Solution Architect – Associate exam, you can take the AWS Certified Solution Architect – Professional exam. Similarly, after passing the AWS Certified Developer – Associate exam, you can take the AWS Certified DevOps Engineer – Professional exam. The same goes for the AWS Certified SysOps Administrator – Associate exam. The professional-level path for Developer and SysOps remains the same. AWS also provides specialty certifications for advanced networking, big data, and security. You can take a specialty certification exam only if you have at least one of the role-based certifications, either at associate or professional level.

Each of these certification exams has its own difficulty level. The following diagram indicates the difficulty level of each of the certification exams and highlights where the AWS Certified Developer – Associate exam stands in terms of difficulty:

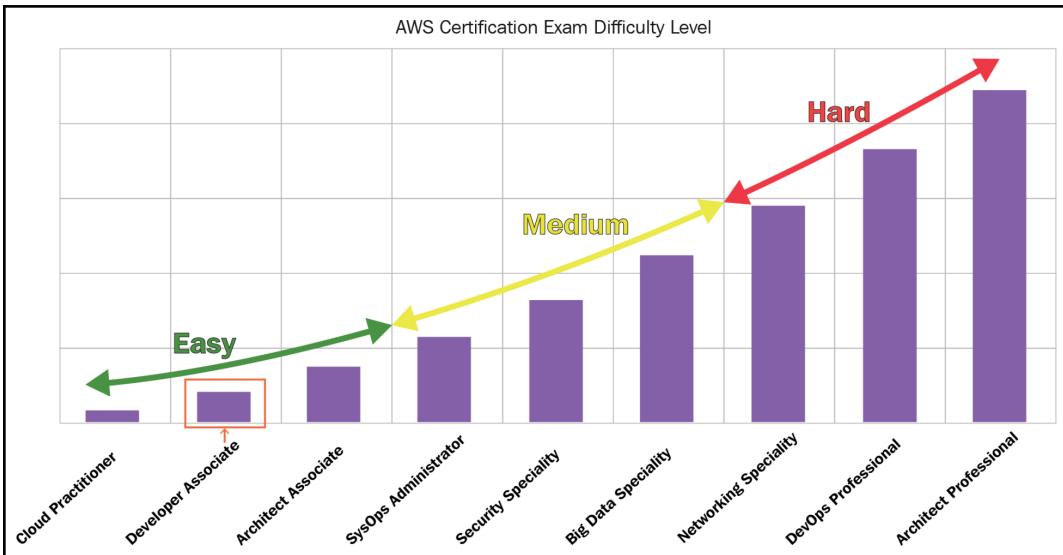


Figure 1.2: AWS certification exam difficulty levels

As you begin your journey towards the certification, you may have a number of questions running through your mind. This chapter covers a number of such questions, the ones that are frequently asked by beginners. Let's see how you should start preparing for the exam.

Amazon publishes an official blueprint for each of its certification exams. The blueprint explains the scope of the exam, the prerequisites for taking the exam, and the knowledge required to pass the exam. The blueprint may change from time to time, and you should look out for the latest copy of the blueprint for the exam from Amazon.

At the time of writing this chapter, the official blueprint for the AWS Certified Developer – Associate exam is available at these URLs:

- **Short URL:** <https://goo.gl/j2dBVY>.
- **Original URL:** https://d1.awsstatic.com/training-and-certification/docs-dev-associate/AWS_Certified_Developer_Associate_Updated_June_2018_Exam_Guide_v1.3.pdf.

The exam scope is divided into five domains, as shown in the following table, with their respective weighting in the exam:

Serial no.	Domain	% weighting in the exam
1	Domain 1: Deployment	22%
2	Domain 2: Security	26%
3	Domain 3: Development with AWS services	30%
4	Domain 4: Refactoring	10%
5	Domain 5: Monitoring and troubleshooting	12%
Total		100%

Frequently asked questions about the exam

The following are the questions that are frequently asked:

1. Are there any prerequisites for the AWS Certified Developer – Associate exam?

There are no prerequisites for the AWS Certified Developer – Associate exam; however, it is recommended that the person preparing for this exam has some prior knowledge or hands-on experience in development.

2. What is the total duration of the exam?

A total of 130 minutes are given to you to complete the exam.

3. How many questions are asked in the exam?

There are around about 60-65 questions in the exam. The number of questions may vary depending on the complexity of the questions asked.

4. What types of question are asked in the exam?

The exam asks multiple-choice questions. It gives a question with multiple answers, and you have to choose one or more right answers from the given list of answers. We have provided some mock tests at the end of the book. You can practice and test your knowledge after you finish reading the book.

5. Where can I register for the exam?

Amazon has partnered with PSI for its certification exams. PSI centers are spread across the globe. You can go to <https://www.aws.training/Certification> and create an account if you do not already have one, or log in with your existing account. After logging in to the site, you can follow the exam registration process on the site to register for the exam at an exam center near you.

6. How much does it cost to register for the exam?

There are two types of exam: **practice** and **final**. The associate-level practice exam costs \$20, and the final exam costs \$150.

7. How should I prepare for the exam?

Here's what is recommended to prepare for the exam:

- Carefully read all the chapters in this book.
- Follow all the tips and tricks in the book.
- Go through the mock tests at the end of the book.
- Go through the whitepapers mentioned in the blueprint.
- Read the FAQs for each of the services given in the book.

8. What is the passing score for the exam?

You have to score 720 out of a total of 1,000.

9. How should I answer the questions in the exam?

The exam poses scenario-based questions. There may be more than one right answer, but you have to choose the most suitable answer(s) out of the given answers. We suggest using the elimination theory whenever you face difficulties answering a question. Start discarding the wrong answers first. When you start eliminating the wrong answers, you may automatically be able to find the right answer, because the eliminated answers will reduce your confusion. Also, do not spend a lot of time on a question if you do not know the answer. Instead, mark the question for review. The exam interface keeps track of all the questions marked for review, and you can revisit them before submitting the final exam.

2

Understanding the Fundamentals of Amazon Web Services

Clouds, as we know from our childhood, are accumulations of tiny droplets of frozen water crystals that are high in the sky, hovering around our planet. So, what do these clouds do? Well, they provide a service to the residents of planet Earth; that is, they bring us rain. **Something** (clouds) that is **somewhere** (up in the sky) provides us with a service by bringing rain. This same concept can be applied to cloud computing.

In cloud computing, the **something** refers to IT services, such as compute, databases, storage, networks, and security. These services are hosted **somewhere** in a secure place (that is, a data center) and are accessible without us needing to worry or even think about how they are configured and licensed. Thus, cloud computing consists of a host of services, which are hosted in a remote location instead of a local server or personal computer, and they are remotely accessible to us.

The following topics will be covered in this chapter:

- Examples of cloud services
- The evolution of cloud computing
- More about AWS
- The benefits of using AWS over traditional data centers
- Accessing AWS services
- An overview of AWS
- Understanding virtualization
- Elasticity versus scalability
- Comparing AWS cloud and on-premises data centers
- Creating a new AWS account

- Understanding the AWS dashboard
- Core AWS services
- The shared security responsibility model
- AWS soft limits
- DR with AWS

Examples of cloud services

Let's take a look at some simple examples of accessing cloud services.

One example is filling in a registration form and using public email services (such as Gmail, Hotmail, or Yahoo). In this case, we start using a service; we don't worry about how the mail services are configured, how the infrastructure is secured, how the software is licensed, or whether highly qualified staff is available to maintain the infrastructure. We just start using email services by providing a secure password.

Another example could be a mobile phone or an electricity connection at home or the office. We just buy a SIM card from a telecom provider, or buy an electrical connection from a local power company, and we don't need to worry about how the telecom network works, or how power is generated and reaches our home or office. We just use them and pay bills per month for the services that we actually consume.

The AWS cloud can be imagined in the same way as a public email, mobile network, or an electricity-providing company. AWS is a public cloud, where we can fill in a form and start using the cloud services (that is, the IT services). It can be used to host personal, commercial, or enterprise-grade IT infrastructures. Various IT services (such as compute, databases, networks, and storage services) can be used as building blocks to create the desired IT infrastructure of an organization.

At a higher level, clouds are of the following three types:

- **Private cloud:** A private cloud is a host of infrastructure, platform, and application services, located in secure remote facilities, that provide compute, platform, or other IT services on-demand, which are accessible and controlled only by a single specific organization. It is preferred by companies who require a secure and dedicated data center or hosting space. Constant upgrades of staff skills and the data center infrastructure are required. It is generally very costly and time-consuming to maintain a private cloud.

- **Public cloud:** A public cloud is a host of infrastructure, platform, and application services, located in secure remote facilities, that provide compute, platform, or other IT services on-demand on a shared but isolated platform, which is open and accessible to the public for subscription. It is preferred by start-ups, MNCs, government organizations, military organizations, scientific organizations, pharmaceutical companies, and other such organizations that intend to utilize on-demand cloud computing. Cloud computing enables organizations to focus on their actual business rather than periodically getting engaged in upgrading existing IT infrastructure to design cutting-edge solutions to compete with other businesses in the market. In a public cloud, all services are provided on a pay-as-you-go model. Hence, it is easy and economical to try various architectures to test and finalize the optimum solution to accelerate organizational growth. Another important characteristic of a public cloud is its ability to provide a virtually unlimited pool of resources, as and when required, for expanding IT infrastructure for short-or long-term needs.
- **Hybrid cloud:** This is a cloud environment that uses a combination of on-premises, private cloud, and public cloud services to fulfill organizational needs. In this model, a private cloud and/or an on-premises environment can use a public cloud's resources to meet specific resource requirements. Since private data centers have limited resources, these data centers are extended to a third-party service provider's public cloud. Such hybrid models can be used for any reason, such as budgets, unusual requirements, infrastructure constraints, regulatory requirements, or any organizational need.

The evolution of cloud computing

The evolution of the cloud is shown in the following diagram:

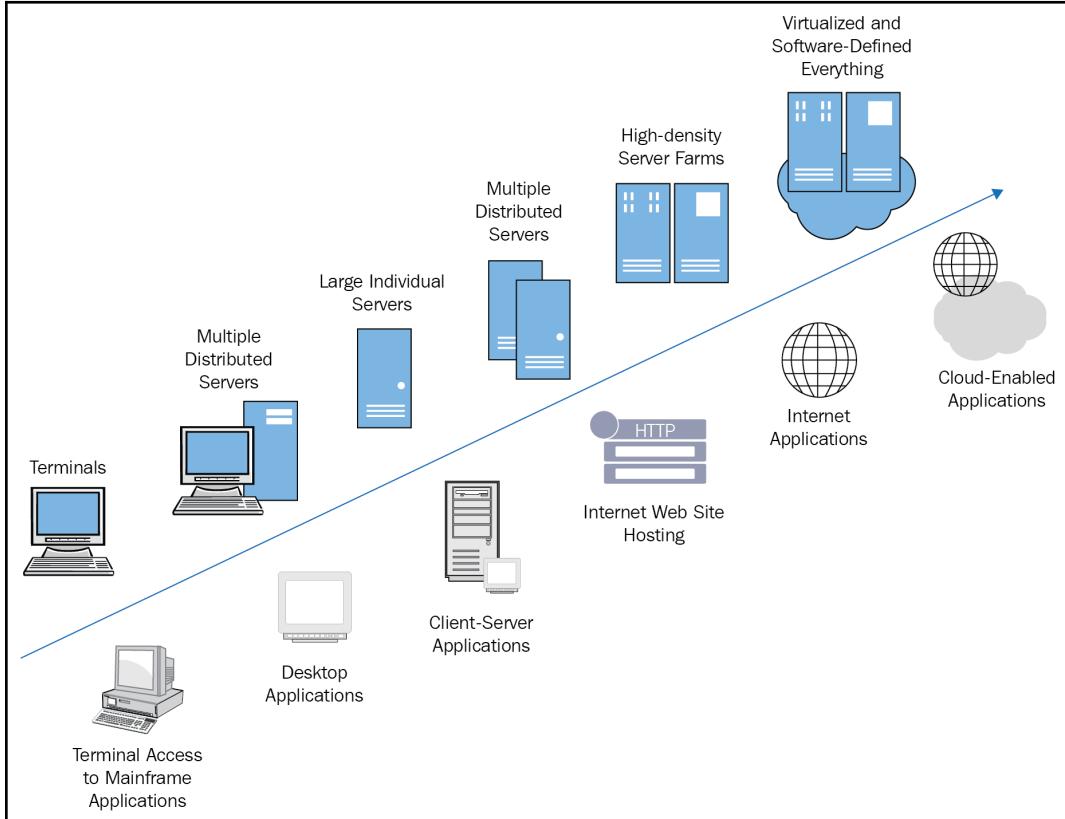


Figure 2.1: The evolution of the cloud

The evolution of the cloud started in the 1950s and concepts such as service-oriented architecture, virtualization, autonomic, and utility computing are the stepping stones of today's cloud computing:

1. In the 1950s, mainframe computers were shared among various users through dumb terminals to save costs and enable the efficient use of resources.

2. In the 1970s, **virtual machines (VMs)** were developed to overcome the disadvantages of earlier technologies. VMs enabled us to run more than one different **operating system (OS)** simultaneously in isolated environments, providing all essential resources (such as CPU, disk controllers, RAM, and NICs) individually to all VMs.
3. In the 1990s, telecom companies started dedicated point-to-point data circuits called **virtual private networks (VPNs)**. These were offered at a fraction of the cost of the then available technologies. This invention made it possible to utilize bandwidth optimally. A VPN made it possible to provide shared access on the same physical infrastructure to multiple users in shared but isolated environments.
4. In 1997, Professor Ramnath Chellappa defined cloud computing.
5. In 1999, Salesforce (<https://www.salesforce.com>) started delivering enterprise-level application services over the internet. This was one of the major moves in cloud history.
6. In the early 2000s, Amazon introduced web-based retail services on its modernized data centers. While Amazon was hardly using 10% of its data center capacity, they realized that new cloud computing infrastructure models could make them more efficient and cost-effective.
7. In the late 2000s, Google introduced Google Docs services directly to end users. This gave a taste of cloud computing and document sharing to end users.
8. In 2006, Amazon formally launched **Elastic Compute Cloud (EC2)** and **Simple Storage Service (S3)**. Subsequently, over the years, Amazon released various cloud services under the name of AWS.
9. In 2008, Google announced the launch of its App Engine services as a beta service. This was the beginning of Google Cloud services.
10. In 2010, Microsoft Azure was formally released, followed by a number of cloud services in subsequent years. In the same year, Rackspace and NASA jointly announced an open source cloud software initiative known as **OpenStack**.
11. In 2011, IBM announced the launch of IBM Smart Cloud and SmartCloud Enterprise Services.
12. In 2012, Oracle announced the launch of Oracle Cloud with **Infrastructure as a Service (IaaS)**, **Platform as a Service (PaaS)**, and **Software as a Service (SaaS)** offerings.
13. In 2017, the Chinese conglomerate, Alibaba, announced the launch of AliCloud.

More about AWS

AWS is a public cloud service; it provides a range of IT services that can be used as building blocks for creating cutting-edge, robust, and scalable enterprise-grade solutions. It can be used to host anything from simple static websites to complex three-tier architectures, from scientific applications to modern **Enterprise Resource Planning (ERP)**, and from online training to live broadcasting events, such as sports events, political elections, and more.

According to Gartner's **Magic Quadrant (MQ)**, which was published in April 2018, AWS is a leader in cloud IaaS. AWS is way ahead of its competitors after pioneering the cloud IaaS market in 2006. The Magic Quadrant image and more details can be found at <https://pages.awscloud.com/gartner-2018-cloud-IaaS.html>.

The MQ is a series of market research reports published by Gartner—the United States-based research and advisory firm. It aims to provide qualitative analysis to a market, including its direction, maturity, and participants. Gartner's reports and MQs are respected in industries worldwide.

The benefits of using AWS over a traditional data center

There are significant benefits of using AWS over a traditional data center, and some of them are listed here:

- **Switching from Capital Expenditure (CapEx) to Operational Expenditure (OpEx):** There is no need to bear the huge upfront cost of purchasing hardware or software and making a CapEx provision in the budget for procuring the same. With AWS, you only pay for what services you use on a monthly basis as OpEx.
- **The cost benefit of massive economies of scale:** Since AWS purchases everything in bulk, this gives them a cost advantage. AWS passes on the benefit of this cost advantage to their customers by offering services at a low cost. As the AWS cloud becomes larger and larger, these massive economies of scale benefit AWS, as well as end customers.

- **There is no need to guess the required infrastructure capacity:** Most of the time, that is, before actual IT implementation, guessing the IT infrastructure requirement leads to either a scarcity of resources or a waste of resources when actual production begins. AWS makes it possible to scale the environment up or down as needed without guessing the needs of the infrastructure.
- **An increase in speed and agility:** While building an on-premises data center, businesses have to wait to get the desired hardware or software from the vendors for an extended period of time. With AWS, it becomes easier for the business to quickly get started and provision the required infrastructure on AWS immediately, without depending on third-party vendors. They don't need to raise a purchase order or wait for delivery; they just log in to their AWS account and have everything at their disposal.
- **Global access:** AWS has data centers and edge locations across the globe. You can take advantage of AWS' global presence and host your infrastructure near to your target market or at multiple locations across the globe at a very nominal cost.



Almost every IT need of an organization can be satisfied by using AWS services; however, there are still a few limitations, such as mainframe computing, which is not currently supported by AWS.

Comparing AWS cloud and on-premises data centers

Whenever an organization thinks of migrating their infrastructure over to a public cloud, the first question that strikes the organization is cost. AWS provides major advantages over on-premises environments, as there is no upfront cost of using AWS. Thus, there is no CapEx requirement, as AWS works on OpEx. This means that a customer only pays for the actual consumption of AWS resources on a monthly basis.

The following table differentiates cost between an on-premises environment and AWS Cloud in regards to various aspects:

Pricing model	One-time upfront cost		Monthly cost	
	Public cloud	On-premises DC	Public cloud	On-premises DC
Server hardware	0	\$\$	\$\$	0
Network hardware	0	\$\$	0	0
Hardware maintenance	0	\$\$	0	\$
Software OS	0	\$\$	\$	0
Power and cooling	0	\$\$	\$	\$
Data center space	0	\$\$	0	0
Administration	0	\$\$	0	\$\$\$
Storage	0	\$\$	\$\$	0
Network bandwidth	0	\$	\$	\$
Resource management software	0	0	\$	\$
24/7 support	0	0	\$	\$

The cost comparison example based on a number of assumptions



AWS provides different pricing options, where you can choose to make upfront payments in lieu of a higher discount. Pricing models are discussed in [Chapter 5, Getting Started with Elastic Compute Cloud \(EC2\)](#).

Total cost of ownership versus return on investment

There is no doubt that public cloud computing has many advantages over traditional data center concepts; for example, it provides a cutting-edge, secure, and robust platform to host an organization's IT infrastructure. It impacts costs by turning CapEx into OpEx. However, when making an investment in any technology or service, it is important for a business to understand two key aspects: **return on investment (ROI)** and **total cost of ownership (TCO)**. Both of these involve careful and critical analysis. It is very important to find the lowest cost in the long run rather than just the lowest initial cost.

TCO

Deriving TCO not only involves purchase cost and maintenance cost, but it also involves hidden costs, such as operating cost, setup cost, change or reconfiguration cost, upgrade cost, security cost, infrastructure support cost, insurance cost, electricity cost, depreciation, tax savings, and environmental impact.

AWS provides an online TCO calculator at <https://awstcocalculator.com>.

ROI

ROI can be derived using a mathematical formula. Primarily, it can be used to evaluate investments and decide how well a particular investment might perform compared to others. In terms of IT, usually, an enterprise's top-level management or CIO performs such a comparison between owning a data center and using a public cloud.

AWS also provides a cost calculator to find monthly estimated expenses at <https://calculator.s3.amazonaws.com/index.html>.

Accessing AWS services

Users can access AWS services in multiple ways; individual services or the whole infrastructure can be accessed using any of the following means:

- **AWS Management Console:** This is a simple to use, browser-based graphical user interface that customers can use to manage their AWS resources.
- **The AWS Command-Line Interface (CLI):** This is mostly used by system administrators to perform day-to-day administration activities. There are individual sets of commands available for each AWS service.
- **AWS Software Development Kits (SDKs):** AWS helps the user reduce the complexity of coding by providing SDKs for a number of programming languages, including Android, iOS, Java, Python, PHP, .NET, Node.js, Go, and Ruby. These SDKs can be used to create custom applications to meet specific organizational needs.
- **Query APIs:** AWS provides a number of HTTP endpoints. These endpoints can be used to send HTTP requests, such as GET and PUT, in order to acquire the present status and any other information for various AWS resources.



Most AWS services can be accessed by using all of the preceding means. However, some AWS services may not have one or two of the previously mentioned access methods.

An overview of AWS

AWS provides a highly reliable, scalable, low-cost infrastructure platform in the cloud, and powers many businesses in almost 190 countries across the world. The following portion of this chapter provides a high-level overview of the basic AWS concepts that you should try to understand before you start working with AWS services.

AWS' global infrastructure

AWS services are available in multiple locations across the globe. AWS provides these services with their infrastructure spread across multiple geographical locations. The AWS infrastructure is segregated in the form of regions, **availability zones (AZs)**, and edge locations, based on geography. Let's now try and understand some of the basic concepts of the AWS global infrastructure.

Regions and AZs

Each region, as shown in the following table, is a collection of at least two or more AZs. The regions are independent and isolated from each other in order to keep them safe from catastrophic events. Such regions actually correlate with geographical areas, such as Asia, Europe, and North America.

AWS is constantly expanding its network across the globe. You can refer to the latest AWS global infrastructure availability details at <https://aws.amazon.com/about-aws/global-infrastructure/> for more information. As shown in the diagram on their website, the blue circles show regions that are currently supported by AWS, while the red circles show upcoming regions. The following table displays the currently available AWS regions in various geographies across the globe:

Geography	Currently available AWS Region	Number of Availability Zones
US-East	North Virginia	6
	Ohio	3
US-West	North California	3
	Oregon	3
Asia Pacific	Mumbai	2
	Seoul	2
	Singapore	3
	Sydney	3
	Tokyo	4
	Osaka-Local	1
Canada	Canada Central	2
China	Beijing	2
	Ningxia	2
Europe	Frankfurt	3
	Ireland	3
	London	3
	Paris	3
South America	Sao-Paolo	3
AWS GovCloud	US-West	3

Figure 2.2: AWS regions in various geographies

Each AZ, as shown in the following diagram, is separated based on a metropolitan area within a region, but they are internally connected to each other through dedicated low-latency networks within the same region to provide failover architecture:

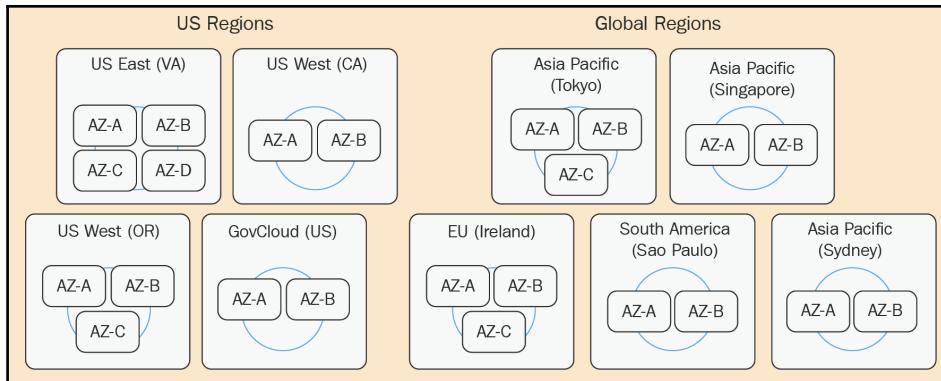


Figure 2.3: An example of the AWS region and AZ configuration

It is highly recommended that you select an AWS region based on the distance to the targeted market or based on legal compliance. For example, if a client's e-commerce website is selling goods and services only in the EU, then it is recommended that you host the website in Frankfurt or Ireland to minimize latency. You should also consider compliance requirements that are specific to a region while deciding on a region for hosting the application infrastructure. For example, if a client is running a website for betting, then it may be illegal in one region, but it could be permitted in another region (that is, in line with the legal compliance requirements of that region). Similarly, many countries' regulatory compliance mandates the hosting of the business data within the country.

AWS constantly evolves its service offerings. New services are launched in specific regions and are then gradually supported in other regions. Due to the gradual approach of AWS in launching a service, there is a chance that not all the services may be available in all regions. It is good practice to review the available services in each region before planning, designing, or proposing any architecture.



Amazon strictly controls, monitors, and audits physical access to AWS data centers by adhering to a number of regulatory requirements.

What are SaaS, PaaS, and IaaS?

Cloud computing is a broad term that covers many services. Common cloud computing models are IaaS, PaaS, and SaaS.

Let's broadly understand these models, as follows:

- **IaaS:** This is when a service provider offers virtualized hardware or computing infrastructure as a service.
- **PaaS:** PaaS is a type of cloud service in which a service provider offers application platforms and tools over the cloud, usually to enable application development. In this service model, the underlying hardware and software is hosted on the service provider's infrastructure.
- **SaaS:** In the SaaS model, the service provider offers software or applications as services. Such services are hosted by the providers and the end customer simply consumes this SaaS without worrying about the underlying hosting platform, infrastructure, and maintenance.

The line of responsibilities in IaaS, PaaS, and SaaS are explained in the following table:

My Software Package	IaaS	PaaS	SaaS
My Application	My Application	My Application	My Application
Data	Data	Data	Data
Runtime	Runtime	Runtime	Runtime
Middleware	Middleware	Middleware	Middleware
Operating Sys	Operating Sys	Operating Sys	Operating Sys
Virtualization	Virtualization	Virtualization	Virtualization
Servers	Servers	Servers	Servers
Storage	Storage	Storage	Storage
Networking	Networking	Networking	Networking

Managed by Me
Managed by Vendor

Figure 2.4: Stack and responsibility separation between the cloud service provider and the customer for IaaS, PaaS, and SaaS

Understanding virtualization

Virtualization is a process of virtually segregating physical hardware resources into a set of virtual resources that can independently work as a computing resource and provide customized and dedicated CPU, RAM, or storage. Each server and its resources are created in an isolated environment. Each isolated environment is abstracted from the physical OS and underlying hardware configuration. Such resources are called VMs or instances.

Virtualization is achieved by using virtualization software that maintains the abstract and virtual layers on top of physical hardware. Let's understand these virtualization software and virtualization types in the following sections.

Virtualization types based on virtualization software

As shown in the following diagram, virtualization software can be broadly categorized into two categories—class 1 and class 2:

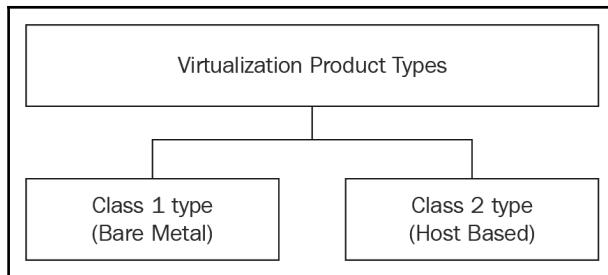


Figure 2.5: The types of virtualization based on the virtualization software

These virtualization production types can be explained as follows:

- **Class 1 type:** This is also known as the bare-metal virtualization type. A very thin (that is, of a small size) virtualization software called a **hypervisor** is installed directly on the physical server. Class 1 hypervisors are faster than class 2 hypervisors. Some examples of class 1 hypervisors are Xen, Hyper-V, KVM (Kernel-based virtual machine), and vSphere. The AWS cloud uses a customized Xen hypervisor and KVM.
- **Class 2 type:** This is also called a hosted hypervisor. These types of hypervisors are installed above the base OS, such as Windows or Linux. Examples of class 2 hypervisors include VMware workstation, VirtualBox, and virtual PC.

Virtualization types based on virtualization methods

As shown in the following diagram, virtualization can also be categorized according to virtualization methods, as follows:

- **OS-level virtualization:** In this type of virtualization, host machines and VMs have the same OS with the same patch level.

- **Software virtualization (hypervisor):** This method of virtualization can be split into three categories, as follows:
 - **Binary translation:** In this virtualization, sensitive instructions from VMs are replaced by hypervisor calls.
 - **Para Virtualization Mode (PVM):** In this virtualization, the guest OS is modified in order to deliver performance.
 - **Hardware Assisted Virtual Machine (HVM):** This method of virtualization creates an abstract layer between the host and guest VMs. It uses special CPU instruction sets (that is, Intel-VT and AMD-V) to boost the performance of guest VMs.
- **Hardware emulation:** This makes it possible to run an unsupported OS, such as running Android on a PC:

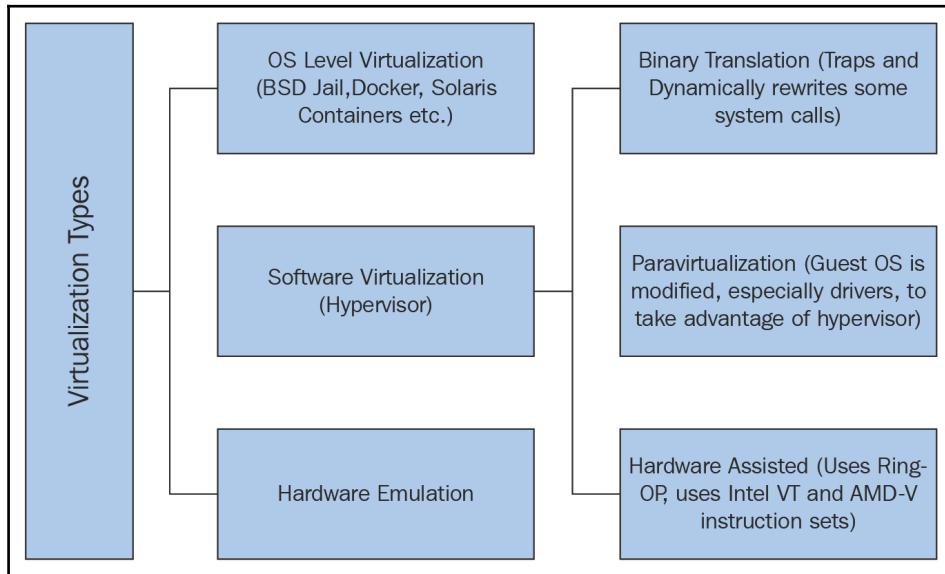


Figure 2.6: The types of virtualization based on virtualization methods

Elasticity versus scalability

Elasticity and scalability are two important characteristics of cloud computing. They describe the way a cloud infrastructure is able to expand and shrink to match the actual dynamic workload, which is described as follows:

- **Scalability:** This means increasing the capacity of an existing instance (that is, scale up) or adding more instances in parallel to an existing instance (that is, scale out). Scalability is essential in order to achieve elasticity:
 - **Scale up:** Changing the instance type from small to large (that is, changing to more memory or compute) is called scaling up; it is also called **vertical scaling**. It may require stopping the existing and running instance. Usually, scaling up is done in order to get more compute and memory on the same instance. Scaling up is usually recommended for applications that do not support clustering modes easily, such as **Relational Database Management Systems (RDBMS)**. Usually, scaling up is achieved manually and requires downtime.
 - **Scale out:** Placing one or more new instances in parallel to the existing instance is called **scale out**. It is also referred to as horizontal scaling. It offers good performance and availability as instances can be placed across multiple AZs. By having individual resources, such as an NIC and disk controller, for each instance, much better performance can be achieved compared to scaling up. Usually, scaling out is suggested for clustering-enabled applications, such as stateless web servers, big data, and NoSQL. Scaling out generally does not require any downtime.
- **Elasticity:** In physics, elasticity can be defined as a material's ability to expand and shrink with external parameters. Similarly, in the cloud infrastructure, elasticity can be defined as the ability to automatically provision additional resources to meet high demand and reduce the extended number of resources when the demand lowers.

Creating a new AWS account

Creating a new account on AWS is as easy as signing up for an email account. The steps to create an AWS account are as follows:

1. In a web browser, open the following URL: <https://aws.amazon.com/>.
2. Click on the **Sign up** button.
3. Fill in the form given in the subsequent screen, and then click on the **Continue** button, as shown in the following screenshot:

The screenshot shows the 'Contact Information' page from the AWS sign-up process. At the top, it says 'Contact Information' and 'All fields are required.' Below that, instructions say 'Please select the account type and complete the fields below with your contact details.' There are two radio buttons for 'Account type': 'Professional' (selected) and 'Personal'. The 'Professional' option is highlighted with a blue border. Below the account type are several input fields: 'Full name' (empty), 'Company name' (empty), 'Phone number' (empty), 'Country/Region' (dropdown menu showing 'United States'), 'Address' (two stacked input fields for street and suite/building), 'City' (empty), 'State / Province or region' (empty), and 'Postal code' (empty). At the bottom, there is a checkbox labeled 'Check here to indicate that you have read and agree to the terms of the [AWS Customer Agreement](#)' followed by a large yellow 'Create Account and Continue' button.

Figure 2.7: Contact information

4. On the subsequent screen, select the account type as either **Personal** or **Professional** and then fill in the contact information. Select the account type as **Professional** if you're going to use this account within your company, educational institution, or organization; otherwise, you can select **Personal**.
5. On the subsequent screen, fill in the payment information, and then click on **Secure submit**.
6. On the subsequent screen, fill in the contact details and verify the CAPTCHA image. The AWS automated system makes a verification call when you click on **Call Me Now**. When you receive an automated verification call, you need to provide a four-digit verification PIN.

Confirm your identity

Before you can use your AWS account, you must verify your phone number. When you continue, the AWS automated system will contact you with a verification code.

How should we send you the verification code?

Text message (SMS) Voice call

Country or region code

India (+91)

Cell Phone Number

Security check

Type the characters as shown above

Send SMS

Figure 2.8: Identity verification

7. In the final step, select the appropriate **Support Plan**. Based on the selected plan, charges are applied to the monthly billing for your account.

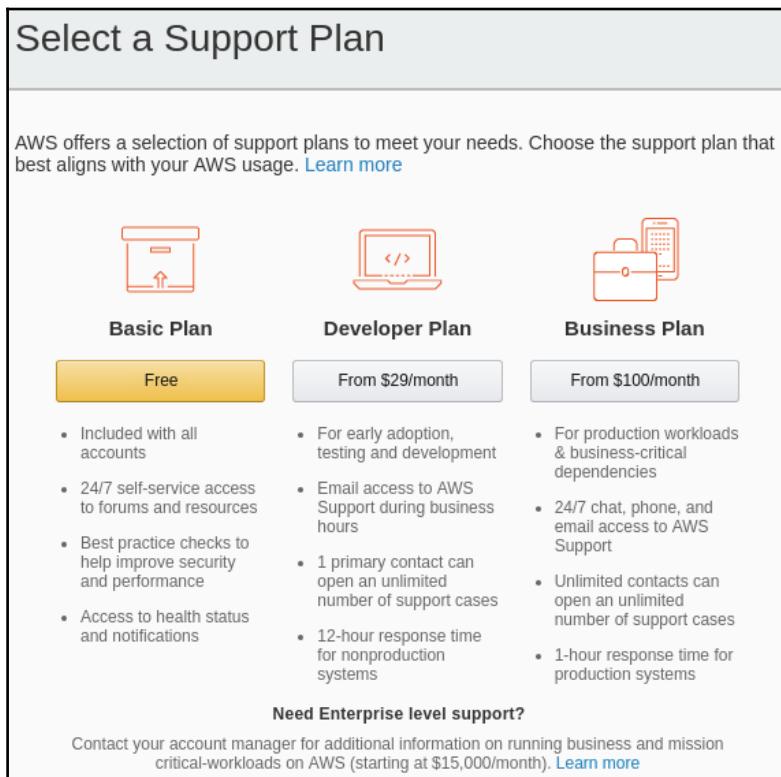


Figure 2.9: Support Plan

AWS' free tier

A new AWS account comes with a limited free tier capacity for 12 months, mostly on all services with certain impediments on utilization. The main purpose of the free tier is to allow users to allow hands-on experience and build their confidence in using AWS services. The latest AWS free tier details can be obtained at <https://aws.amazon.com/free/>.

If AWS resource consumption exceeds the limit of the free tier, then actual charges are applicable and they are billed in a monthly cycle. It is highly recommended that you continuously monitor the running resources on the AWS cloud.

Root user versus non-root user

The email address that is used to create an AWS account is called the root user. This user has the highest privileges. The root user can log in to the AWS account at <https://console.aws.amazon.com/console/home>.

Alternatively, you can go to <https://aws.amazon.com> and select **My Account**; then, select **AWS Management Console** from the drop-down menu. This also brings up the same login screen that was shown in the preceding screenshot.

Root accounts should not be used for day-to-day activities. It is highly recommended that you create appropriate **Identity and Access Management (IAM)** users for day-to-day activities, such as database administration, system administration, and more. IAM users can log in to the AWS dashboard using a different URL, which is called **IAM users sign-in link**.

Here's how you can obtain the IAM user's sign-in link:

1. Sign in to your AWS account using the root user address.
2. Go to the IAM dashboard at <https://console.aws.amazon.com/iam/home>.
3. Copy the **IAM users sign-in link**, which is displayed on the screen.

The **IAM users sign-in link** can be in one of the following formats:

- An `https://<account-number>.signin.aws.amazon.com/console/` URL format, such as <https://123456789012.signin.aws.amazon.com/console/>.
- An `https://<account-name>.signin.aws.amazon.com/console/` URL format, such as <https://packtpub.signin.aws.amazon.com/console/>.

When creating the account, if you provide the account name, then the same account name is used in the aforementioned sign- in URL. If the account name is not given, then you can use a 12-digit account number in the sign- in URL. This account name can be customized from the IAM dashboard.

Deleting an AWS account

AWS provides you with an option to delete or close an AWS account. For closing your account, you need to log in to the AWS account using the root user. To do this, execute the following steps:

1. Go to **My Account** by clicking on your account name, which is given in the top-right corner of the screen. The account name is usually the name given at the time of creating the AWS account.
2. When you click on the account name, a drop-down menu appears.
3. Select **My Account** from the drop-down menu; it opens the account settings page in a new tab. At the bottom of the page, there is an option to close your account.
4. You can select the checkbox under **Close Account** and, finally, click on the **Close Account** button.
5. Be very careful if you're just checking the interface, as all AWS resources and data are wiped out when the account is closed.
6. Once the account is closed, there is no mechanism to undo or get the data back. It is highly recommended that you back up important data to a secure and safe place before closing your AWS account:

▼ Close Account

I understand that by clicking this checkbox, I am willing to close my AWS account. Monthly usage of certain AWS services is calculated and billed at the beginning of the following month. If you have used these types of services this month, then at the beginning of next month you will receive a bill for usage that occurred prior to termination of your account. If you own a Reserved Instance for which you have elected to pay in monthly installments, when your account is closed you will continue to be billed your monthly recurring payment until the Reserved Instance is sold on the Reserved Instance Marketplace or it expires.

Close Account

Figure 2.10: Closing an account

Understanding the AWS dashboard

Having a good understanding of the AWS dashboard is essential for performing development activities. There are a number of components of the AWS dashboard, as shown in the following screenshot. The AWS dashboard layout may change from time to time; an overview of the AWS dashboard can be viewed as follows:

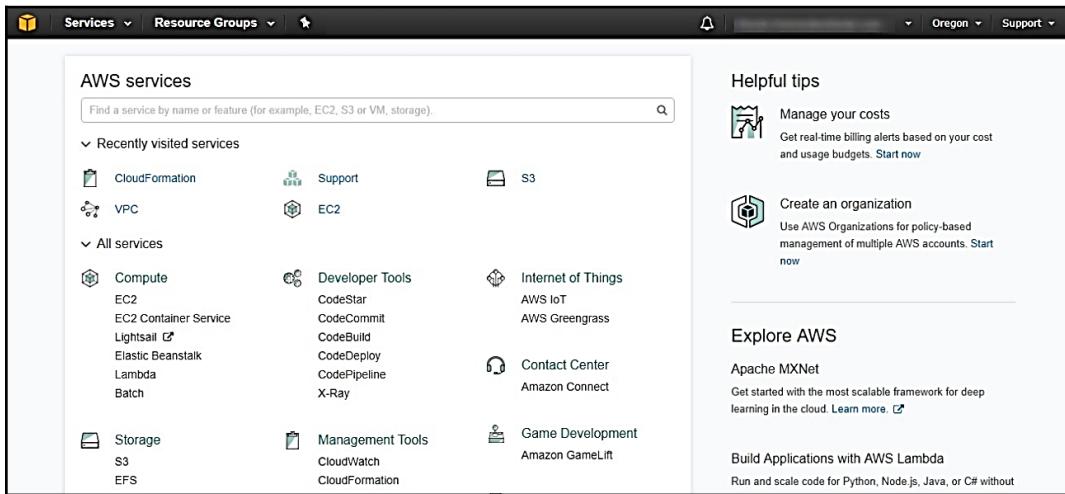
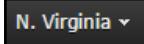


Figure 2.11: The AWS dashboard

Components of the AWS dashboard

As you can see in the preceding screenshot, there are a number of components on the AWS dashboard. The following table gives you an overview of these components:

	This icon represents the console home. By clicking on this, you can go to the dashboard home.
Services ▾	This drop-down menu lists a number of AWS services. By clicking on an individual service, you can go to the relevant service console.
Resource Groups ▾	You can segregate various services in resource groups. Resource groups provide you with a way to quickly group and access frequently used services based on your requirements.
	Frequently accessed services can be dragged and pinned to the top bar. It can be toggled between icon only, text only, and the icon with text.

	This icon indicates all the alerts from AWS. These alerts may be related to any open issues, scheduled changes by AWS, or any other kinds of notifications.
	This is a drop-down menu that is used to access the My Account , My Billing Dashboard , My Security credentials , and Sign Out options.
	Based on your region, you may see the region name here; if you need to change the region, then you can select it from the drop-down menu. Most of the services are region-specific; however, a few of these services are global, such as AWS dashboard, IAM, and a few others.
	The Support drop-down menu lists options for accessing Support Center , Forums , Documentation , and Training and Other Resources .
	Sometimes, it may be time-consuming to find the desired AWS service from the long list of services on the dashboard. AWS provides a search bar on the dashboard that allows you to quickly find any AWS services, and also makes it easier to navigate.

AWS also makes it easier for users by providing recently accessed services at the top of the AWS dashboard. The list of recently accessed services keeps changing, based on your usage pattern.

Core AWS services

AWS services are divided into various groups, based on their use. The following tables describe a number of services provided by AWS with a brief description of the services. As AWS continuously evolves its service catalog, there may be periodic additions made to this list. You can refer to following URL for the latest product listing: <https://aws.amazon.com/products/>

AWS compute services

The following table describes the AWS compute services in brief:

AWS service	Description
EC2	This provides scalable compute capacity, such as virtual servers.
EC2 Container Service	This is a highly scalable and high performance container management service. This supports Docker and runs on a managed cluster of EC2 instances.
Lightsail	This provides template-based computing; it is also called Virtual Private Servers (VPS) . This makes it possible to quickly launch VMs from templates rather than selecting individual components in EC2.
Elastic Beanstalk	This helps developers by deploying and managing applications in the AWS cloud very quickly. Developers just have to upload their application, and the rest is taken care of by Elastic Beanstalk.
Lambda	This facilitates serverless computing by allowing you to run functions or code without actually spinning servers. Unlike EC2 instances, you are only charged for running the functions. AWS does not charge anything for hosting the function.
Batch	This is used for running any amount of batch processing for developers, scientists, and engineers in the AWS cloud.

AWS storage services

The following table describes the AWS storage services in brief:

AWS service	Description
S3	This provides a highly scalable, reliable, and low-latency object storage service.
Elastic File System (EFS)	This is a fully managed, scalable, and sharable storage service among thousands of EC2 instances.
Glacier	This offers secure, durable, and extremely low-cost solutions for backups and archiving.
Storage Gateway	This seamlessly connects on-premises applications or services with AWS cloud storage.

AWS database services

The following table describes the AWS database services in brief:

AWS service	Description
Relational Database Service (RDS)	This service manages RDSs. It supports the Amazon Aurora, MySQL, MariaDB, Oracle, SQL Server, and PostgreSQL database engines.
DynamoDB	This is a fast and flexible NoSQL database service; it provides predictable performance.
ElastiCache	This service makes it easy for you to deploy Memcached or Redis protocol-compliant server nodes in the cloud. Primarily, it improves application performance by storing frequently accessed information into memory.
RedShift	This is a fully managed petabyte-scale columnar data warehousing service. It also provides Open Database Connectivity (ODBC) and Java Database Connectivity (JDBC) connectivity and SQL-based retrieval.

AWS networking and content delivery services

The following table describes the AWS networking and content delivery services in brief:

AWS service	Description
Virtual Private Cloud (VPC)	This service logically isolates networks. It allows us to define IP range selections, create subnets, configure routes, and network gateways.
CloudFront	This provides a Content Distribution Network (CDN) service. Here, content is distributed using edge locations, providing low-latency and high data transfer speeds across the globe.
Direct Connect	This service provides dedicated network connectivity between private data centers and the AWS cloud. It's useful for creating a hybrid cloud.
Route 53	This is a highly available and scalable global DNS service.
API Gateway	This service allows developers to publish, monitor, and maintain APIs in a secure and scalable manner.

AWS migration services

The following table describes the AWS migration services in brief:

AWS service	Description
Database Migration Service (DMS)	This is a cross-schema conversion and database migration tool for Oracle PL/SQL, SQL Server, and more.
AWS Migration Hub	This service provides a centralized location to track and monitor the progress of application migrations. It also provides a set of tools for facilitating the migration process.
Application Discovery Service	This service facilitates the application migration process by quickly discovering an application's dependencies and performance profile that are running on an instance.
Server Migration Service	This service provides an automated process to migrate on-premises VMWare vSphere or Microsoft HyperV/SCVMM instances to the AWS cloud.
Snowball	This is a physical storage device that accelerates the transfer of TBs of data between on-premises data centers and AWS. It is available in two sizes: 50 TB and 80 TB.
Snowmobile	This service is an exabyte-scale data transfer service. One snowmobile can transfer up to 100 PB of data from a data center to AWS, and vice-versa. It comes in a 45-foot long rugged shipping container.

AWS developer tools

The following table describes the AWS developer tools in brief:

AWS service	Description
CodeCommit	This service provides a scalable and managed private Git repository. Anything from code to binaries can be stored in it.
CodeBuild	This service is a fully managed build service. It can compile and test source code, making it ready for deployment.
CodeDeploy	CodeDeploy can be used to automate code deployment to any instance on AWS EC2 or on-premises.
CodePipeline	CodePipeline facilitates continued deployment on AWS. It can be used to automate the software deployment process, allowing a developer to quickly model, visualize, and deliver code for new feature updates.

CodeStar	AWS CodeStar facilitates the creation and management of software development projects on AWS. It helps in rapidly developing, building, and deploying applications on AWS. It easily integrates with other project development toolchains on AWS, such as CodeCommit, CodeBuild, CodeDeploy, CodePipeline, and more.
Cloud9	AWS Cloud9 is a browser-based Integrated Development Environment (IDE) , which can be used to write, run, and debug your application code.
X-Ray	AWS X-Ray is a service that helps you see inside your applications by collecting application request data and analyzing it. Using various tools provided by X-Ray, you can identify issues and opportunities to optimize your application.

AWS management tools

The following table describes the AWS management tools in brief:

AWS service	Description
CloudWatch	CloudWatch can be configured to monitor AWS resources. It can collect metrics and logs to monitor and generate alerts.
CloudFormation	CloudFormation automates and simplifies repeated infrastructure tasks, such as repeatedly creating the same infrastructure in the same or different AZ or region.
CloudTrail	This service records each AWS API call and creates an audit trail. It stores the audit trail/log files in an S3 bucket.
AWS Config	This service provides the AWS resource inventory, configuration history, change notifications, and also helps in security and governance.
OpsWorks	AWS OpsWorks is a configuration management service that uses automation platforms, such as Chef and Puppet, and generates infrastructure, as well as server configurations as code.
Service Catalog	The AWS Service Catalog allows organizations to manage commonly deployed IT services centrally, and assists organizations in achieving consistent governance to meet compliance requirements. It allows users to quickly deploy a set of approved IT services using the governance model set by the organization.
Trusted Advisor	This service helps to reduce monthly billing and increase performance and security.

Systems Manager	This is an automated advisory service specific to an AWS account, which inspects your AWS environment and provides suggestions to reduce cost, improve system performance, and address security risks.
Managed Services	This is a service offered by AWS for the ongoing management of your AWS infrastructure. If you opt for AWS Managed Services, AWS implements best practices to manage your infrastructure, increase operational efficiency, reduce the operational overhead, and minimize the risk.

AWS security, identity, and compliance services

The following table describes the AWS security, identity, and compliance services in brief:

AWS service	Description
IAM	IAM is an AWS service that allows you to create users, groups, and roles to access AWS services and securely control access to various AWS resources through specific permissions.
Inspector	Inspector is an automated security assessment service that is used to test the security state of applications hosted on EC2.
Certificate Manager	This creates and manages Secure Sockets Layer (SSL)/ Transport Layer Security (TLS) certificates. The service also makes it easy to deploy these certificates across various AWS services.
Directory Service	This is an AWS directory service for Microsoft Active Directory (AD) . It makes it easy to deploy directory-aware workloads on AWS resources to use and manage AD in the AWS cloud.
Web Application Firewall (WAF)	With this service, you can configure various application firewall rules to allow, block, or just monitor web requests over an application. It protects web applications from external attacks.
Shield	This is an advance security shield for web applications. It is provided as a managed service to protect web applications running on AWS against Distributed Denial of Service (DDoS) attacks.
Cognito	This provides sign-up authentication tools to web and mobile apps. It also provides the synchronization of data between various devices, such as mobiles, tablets, laptops, and more.

Secrets Manager	This is a management service that facilitates access protection for applications, services, and other IT resources. With the help of Secrets Manager, you can simply manage, retrieve, and rotate database credentials, API keys, and other secrets.
GuardDuty	GuardDuty provides a mechanism that allows you to intelligently detect threats. It continuously monitors and protects the AWS account, as well as the workloads associated with it.
Macie	Macie is an intelligent security service, which uses machine learning for automatically discovering, classifying, and protecting sensitive data on your AWS account.
Single Sign-On	AWS SSO provides a single sign-on service that allows organizations to use their existing corporate authentication service to provide access to all their applications from a centralized location.
CloudHSM	CloudHSM is an integrated security service offered by Amazon, which provides isolated Hardware Security Module (HSM) appliances that can be used by customers for an additional layer of protection for sensitive data that is in line with the strict regulatory compliance requirement of an organization.
Artifact	AWS Artifact provides an on-demand service to download AWS security and compliance documents, which are generally required by auditors and regulators. These artifacts, which are also called audit artifacts, demonstrate the regulatory compliance of AWS infrastructures and services. Some of the regulatory compliance artifacts provided by AWS include ISO, PCI DSS, SOC, HIPPA, and more.

AWS analytics services

The following table describes the AWS analytics services in brief:

AWS service	Description
Athena	Athena is an interactive query service that is used to analyze data in Amazon S3 using SQL.
Elastic Map Reduce (EMR)	EMR is a service based on the Hadoop framework that provides easy and cost-effective big data solutions.
CloudSearch	CloudSearch is a fully managed and scalable textual search solution for websites or applications.

Elasticsearch	Elasticsearch is a scalable managed service that can be used for log analytics, full text search, application monitoring, and more.
Kinesis	Kinesis is an easy-to-use, real-time data collection, processing, analysis, and data streaming service.
Data Pipeline	AWS Data Pipeline facilitates the automated movement and transformation of data. It allows a user to define a data-driven workflow to accomplish one or more tasks.
QuickSight	QuickSight is a fast, easy-to-use business analytics service, which can build data visualization, perform ad hoc analysis, and provide insights into your data.
AWS Glue	AWS Glue is an easy-to-use, cost-effective, and fully managed Extract, Transform, and Load (ETL) service, which can be used to categorize, clean, enrich, and reliably move data across multiple data stores.

AWS machine learning services

The following table describes the AWS machine learning services in brief:

AWS service	Description
Lex	Lex provides a platform to build text and voice-based conversational interfaces, with high-quality speech recognition capabilities.
Polly	Polly is a powerful text-to-speech service provided by Amazon.
Rekognition	Rekognition is a fully managed image recognition service powered by deep learning algorithms.
Amazon Machine Learning	Amazon ML is an easy-to-use machine learning service that allows you to build, train, and test algorithm-based predictive models.
SageMaker	SageMaker is a fully managed machine learning platform that can be used by developers and data scientists for building, training, and deploying machine learning models. SageMaker provides more flexibility and features compared to Amazon Machine Learning.
Amazon Comprehend	Amazon Comprehend helps in building intelligent applications by analyzing and understanding the intent and sentiment of any unstructured text documents, search queries, and chat conversations.

AWS DeepLens	AWS DeepLens is a programmable video camera with tutorials, example code, and pretrained models that can be used to build intelligent applications.
Amazon Transcribe	Amazon Transcribe is an easy-to-use speech-to-text service, based on Automatic Speech Recognition (ASR) technology.
Amazon Translate	Amazon Translate is an easy-to-use language translation service.

AWS IoT services

The following table describes the AWS IoT services in brief:

AWS service	Description
IoT Core	AWS IoT Core is a platform service that helps you in building IoT applications. It does so by providing mechanisms that connect IoT devices to AWS services and other IoT devices. It also helps in securing data and device communications, making it easier to manage and process device data.
IoT 1-Click	AWS IoT 1-Click is a service that can be used by devices, such as IoT buttons, to trigger Lambda functions at the click of a button.
IoT Device Management	AWS IoT Device Management is an easy-to-use cloud-based device management service, which helps in securely managing IoT devices.
IoT Analytics	AWS IoT Analytics is a fully managed IoT data analytics service that supports petabyte-scale IoT data. It is based on a pay-as-you go model.
Greengrass	AWS Greengrass is an IoT software that can be installed on IoT devices to manage and run applications.
FreeRTOS	Amazon FreeRTOS is an OS for microcontrollers, which makes it easy to connect to, securely deploy programs to, and manage low-powered edge devices.

AWS game development services

The following table describes the AWS game development services in brief:

AWS service	Description
GameLift	GameLift is a fully managed service that is used to deploy, operate, and scale session-based multiplayer game servers in the cloud.

AWS mobile services

The following table describes the AWS mobile services in brief:

AWS service	Description
Mobile Hub	This service provides a platform to build, test, and monitor mobile app usage.
AppSync	AWS AppSync is a fully managed data query language service that provides real-time data synchronization and offline programming features. It is based on GraphQL—a data query language developed by Facebook.
Device Farm	AWS Device Farm is an application testing service on AWS Cloud, which provides real physical devices in a virtual environment to test Android, iOS, and web applications on phones and tablets.
Mobile Analytics	Amazon Mobile Analytics is an analytics service for mobile applications that can collect, visualize, and analyze application usage data.

AWS application integration services

The following table describes the AWS application integration services in brief:

AWS service	Description
Step Functions	AWS Step Functions is an application integration service that helps in building distributed applications using visual workflows. As the name suggests, you can build a series of steps for your application in visual workflows that are executed in defined sequences with built-in error handling mechanisms.
Simple Workflow (SWF)	Amazon SWF is a workflow management service that coordinates work between distributed application components in order to accomplish tasks defined in an application workflow.
Simple Queue Service (SQS)	Amazon SQS is a fully managed distributed message queuing service. It makes it easier to decouple application components and scale microservices, distributed systems, and serverless applications.

Simple Notification Service (SNS)	As the name suggests, SNS is a notification service that works on a pub-sub model, where you can publish or subscribe to a specific notification. It can send a message to a subscribing endpoint or a client based on a trigger.
Amazon MQ	Amazon MQ is a fully managed message broker service based on Apache ActiveMQ. The purpose of a message broker is to make it easier for application components to communicate using formal messaging protocols in a number of programming languages.

AWS desktop and app streaming services

The following table describes the AWS desktop and app streaming services in brief:

AWS service	Description
WorkSpace	Amazon WorkSpace is a virtual desktop service on the cloud that allows end users to connect to high-end desktops in the cloud using thin clients.
AppStream 2.0	Amazon AppStream 2.0 is an easy-to-use, secure, and fully managed application streaming service, which can stream desktop applications from the AWS environment to HTML 5-compatible browsers.

AWS business productivity services

The following table describes the AWS business productivity services in brief:

AWS service	Description
Alexa for Business	Alexa for Business is an intelligent voice-enabled assistant at work. People can ask questions to get answers to their query. It can be integrated with a number of Alexa-enabled IoT devices and can be controlled through voice.
Amazon Chime	Amazon Chime is a unified communication service that can be used to conduct online meetings with high quality audio, video, and other online meeting features.
WorkMail	Amazon WorkMail is a managed business emails, contacts, and calendar service. It allows seamless access from mobile devices, web browsers, and desktop clients.
WorkDocs	Amazon WorkDocs is an easy-to-use, fully managed, and secure document storage and sharing service.

AWS customer engagement services

The following table describes the AWS customer engagement services in brief:

AWS service	Description
Amazon Connect	Amazon Connect is a contact center solution over the cloud. Apart from providing usual contact center features, it can be integrated with some of the existing systems and business applications to provide more insight into customer calls.
Pinpoint	Amazon Pinpoint is a customer engagement management solution that can be used to send email, SMS, mobile push notifications, and more. It can also be integrated in applications to get usage data to provide better insights into application usage.
Simple Email Service (SES)	Amazon SES is a cloud-based email service that can be used to send and receive emails.

AWS media services

The following table describes the AWS media services in brief:

AWS service	Description
Elastic Transcoder	Elastic Transcoder is a media transcoding service that helps in converting media files from one format into another. It is mainly used for creating media files that are compatible with different mobile devices, tablets, and personal computers.
Kinesis Video Streams	Amazon Kinesis Video Streams is a fully managed live video streaming service that can be used to stream live video in the AWS cloud. It also facilitates the development of real-time video processing applications and video analytics.
Elemental MediaConvert	Elemental MediaConvert is a cloud-based media transcoding service that can convert an input video format into multiple output formats, which can be viewed on devices with different resolutions.
Elemental MediaLive	Elemental MediaLive is a cloud-based live video processing service. It can create a video stream for broadcasting on a TV and any other internet-connected device. It is helpful in compressing large video files and distributing these files to viewers.

Elemental MediaPackage	Elemental MediaPackage is a cloud-based media service that can prepare video streams for distribution over the internet. It also protects the media by using Digital Rights Management (DRM) .
Elemental MediaStore	Elemental MediaStore is a storage service, specifically designed to serve media content. It provides low latency, consistency, and high performance for delivering live video streams.
Elemental MediaTailor	Elemental MediaTailor is a media service that can embed targeted advertisements into live or on-demand video streams without compromising the viewing experience of the user.

The shared security responsibility model

Before designing and making cloud solutions operational, it is important to understand the security responsibility that is shared between AWS and the customers who consume these services. The following diagram distinguishes between the responsibilities of AWS as a cloud service provider and the customers who consume these services:

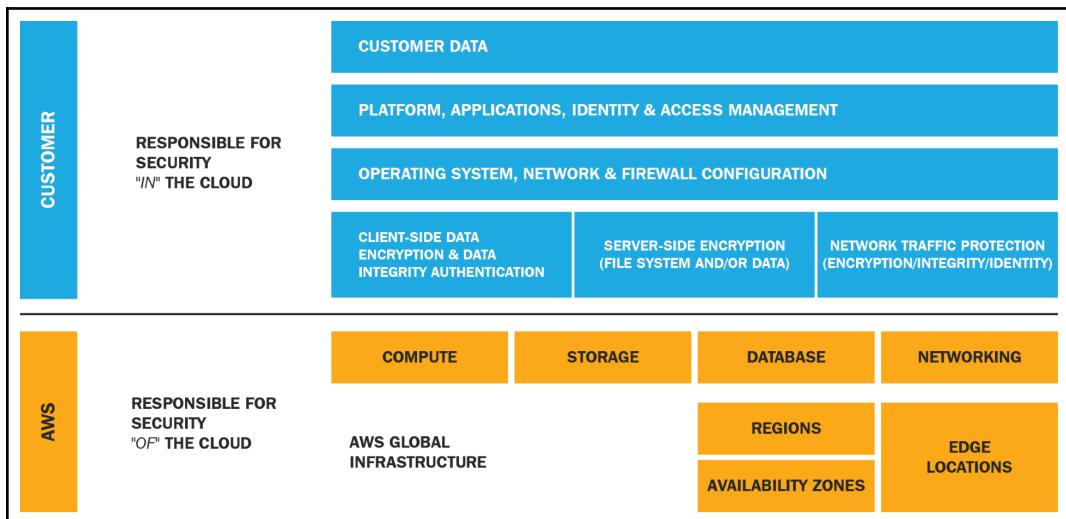


Figure 2.12: The shared security responsibility model

Amazon promises that security is its highest priority as a public cloud service provider. AWS is committed to providing consistent, robust, and secure AWS public cloud services to their customers. Amazon achieves this by securing foundation services (that is, compute, storage, database, and networking) and global infrastructures (such as regions, AZs, and edge locations). Customers have to manage the security of their data, OSes, application platforms, applications, networks, systems, or any customer-specific services that are deployed on AWS by using IaaS. AWS provides a number of security-related services, such as IAM, AWS Inspector, Certificate Manager, Directory Service, WAF, Shield, Cognito, Secrets Manager, GuardDuty, Macie, Single Sign-On, CloudTrail, **Key Management Service (KMS)**, and CloudHSM. All of these security services can be used to manage security in an automated way so that the operations team doesn't need to spend much time on routine security and audit tasks. You can use these services as a building block for your environment in AWS.

In other words, AWS is responsible for providing security to the AWS cloud while the customer is responsible for the security of the resources that are deployed within the cloud. In the case of managed services (that is, DynamoDB, RDS, and Redshift), AWS is responsible for handling the basic security tasks of the underlying AWS resources at the OS level.

To match an organization's IT compliance requirement, AWS also provides third-party audit reports to ensure that the AWS cloud fulfills all the essential compliance needs. You can refer to <https://aws.amazon.com/compliance> for more details.

Some compliance standards followed by AWS are given as follows:

- The IT security compliances include the following:
 - SOC 1/SSAE 16/ISAE 3402 (formerly SAS 70)
 - SOC 2
 - SOC 3
 - FISMA, DIACAP, and FedRAMP
 - DOD CSM Levels 1-5 PCI DSS Level 1
 - ISO 9001 / ISO 27001 ITAR
 - FIPS 140-2
 - MTCS Level 3

- The industry compliances include the following:
 - **Criminal Justice Information Services (CJIS)**
 - **Cloud Security Alliance (CSA)**
 - **Family Educational Rights and Privacy Act (FERPA)**
 - **Health Insurance Portability and Accountability Act (HIPAA)**
 - **Motion Picture Association of America (MPAA)**

The AWS network provides protection against traditional network security problems, such as DDoS, MITM attacks, IP spoofing, and port scanning.

Remember that you need to get prior approval from AWS in order to perform penetration testing on your AWS account; otherwise, AWS understands such testing as a malicious attack and your AWS account may be blocked.

When any **Elastic Block Store (EBS)** volumes are deleted from an AWS account, AWS also ensures that internal wiping takes place before it is reused for another AWS account. The wiping process is performed as per industry standards (that is, DoD 5220.22-M or NIST 800-88). It is also possible to encrypt sensitive data on EBS volumes. AWS uses the AES-256 algorithm for encryption.



A detailed understanding of the shared responsibility model can be obtained at <https://d0.awsstatic.com/whitepapers/aws-security-whitepaper.pdf>.

AWS soft limits

For every AWS account, region-based limits are enabled for each AWS service. Such limits restrict an AWS account to provision resources up to a specific limit. For example, AWS imposes a soft limit of around 20 EC2 instances in a new account. This limit may vary according to resource types and the respective AWS services. Some of these limits are soft limits, and you can raise a support request to AWS for revising this limit in your AWS account.

AWS Trusted Advisor displays the account usage and limits for each specific service region. Authorized IAM users or root accounts can place a request with AWS Support in order to increase these service limits.

Here's how you can request a change in service limits:

1. Log in to your AWS account; in the top right-hand corner, click on the **Support** drop-down menu and select **Support Center**.
2. Click on **Create Case** and select **Service Limit Increase**.
3. In the **Limit Type** drop-down menu, select the AWS service for which the limit is to be increased.
4. Fill in the details and submit the support request.



When any resource's soft limit is reached, no new resources can be provisioned until the soft limit is increased by AWS support. For example, if your soft limit in North Virginia is 20 EC2 instances, then spinning up the 21st instance results in an error.

DR with AWS

For any enterprise, unplanned downtime can have a devastating impact. Unplanned downtime not only impacts the ongoing business, but it can also create an adverse impact on the future of the organization. Any catastrophe or disaster can bring a city or a region to a standstill and can impact businesses for a prolonged period of time. It is critical for organizations to plan for disasters that may halt their business. AWS provides a number of services and features that can be used to overcome unplanned downtime arising out of natural disasters or human error.

DR is the process of designing an architecture that is able to recover from any disaster situation within a stipulated time. The cost of DR planning is inversely proportional to the time required to recover the infrastructure. Traditionally, in the case of a private data center, it may be required to create similar data centers in a distant and safe place. It also requires huge upfront investment, constant staff training, and maintenance.

On the other hand, by hosting infrastructure on AWS, it can be easier to plan for a DR setup in multiple regions or AZs. Resources in such additional regions or AZs can be automatically scaled up or down according to the actual workload, and you pay only for what is actually used. One of the best aspects of AWS is its global infrastructure. You can select from a wide range of regions and AZs to host your DR infrastructure.

Before DR planning begins, there are two very critical aspects of a DR plan that we need to understand:

- **Recovery Time Objective (RTO):** This defines the time period in which business processes should be recovered from downtime. The maximum time a business process can sustain a down condition depends on the critical nature of the process. Generally, the RTO is defined separately for each business process and related environment. For example, let's consider the RTO of a business process as 3 hours. In this case, if a disaster occurs, then businesses can afford a downtime of 3 hours at the most – the environment should be recovered within this time.
- **Recovery Point Objective (RPO):** This defines the acceptable amount of data loss, measured in time. For example, let's consider that the RPO of a business process is 1 hour. In this case, if a disaster occurs at 2:00 PM, then all the data stored until 1:00 PM should be made available. Thus, a business can afford to lose 1 hour's data in case of any disaster.

The best way to define the RTO is to understand the financial impact on the business if the system is not available. Similarly, the best way to define the RPO is to understand the financial impact on the business; that is, if the business data of a specific time frame is not available. By considering these aspects, along with the technical feasibility and cost of setting up DR, you can understand and define the RTO and RPO of a business process.

Hosting a DR site usually involves a huge upfront capital investment. On the other hand, by hosting an IT infrastructure on AWS, it is possible for you to go for a pay-as-you-go model without any upfront capital investment. AWS services and concepts, such as regions, AZs, EC2 instances, the DNS (Route 53), networking services (VPC), **Elastic Load Balancing (ELB)**, autoscaling, CloudFormation, and many others, can help in designing a robust DR plan.

The following diagram shows the various DR reference models that use AWS services:

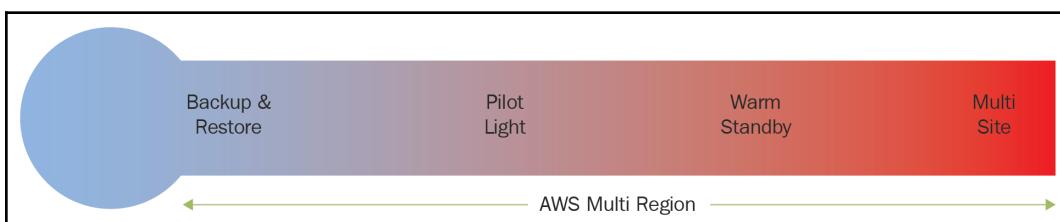


Figure 2.13: DR options

The DR approach can be broadly categorized into four models, as follows:

- Backup and restore
- Pilot light
- Warm standby
- Multi-site

Out of the four models described in the preceding diagram, backup and restore is the most economical model, but it may take more time in recovery. If you go from left to right in the preceding diagram, the time to recover decreases, but at the same time, the cost to recover increases. Similarly, if you go from right to left, the time to recover increases and the cost to recover decreases. Let's understand these DR models in the subsequent sections.

The backup and restore DR model

This approach to DR involves periodically backing up critical data and keeping it in a safe and secure place for later use. In the event of disaster, the backed up data can be restored as needed.

Let's consider a traditional data center approach to understand the backup and restore DR model. In this approach, data is periodically backed up on tape drives and sent offsite. In the event of a disaster in DC, you need to bring the backup tapes from an offsite location to wherever restoration takes place. It takes longer to restore the data from tape drives, as tape drives perform a sequential read of data. Usually, this type of DR mechanism is obsolete in a modern, fast-paced enterprise.

For a similar backup and restore approach, AWS provides a number of services, such as the S3 and Glacier services. Transferring data to and from S3 and Glacier is much faster, safer, and economical compared to tape drives. S3 can be compared to offline object storage, whereas Glacier can be compared to an economical tape drive that is used for archival purposes. S3 provides instant access to your data, whereas the comparatively cheaper Glacier option may take a minimum of 3 to 4 hours to restore the archived data. AWS also provides a service called Import/Export, which offers physical data storage devices that can be used to perform large data transfers directly to AWS. There are other options as well for importing/exporting a large amount of data on AWS, such as Snowball or Snowmobile.

AWS Snowball is a service that speeds up the movement of huge amounts of data in and out of AWS using physical storage appliances that bypass the internet.

AWS Snowmobile is an exabyte-scale data transfer service that's used for moving huge amounts of data to AWS. You can transfer up to 100 PB per Snowmobile. It's a 45-foot long, rugged shipping container that is pulled by a semi-trailer truck.

AWS also provides a service called Storage Gateway. It directly connects a physical data center to AWS storage services. Using Storage Gateway, organizations can directly store data in S3 or Glacier as a part of their DR strategy. Storage Gateway is fast, economical, and robust.

The following diagram shows a reference configuration of a backup and restore DR setup scenario on AWS and a physical data center:

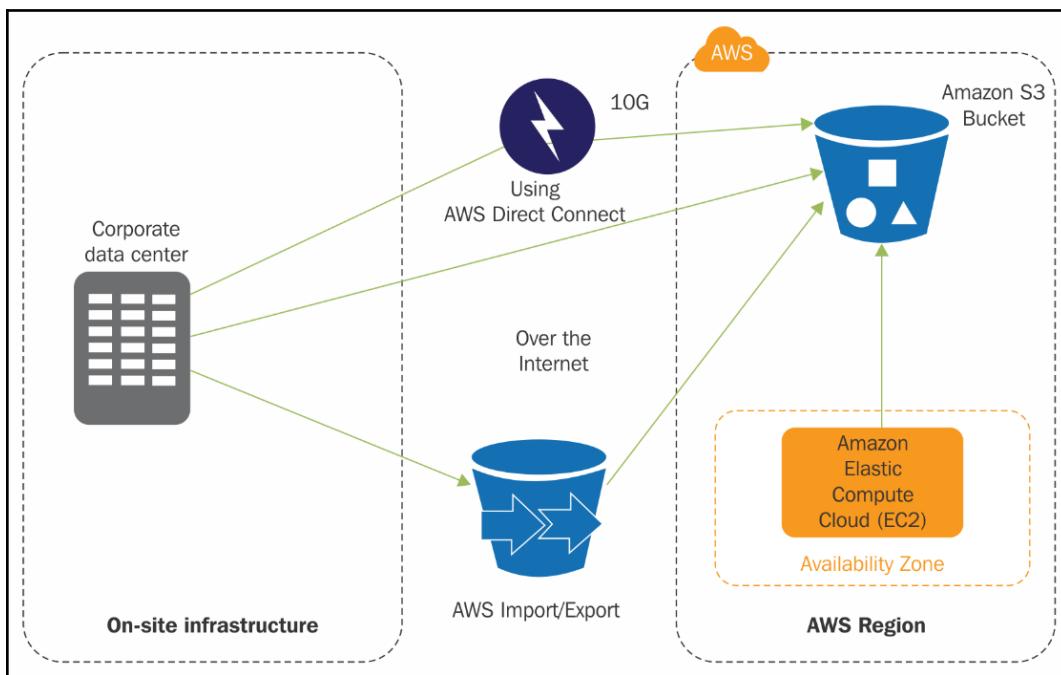


Figure 2.14: A backup and restore DR setup

The pilot light DR model

This model can be easily explained using the example of a gas heater, where a small flame is always ignited and can be quickly ignited into a larger flame in a boiler in order to heat up an entire house. The term pilot light is derived from the same concept.

In this DR scenario, the minimal DR setup with the most critical components of the environment is always kept running in parallel with the production environment. This can be explained through a number of examples:

- The primary site runs on an on-premises DC, and a DR setup with minimal critical resources is kept running in the cloud. In the event of any issue in the data center, other required resources can be provisioned in the cloud infrastructure and it can start serving traffic after a failover to the DR setup.
- The primary site runs on AWS in one region and a DR setup with minimal critical resources is kept running in another region on AWS. In the event of any issue in one AWS region, other required resources can be provisioned in another region to match the primary environment setup. It can start serving traffic after a failover to the DR setup.

When an enterprise's IT runs on a traditional data center, it is also possible to use AWS as a DR site. You can shift critical applications, load them to the AWS cloud, and run the rest of the infrastructure in a traditional fashion. The following diagrams display how this model works in normal conditions and failover conditions, respectively:

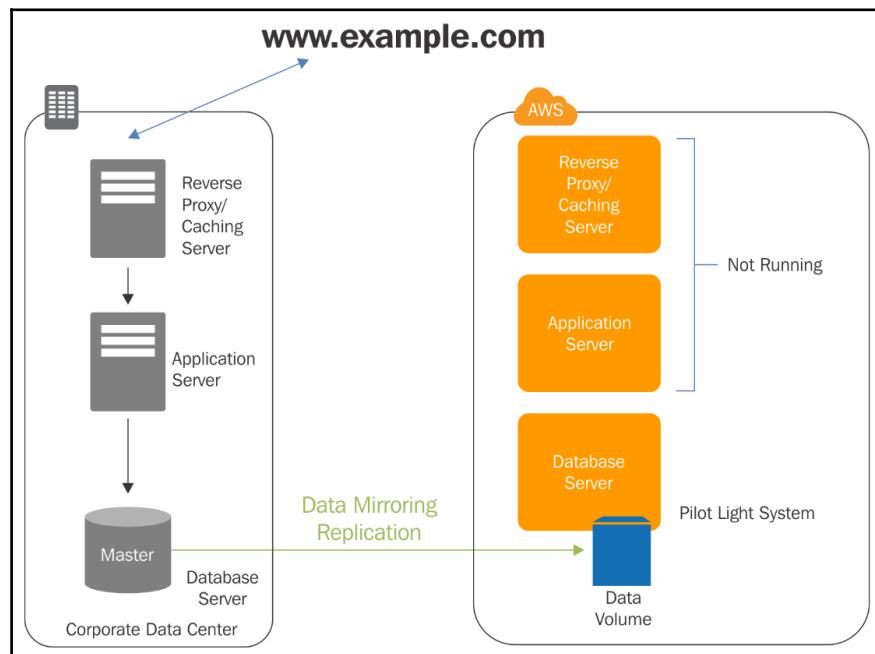


Figure 2.15: An application running in a normal situation

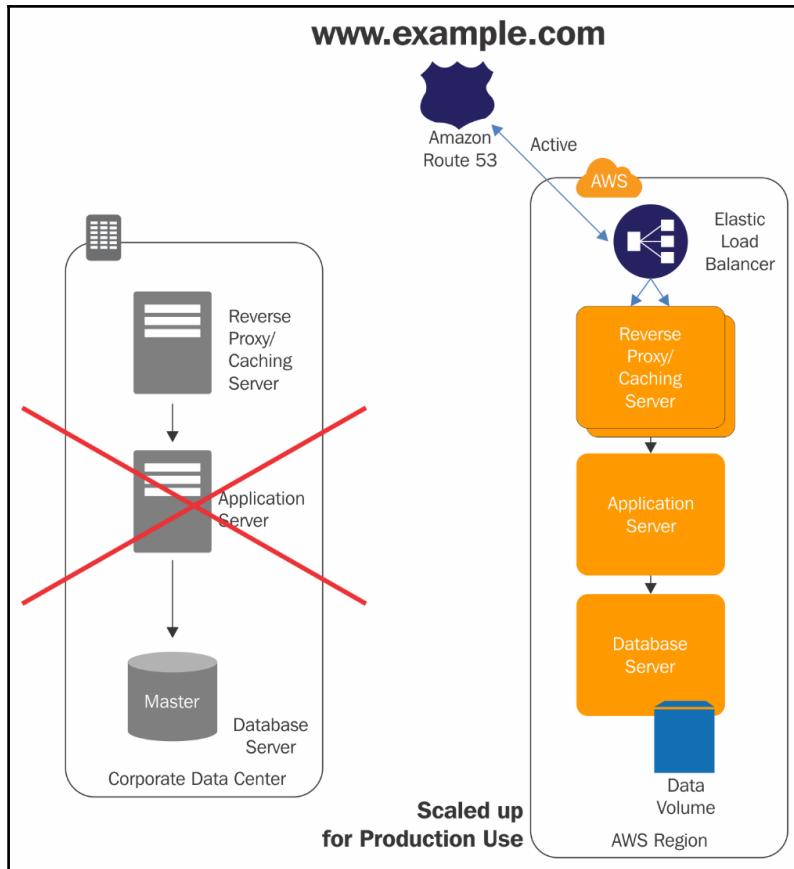


Figure 2.16: The earlier infrastructure in a failover condition

With the help of the preceding diagrams, we can clearly understand that in the event of any disaster at the primary site (that is, a data center or a cloud-based site), traffic is automatically diverted to the secondary site. This is also called a failover scenario; thus, users experience minimal or no performance degradation. Compared to the backup and restore model, the pilot light model is much faster. It requires a few manual steps to perform or it can be automated with the help of an automation process using DevOps.

The warm standby DR model

In this DR scenario, a scaled-down and fully functional environment is always kept running in the cloud. It is an extension of the pilot light DR model. In the case of the pilot light model, you need to create additional instances or resources to match the size of the primary environment. In contrast to the pilot light model, warm standby keeps a fully functional DR setup with a minimum fleet of instances with the minimum possible size of instances. In the event of any disaster, the DR setup is scaled up to match the primary site, and traffic is failed over to the DR setup. Since a fully functional infrastructure is always kept running in the warm standby approach, it further reduces the required recovery time.

The following diagrams explain the concepts of primary and secondary sites under a warm standby DR model. They explain how the load is transferred to a secondary site when the primary site fails:

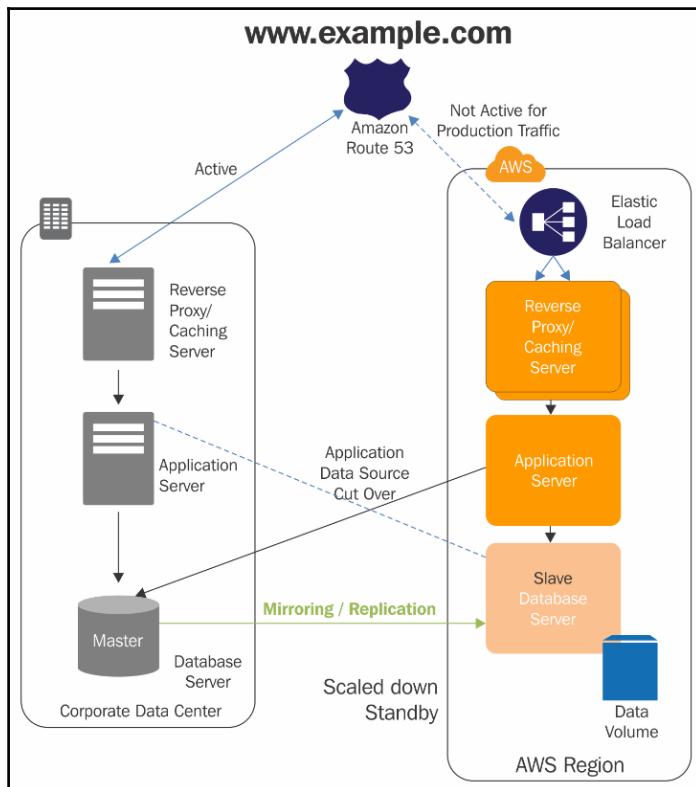


Figure 2.17: The warm standby DR setup in normal conditions

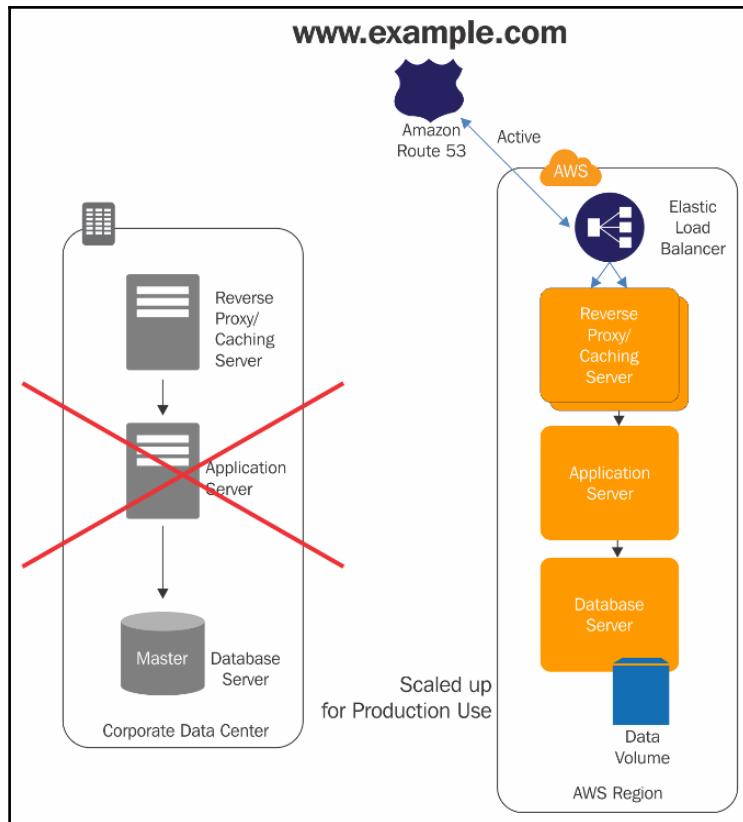


Figure 2.18: End users are diverted to a secondary site under the warm standby DR model

The multi-site DR model

In a multi-site DR model, the infrastructure of the same size is always kept running in sync at multiple locations. The DR setup can be hosted on a combination of a physical data center and AWS, or an entire DR setup can be hosted on multiple AWS regions. In either case, when a primary site fails, the total workload is transferred to the secondary site. In some practices, the secondary site keeps handling partial traffic in parallel to the production environment, even in a normal scenario. Such ratios can vary from 50:50 to 80:20 between primary and secondary sites.

In multi-site models, resources are always synchronized across primary and secondary sites. Compared to all the previous DR models, the multi-site option provides the least downtime (that is, virtually none), but it is comparatively more expensive. Since AWS follows the pay-as-you-go model for actual usage, it doesn't cost much compared to the loss that can occur to an enterprise if a site fails. The following diagrams describe multi-site DR configuration:

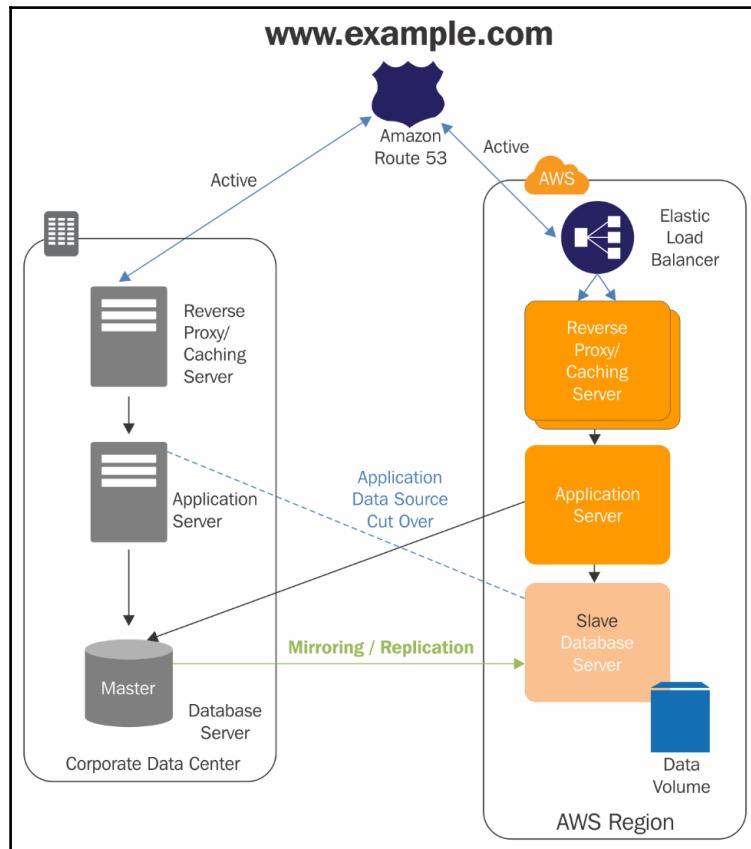


Figure 2.19: End users access primary and secondary sites in parallel under normal circumstances

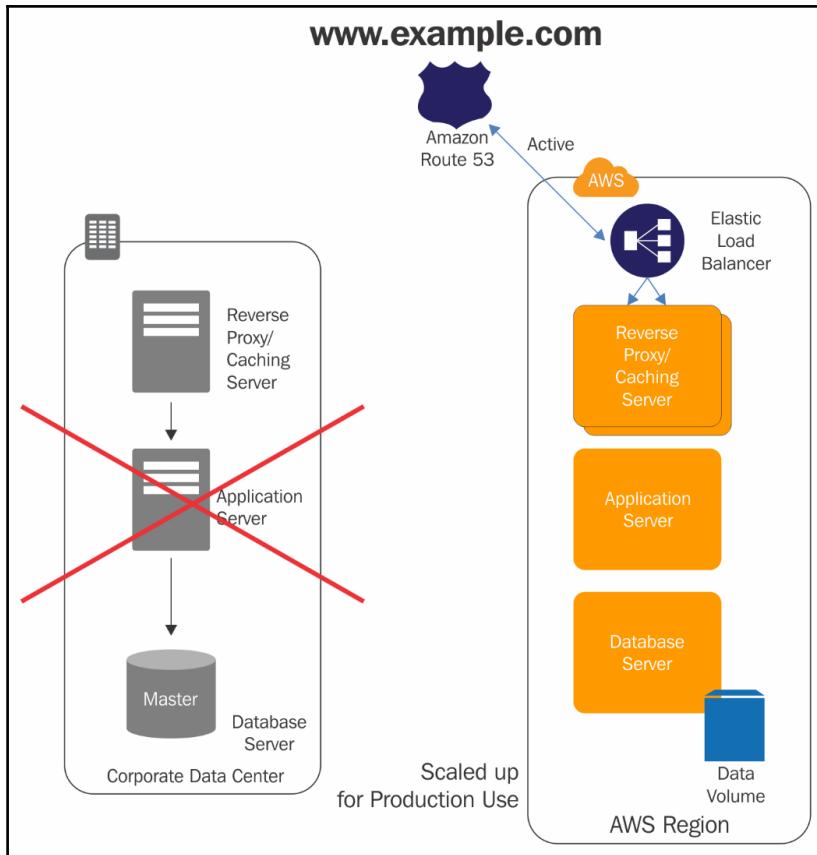


Figure 2.20: The whole workload is transferred to a secondary site if the primary site fails

Summary

This chapter elaborated on the fundamentals of Amazon Web Services. We started off with providing a basic understanding of what the cloud is and took time to go through a brief journey of familiarizing ourselves with the basic building blocks of Amazon Web Services. This chapter highlighted some of the critical aspects of how AWS works and provided an overview of AWS' core infrastructure. In the next chapter, we will get familiar with **Identity and Access Management (IAM)**.

3

Identity and Access Management (IAM)

Any organization using IT services may have a number of resources, processes, environments, projects, and operational activities. All the organizational activities are carried out by various departments. There is always a clear distinction as to who needs to do what. Infrastructure and operational activities are performed by a specific set of personnel in the organization. Similarly, there may be a development team, testing team, project team, finance team, security team, and other relevant teams in the organization that need to perform specific tasks in the organization.

Just as there are various teams to perform different activities in the organization, similarly, there is a clear set of responsibilities and accountability for each of the individuals in a team. A developer may not need to access testing resources. Similarly, a testing resource may not need to be accessed in production environments. On the other hand, a system administrator may need to access all the servers, but may not need to access security compliance hardware.

Based on the preceding examples, we can understand that an organization has complex access and security requirements. Addressing an organization's access and security requirements is a critical task. AWS provides a robust service to fulfill this requirement, called the **Identity and Access Management (IAM)** service.

Here's how we can define IAM—**AWS IAM** is a global service that is specifically designed to create and manage users, groups, roles, and policies for securely controlling access to various AWS resources. IAM can be used to control who can use the resources (authentication), what resources can be used, and the different ways in which these resources can be used (authorization).

We will cover the following topics in this chapter:

- Understanding the AWS root user
- Elements of IAM:
 - Introducing the AWS CLI
 - Groups
 - IAM roles
 - Policies
 - STS
- IAM best practices
- Exam tips

Understanding the AWS root user

Creating an AWS account also creates a root user. The email address and password supplied at the time of creating the AWS account become the username and password for the root user. This combination of an email address and password is called the **root account credentials**.

The root account (that is, the root user) has complete, unrestricted access to all resources on the account, including billing information. This account is a superuser and its permissions cannot be altered by any other user on the account.

Since the root account has unrestricted access to all the resources on the account, it is highly recommended that you avoid using the root account for day-to-day activities. On a newly created AWS account, it is recommended that you create individual IAM users based on organizational needs and assign them the required permissions. These non-root-user accounts should be used for day-to-day activities.



Never share your credentials with other users, especially root credentials, as they give unrestricted access within an AWS account.

Elements of IAM

It is essential to understand a few basic IAM terms to effectively manage real-life organizational users and their permissions for accessing AWS resources as per their roles and responsibilities. The following list briefly describes these terms, and subsequently goes into the details of each of the elements of IAM:

- **User:** A user is a person or an application that requires access to various AWS resources to perform designated tasks. A user can access AWS resources with either a username and password, or with an access key and secret key.
- **Access key:** An access key is a 20-character alphanumeric key that acts as a user ID.
- **Secret key:** A secret key is a 40-character alphanumeric key that acts as a password or secret key. The access key and secret key are used together for initiating API, SDK, and CLI authentication.
- **Password policy:** The password policy specifies the complexity requirements of passwords, and defines the mandatory rotation period for a password associated with IAM users.
- **Multi-factor authentication (MFA):** This is an extra layer of security protection for user authentication that requires users to enter a six-digit token along with their username and password.
- **Group:** A group is a collection of IAM users.
- **Role:** A role is an IAM entity that constitutes one or more IAM policies defining resource permissions. A role enables access to perform specific operations mentioned in the respective policies associated with the role.
- **Policy:** A policy is a document, written in JSON format, that formally states one or more permissions as per the IAM policy standards.

Let's now look at all of these terms in detail, and their significance in IAM.

Users

AWS IAM users can be created for any organizational entity (as in actual end users, including a person or an application). As per their roles and responsibilities in the organization, these users need access to AWS resources to perform their day-to-day tasks. By asking the question *who is that user?*, we will get an idea of whether that entity is a user or an application.

Usually, an individual user is authenticated by a username and password. Similarly, requests for programmatic access (that is, SDKs and CLIs, also known as applications) are authenticated using an access key and secret key. Individual users can also use an access key and secret key by configuring them on EC2 instances or using physical computers to execute AWS CLI commands.

It is best practice to identify organizational entities and create respective IAM users with the necessary credentials to give them access to the AWS platform. Every user, whether it is an individual or an application, must provide appropriate credentials for authentication. Only after a successful authentication can a user access AWS resources, such as the AWS dashboard, API, CLI, or any other AWS service.

A logical representation of organizational users can be found in the following diagram:

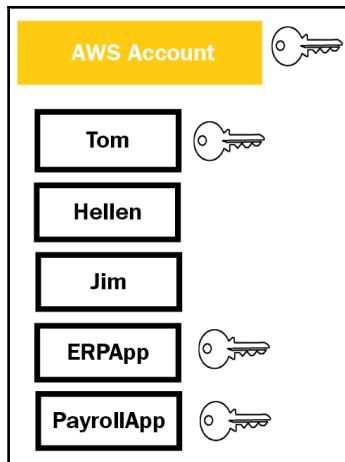


Figure 3.1: Conceptual understanding of IAM users in AWS accounts

As shown in the preceding diagram, various users are created in AWS accounts for individuals or applications. Let's suppose that **Tom** is a system administrator, **Hellen** is a network administrator, and **Jim** is a database administrator. The best practice is to grant the minimum privileges required for each user, based on what their role is expected to be. A system administrator may need access to all the infrastructure, while a network administrator may need access to all network services and resources. Similarly, a database administrator may need access only to databases.

On the other side, applications such as ERP and payroll may also need to access the required AWS resources. Applications hosted on EC2 may need to access a hosted database on RDS. For all such applications, there may be a user ID with an access key and secret key that an application can use to access the respective resources on AWS.

Permissions can be granted to applications using an AWS role as well. In subsequent sections, we will see how authentication works with AWS roles.

The access key and secret key are not generated for all users, as we saw in the preceding diagram. They are only generated for users who need to access AWS resources using APIs, SDKs, and CLIs. To access services using an AWS console, you can use a username and password.

Access key ID and secret key

An access key ID and secret key come in a pair. An access key ID is a 20-digit key and a secret key is a 40-digit key. Only corresponding keys work with each other for authentication. These keys are used along with AWS SDK, CLI, REST, or Query APIs. As the name suggests, a secret key is meant to be kept secret and protected. The best practice is to not hardcode an access key ID and a secret key in application coding. If these keys are hardcoded and not removed before sharing an AMI or EBS snapshot with others, it may pose a security risk.

You can generate access key IDs and secret keys, either at the time of creating IAM users or later, as and when required. At the time of generating an access key and a secret key, AWS gives an option to download them in CSV format. Once it is created, you need to download it and keep it secure. AWS does not provide any mechanism to retrieve an access key ID and a secret key if these keys are lost. The only solution is to delete old keys and generate new keys. As a result, you will need to edit an earlier key pair with a newly regenerated key pair for applications to work smoothly. A maximum of two sets of access keys and secret keys can be attached to any IAM user profile.



An access key and secret key look like this:

Access key ID : AKIAJ4B7SOIHQBQUXXXX

Secret key: oSpG3je8kYS1XpMDRG8kpo1awLizvnv1GaNBXXXX

It is a best practice to periodically rotate access keys and secret keys for security purposes. It is also a recommended practice to periodically remove unutilized and unwanted access keys and secret keys from an AWS account.

Password policies

A password policy specifies the complexity requirement of a password and defines a mandatory rotation period for passwords associated with IAM users.

While creating an IAM user, an IAM administrator can provide a reasonably strong password on behalf of the user. Optionally, an IAM administrator can also configure a prompt to the user to change the respective user password when the user logs in to AWS for the first time. A password policy can be configured from **Account settings** within the IAM dashboard. As per the organization's compliance requirements, password complexity can be configured by choosing one or more options, as shown in the following screenshot:

The screenshot shows the 'Password policy' configuration interface. It includes fields for 'Minimum password length' (set to 6), a list of complexity requirements (with 'Allow users to change their own password' checked), and options for password expiration and reuse. At the bottom are 'Apply password policy' and 'Delete password policy' buttons.

Minimum password length:	6
<input type="checkbox"/> Require at least one uppercase letter ⓘ	
<input type="checkbox"/> Require at least one lowercase letter ⓘ	
<input type="checkbox"/> Require at least one number ⓘ	
<input type="checkbox"/> Require at least one non-alphanumeric character ⓘ	
<input checked="" type="checkbox"/> Allow users to change their own password ⓘ	
<input type="checkbox"/> Enable password expiration ⓘ	
Password expiration period (in days):	
<input type="checkbox"/> Prevent password reuse ⓘ	
Number of passwords to remember:	
<input type="checkbox"/> Password expiration requires administrator reset ⓘ	
Apply password policy	
Delete password policy	

Figure 3.2: Password policy options

It is important to note that the password policy only affects the user password. It does not affect the access key and secret key in any way. As a result of a password policy, a user's password may expire after the configured number of days, but the access key and secret key do not expire. When a password expires, the user cannot log in to the AWS console, but API calls work fine using an access key and a secret key.



A change in password policy comes into effect for all new users, but for all existing users, it comes into effect whenever their respective password is changed. It does not apply to existing users' passwords until their passwords are updated.

Multi-factor authentication

Multi-factor authentication (MFA) is an extra layer of security protection for user authentication that requires users to enter a six-digit token on top of their usernames and passwords. MFA can be enabled for individual IAM users. It is a recommended practice to enable MFA for all users. It adds an extra layer of protection on top of the username and password. Once it is enabled, the user needs to enter a unique six-digit authentication code from an approved authentication source when trying to access the AWS Management Console.

MFA can be enabled for both types of users—an individual console login, and an application's programmatic calls to AWS. It can also be enabled for the root user.

MFA can be enabled using a hardware-based MFA device, or a software-based or virtual MFA device. AWS used to support an SMS token service for MFA, but this was deprecated at the beginning of 2019.

AWS has recently introduced **Universal 2nd Factor (U2F)** as an option for MFA. U2F is based on open authentication standard. U2F can be based on specialized USB keys, or on **near-field communication (NFC)** devices—the technology similar to the one found in smart cards. AWS currently supports U2F only based on USB keys; it does not support NFC devices for MFA. You can refer to the following diagram to understand what a U2F device looks like:

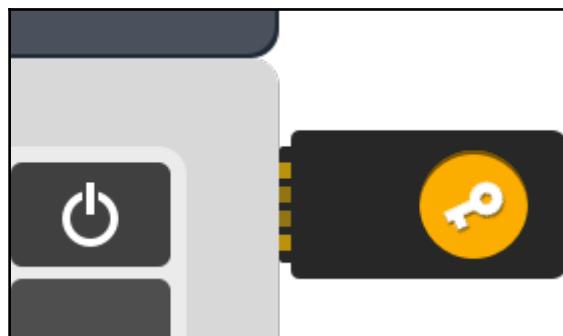


Figure 3.3: U2F device

For more details on U2F, you can refer to the official documentation, available at https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_mfa_enable_u2f.html.

Security token-based MFA

When it comes to security token-based MFA, there are two options available: hardware-based and software/virtual (that is, mobile application)-based. You can purchase hardware-based security tokens from an authorized vendor and then install a virtual security token application on your smartphone. A hardware-based MFA token device may look something like this:



Figure 3.4: A hardware MFA device for RSA tokens

To enable MFA tokens on an IAM user, MFA hardware or software applications need to be registered with an IAM user. Once a user is registered with either a hardware device or software application, it keeps generating six-digit numeric codes based on a time synchronization one-time password algorithm. It appears for 30 seconds and keeps changing. Enabling MFA hardens the security layer. If the username and password fall into an unauthorized person's hands, the person still can't misuse it without an MFA token. The MFA token keeps on rotating the token and is generated only through a synchronized MFA device for that particular IAM user.

Steps for enabling a virtual MFA device for a user

The following steps describe how to enable MFA for a user:

1. Log in to the AWS console.
2. Go to the IAM dashboard.
3. Select **Users** from the left-hand pane and click on a user, as shown in the following screenshot:

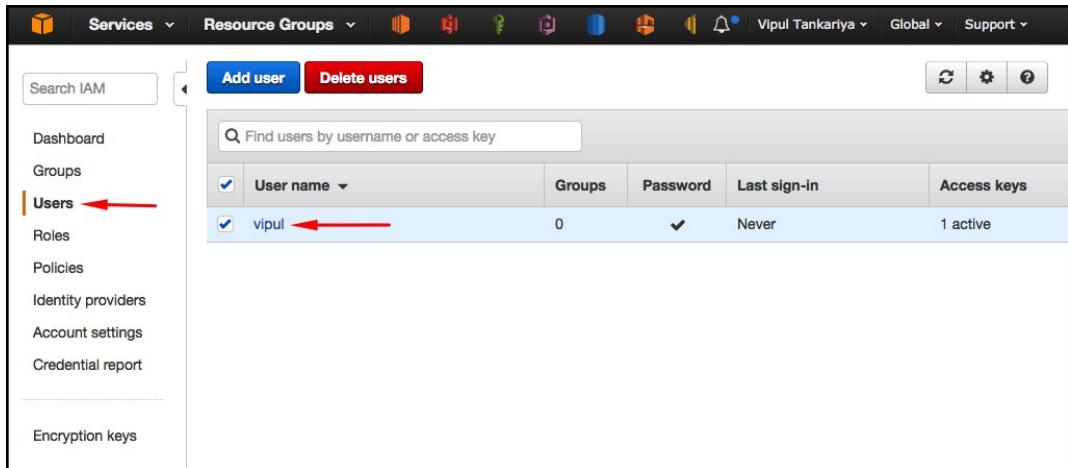


Figure 3.5: User selection when enabling virtual MFA

4. Select the **Security credentials** tab, as shown in the following screenshot:

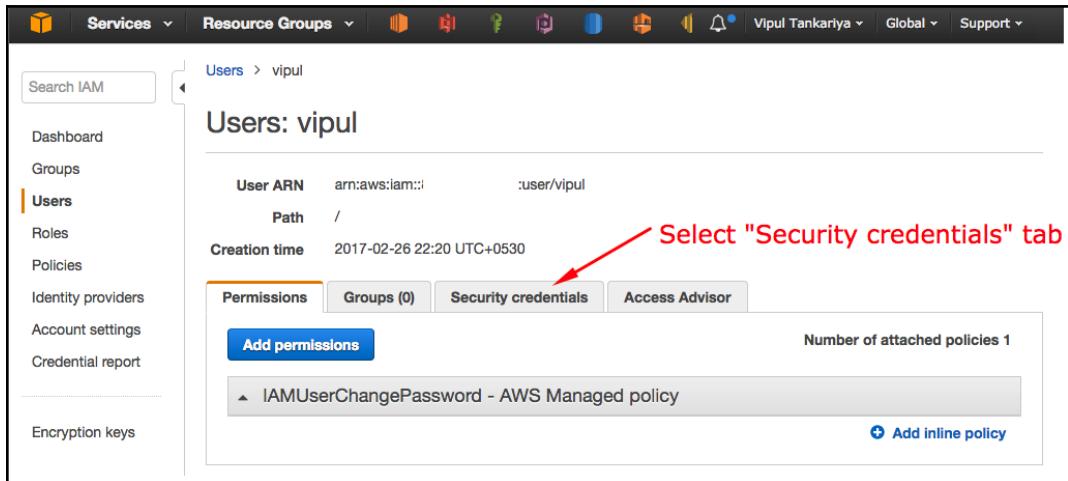


Figure 3.6: Enabling MFA – Security credentials tab selection

5. Click on the **Manage** link, next to **Assigned MFA device**, as shown in the following screenshot:

The screenshot shows the AWS IAM User Summary page for a user named 'bhavin'. The left sidebar has 'Users' selected. The main area shows the user's ARN, creation time, and a list of security credentials. The 'Security credentials' tab is active. Under 'Sign-in credentials', there is a summary and a 'Console password' section. Under 'Assigned MFA device', it says 'Not assigned' and has a 'Manage' link, which is highlighted with a red arrow.

Figure 3.7: Enabling MFA – editing assigned MFA devices

6. Select **Virtual MFA device** and click on the **Continue** button:

The screenshot shows the 'Manage MFA device' dialog box. It asks to choose the type of MFA device to assign. Three options are available: 'Virtual MFA device' (selected), 'U2F security key', and 'Other hardware MFA device'. A red arrow points to the 'Virtual MFA device' option. At the bottom, there is a 'Cancel' button and a blue 'Continue' button.

Figure 3.8: Enabling MFA – selecting a virtual MFA device

7. On the subsequent screen, click on the **list of compatible applications** link, as shown in the following screenshot. This provides a list of AWS MFA-compatible apps, supported for various mobile platforms:

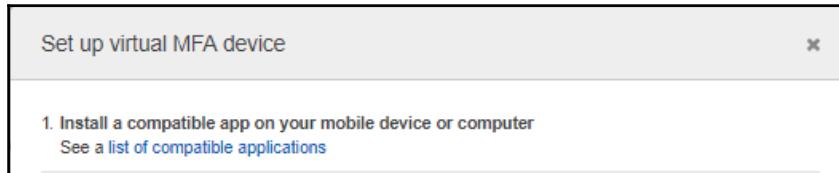


Figure 3.9: Enabling MFA – selecting a virtual MFA device

8. You can download the virtual MFA applications from your respective device application store. Supported applications for various platforms are indicated in the following screenshot. You can close this information window to go back to the previous screen, as indicated in the previous screenshot, and click on **Next Step**:

Virtual MFA Applications	
Applications for your smartphone can be installed from the application store that is specific to your phone type. The following table lists some applications for different smartphone types.	
Android	Google Authenticator; Authy 2-Factor Authentication
iPhone	Google Authenticator; Authy 2-Factor Authentication
Windows Phone	Authenticator
Blackberry	Google Authenticator

Figure 3.10: Enabling MFA – virtual MFA applications

9. Determine whether the MFA application supports QR codes, and then do one of the following:
- Use your mobile app to scan the QR code. Depending on the application that you use, you may have to choose the camera icon or some similar option. You will need to use the device camera for scanning the code.
 - In the **Manage MFA Device** wizard, choose **Show secret key for manual configuration**, and then type the secret configuration key into your MFA application.

When you are finished, the virtual MFA device starts generating one-time passwords:

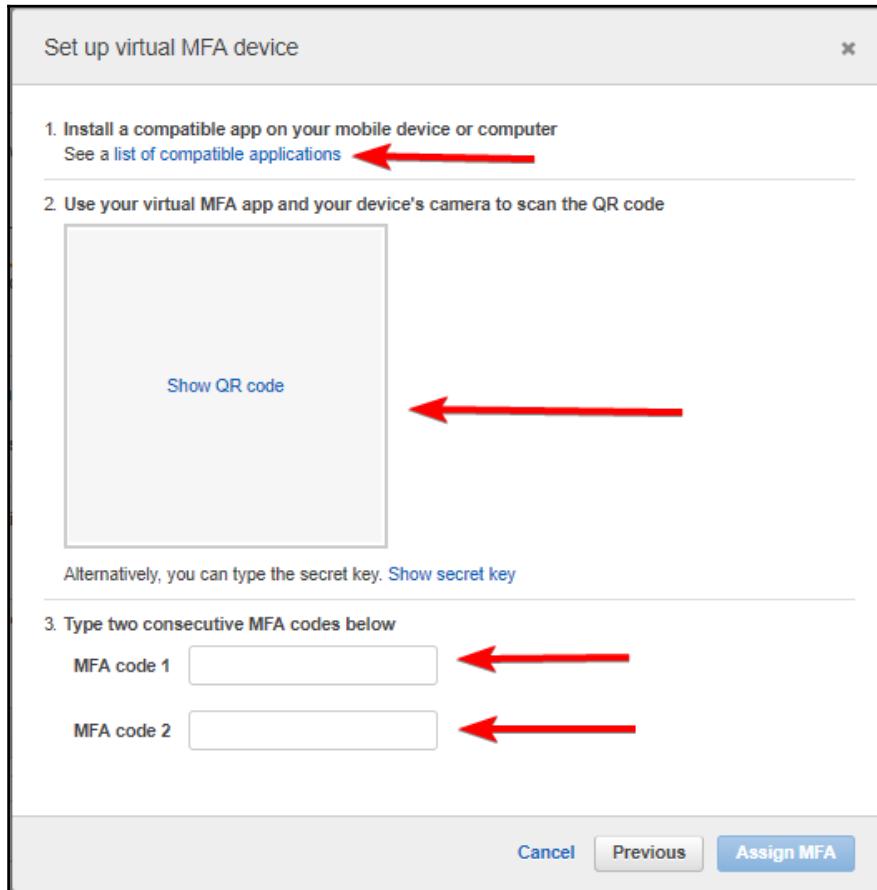


Figure 3.11: Enabling MFA – entering authentication codes

10. As shown in the previous screenshot, you need to type the one-time password into the **MFA Code 1** field. You can use the one-time password given in the virtual MFA device. Before you can enter the second one-time password into the **MFA Code 2** box, you need to wait for approximately 30 seconds.
11. After a wait period, the device generates another one-time password. You can use the fresh one-time password and enter it into the **Authentication Code 2** box. Subsequently, you can choose **Activate Virtual MFA**.

Now your virtual MFA device is ready for use. When a user for whom the MFA token is enabled tries to log in to the AWS console, AWS poses an MFA token challenge after authenticating the user with a valid user ID and password.

Creating an AWS IAM user using the AWS dashboard

The steps for creating an AWS IAM user using the AWS dashboard are as follows:

1. Log in to the AWS Management Console with the appropriate credentials. The IAM user must have sufficient privileges to create IAM resources (that is, users, groups, policies, and so on). In the case of a fresh AWS account, you need to log in with root credentials.
2. Select **IAM** under the **Security, Identity & Compliance** group from the AWS dashboard. This will take the user to the IAM dashboard.
3. Select **Users** and click **Add user**. This displays the following screen:

The screenshot shows the 'Add user' wizard in the AWS IAM console. The top navigation bar includes 'Services', 'Resource Groups', and various AWS service icons. On the right, there are user profile and account settings. The main title is 'Add user'. Below it, a progress bar shows four steps: '1 Details' (highlighted in blue), '2 Permissions', '3 Review', and '4 Complete'. The first step, 'Set user details', asks for a 'User name*' (e.g., 'Vipul') and provides a link to 'Add another user'. The second step, 'Select AWS access type', offers two options: 'Programmatic access' (selected) and 'AWS Management Console access'. Both descriptions mention the generation of access keys or passwords. At the bottom, a note says '* Required' and provides links to 'Cancel', 'Next: Permissions', 'Feedback', 'English', and legal links.

Figure 3.12: IAM—the Add user screen

4. Provide a meaningful and relevant username to resemble a real-world entity. This will help to easily identify the correct user when performing day-to-day maintenance activity. A valid username can contain only alphanumeric characters and the _, +, =, ., @, and – symbols. It is also possible to add multiple users (up to a maximum of 10 users) at the same time by clicking on the **Add another user** link given next to the **User name** textbox, as shown in *Figure 3.12*.
5. After entering the username, you must select **Access type**. You need to select at least one option. It is also possible to select both options. Usually, **Programmatic access** is preferred for authentication through an access key and a secret key while using APIs, SDKs, and the CLI. Ideally, for individual users, **AWS Management Console access** is selected. If you select **Programmatic access**, you can proceed to *step 7*. If you select **AWS Management Console access**, it displays more options on the same screen, as shown in the following screenshot:



Figure 3.13: Password configuration options while creating an IAM user

6. If you select **AWS Management Console access**, it allows you to configure an **Autogenerated password** or **Custom password** for the user. Also, the IAM administrator can force a user to reset the password on the next login by selecting options for this, as shown in the preceding screenshot. After selecting the required options, select the **Next: Permission** button.
7. You can now see a screen with three options to assign permissions to the user, as shown in *Figure 3.14*. You can create a group and add the user to a new group, or you can add the user to an existing group. It is recommended that you add a user to a group as a rule, for better user management and access control; however, it is not mandatory. A user can be added to any group in the future without any requirements:

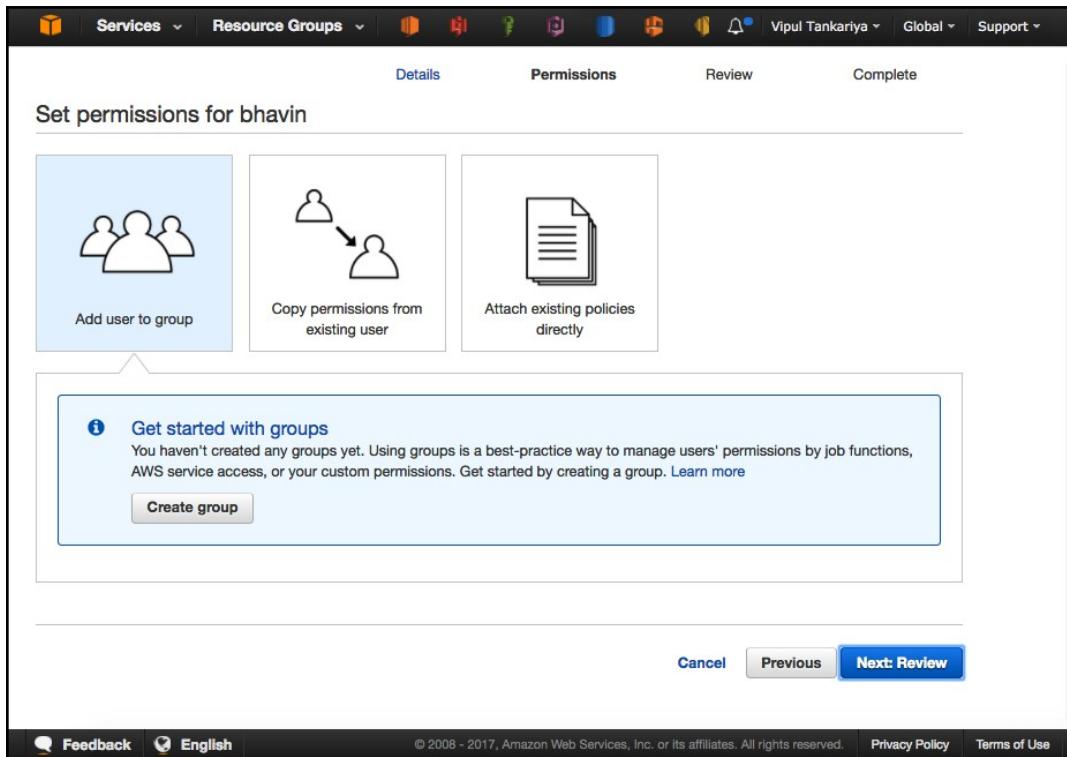


Figure 3.14: IAM – Adding user permissions

8. The next step is to assign permissions to the newly created user. To assign permissions to the user, you can either **Copy permissions from existing user** or **Attach existing policies directly**, as shown in *Figure 3.14*. By default, newly created users do not have any privileges on the AWS platform, until and unless the appropriate policy is attached to the user ID. Any permissions you grant to a user can be modified at a later stage. After adding the user to an appropriate group or policy, you can click on the **Next: Review** button.
9. Verify the details when the final review page appears. If there is any ambiguity, you can perform modifications by going to the previous pages. It is also possible to modify the user's property after it is created. In worst-case scenarios, an existing user can be deleted and recreated.
10. Finally, click on **Create** to create an IAM user.

Introducing the AWS CLI

Before you start using the AWS CLI, it is essential to set up a CLI environment. To set up a CLI environment, you need to install the AWS CLI utility based on the operating system in use by the system on which you want to set up the CLI. For Windows, you can install it with the MSI installer; for Unix/macOS, you can install it with a bundled installer or pip.

Installing the AWS CLI

AWS CLI installation on Windows operating systems is very easy using a step-by-step wizard with the MSI installer. This section describes how to install the AWS CLI on Linux and macOS. On Linux and macOS, the AWS CLI can be installed using pip, a package manager for Python. The minimum requirement to install the AWS CLI is to have the pip package manager and Python 2.6.5+ or Python 3.3+. You can find information on installing Python and pip at <https://docs.python-guide.org/starting/install3/linux/>.

Once the pip package manager is installed, the AWS CLI can be installed using the following command:

```
$ pip install --upgrade --user awscli
```



Based on the OS, specific methods can be used to install the Python pip package manager. In this example, all the commands are related to RHEL/CentOS.

The functionality of the AWS CLI utility is periodically updated by AWS to add support for recently added services and features. To update the installed AWS CLI instance, the same command can be used:

```
$ pip install --upgrade --user awscli
```

You can use the following command to uninstall the AWS CLI:

```
$ pip uninstall awscli
```

To make sure the AWS CLI is properly installed and to check the version, the following command can be used:

```
$ aws --version
```

It may display the following, or the appropriate information for the version installed on your system:

```
aws-cli/1.11.55 Python/2.7.12 Linux/4.4.41-36.55.amzn1.x86_64
botocore/1.5.18
pip install --upgrade --user awscli
```

Getting an AWS user access key and secret key

We will now discuss the steps for getting an AWS user access key and secret key:

1. Log in to the AWS Management Console with the appropriate credentials. An IAM user must have sufficient privileges to create IAM resources (that is, users, groups, policies, and so on).
2. Select **IAM** under the **Security, Identity & Compliance** group from the AWS dashboard. It will take you to the IAM dashboard.
3. Select **Users** and select the intended user for which to generate an access key and secret key.
4. Go to the **Security credentials** tab.
5. Click **Create access key** to generate an access key and secret key pair.
6. You need to ensure that the keys are stored safely, as AWS does not allow you to download these keys again.
7. Finally, click **Close**.

Configuring the AWS CLI

After installing the AWS CLI and obtaining the access key and secret key, we need to configure the CLI before we can start using it.



The AWS CLI uses the local-machine date and time as a signature when making calls to AWS. It is important to make sure that the machine's date and time are set correctly, otherwise AWS rejects any CLI request.

To configure the AWS CLI, the following command can be used:

```
$ aws configure
AWS Access Key ID [None]: AKIAJ4B7SOIHQBQUXXXXAWS
Secret Access Key [None]: oSpG3je8kYS1XpMDRG8kpo1awLizvnn1GaNBXXXX
Default region name [None]: us-east-2
Default output format [None]: ENTER
```

It will ask for four inputs—Access Key ID; Secret Access Key; Default region name, where it should request the command; and Default output format. The output format can be JSON, table, or text format. If the output format is not defined, the default output format will be JSON.

AWS CLI syntax

The AWS CLI supports commands for almost all AWS services. All the commands should be preceded with `aws` as described in the following syntax:

```
aws <top level command> <subcommands> <parameters>
```

Top-level commands indicate the AWS service name, such as `ec2`, `s3`, and `iam`, while subcommands are AWS-service-specific.

Getting AWS CLI help

It is possible to get detailed help for any of the AWS top or sub-level commands just by placing `help` at the end of the command, as shown in the following example:

```
$ aws help
$ aws iam help
```

Creating an IAM user using the AWS CLI

The AWS CLI `create-user` subcommand with the `iam` top command can be used to create a new IAM user. The following AWS CLI command shows how to get more details about subcommands:

```
$ aws iam create-user help
```

A new IAM user can be created using the following command. The essential parameter is `--user-name`:

```
$ aws iam create-user --user-name Jack
```

Groups

In an organization, people work in different departments (such as sales, purchase, IT, and so on). Usually, only members from the IT department need to access AWS resources. But it depends on the nature of the organization and its organizational hierarchy. In each department, there can be sub-departments (for example, in IT, there can be many branches, such as development, testing, operations, quality, security, and network). Each sub-department may have several people working in it. An organizational hierarchical structure looks something like the following:

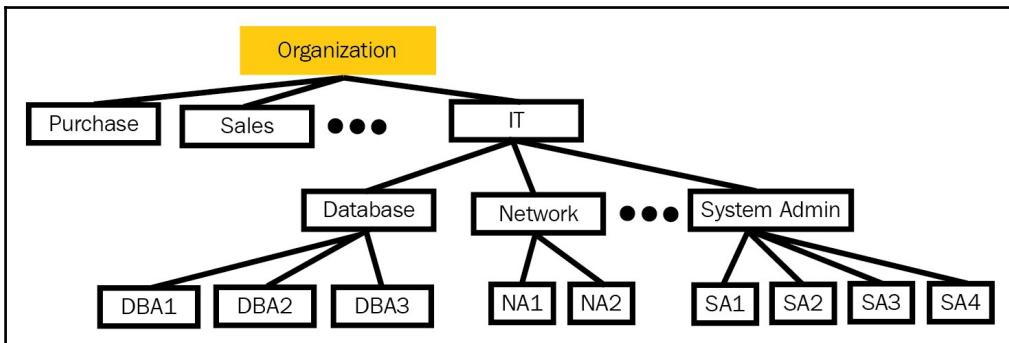


Figure 3.15: A logical representation of organizational entities

It is easy to manage privileges for a few users individually, but it becomes increasingly difficult to manage these users separately as the userbase increases. Most of the time, when users belong to the same department with identical or similar roles and responsibilities, their privileges requirement may also be the same. In such scenarios, it is recommended that you divide the users into logical groups and assign privileges to a single group, instead of individual users. A similar concept is used in IAM groups.

In simple terms, an IAM group is a collection of IAM users. A group lets you add, change, or remove permissions for multiple users at once.

You should assign privileges (that is, policies) to a group, rather than handling privileges at the individual user level. Some users who are part of a group may require extra privileges to perform advanced tasks. For such users, separate policies can be attached at the user level. When multiple policies are attached to any user or a group, such users or groups get all the permissions from the attached policies. The user and group management of a small company is illustrated in the following diagram:

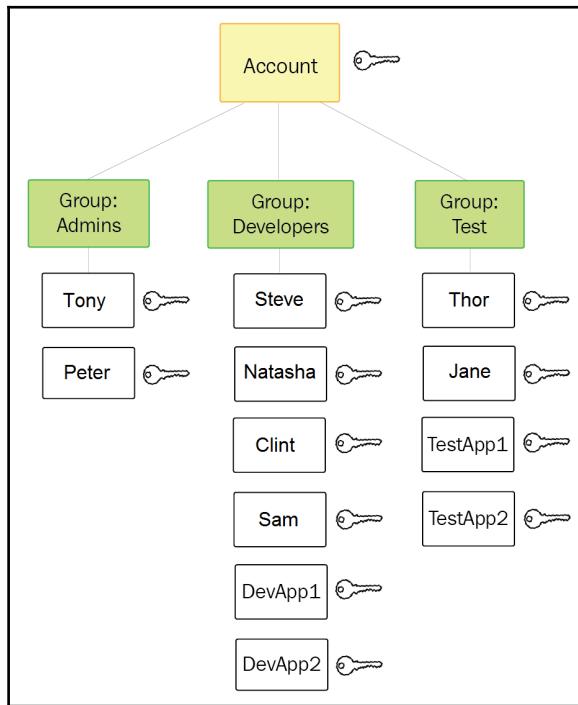


Figure 3.16: A logical representation of groups and users in a small enterprise

As we have seen so far, groups are primarily used to assign specific permissions to a set of users. When any user is added to a group, the user automatically inherits all the permissions granted to that group.

Whenever a new employee joins an organization, the process of onboarding and granting permissions to that employee becomes very easy if the organization follows the concept of user groups. Just by creating an IAM user and adding that user as a member of a group, the user automatically inherits all the privileges assigned to that IAM group. When the same employee either changes department within the organization or leaves the organization, managing the employee's privileges becomes easier, and can be done simply by removing the user from the respective group.

The following are the important characteristics of an IAM group:

- Any group can have many users, and a user can be a member of multiple groups.
- Groups can't be nested (that is, groups within groups); they can contain only users, not other groups.
- By default, users are not part of any group. As and when required, you need to explicitly add them to the required groups.
- There is a soft limit of a maximum of 300 IAM groups and 5,000 users. If more users are required, it is advisable to use the identity federation service to create temporary security credentials.

Creating a new IAM group

The steps for creating IAM groups are as follows:

1. Log in to the AWS Management Console with the appropriate credentials. An IAM user must have sufficient privileges to create IAM resources (that is, users, groups, policies, and so on).
2. Select **IAM** under the **Security, Identity & Compliance** group from the AWS dashboard. This will take you to the IAM dashboard.
3. Select **Groups** and click **Create New Group**.
4. Provide a meaningful group name. Group names can have a maximum of 128 characters. Click **Next Step**.
5. Attach the required IAM policies to the IAM group. When policies attached to a group are modified, all the existing and future members of the group inherit the updated privileges. It is always best practice to grant the minimum privileges required to a user or a group.
6. At this stage, review your group settings prior to creation. If everything is fine, click on **Create Group**.

Creating an IAM group using the CLI

A new IAM group can be created using the following command. The essential parameter is `--group-name`:

```
$ aws iam create-group --group-name Developers
```

Adding existing users to a group

The following steps describe how to add an existing user to one or more groups:

1. Log in to the AWS Management Console with the appropriate credentials. An IAM user must have sufficient privileges to create IAM resources (that is, users, groups, policies, and so on).
2. Select **IAM** under the **Security, Identity & Compliance** group from the AWS dashboard. This will take the user to the IAM dashboard.
3. Select **Existing user** from **Users**.
4. Select the **Groups** tab.
5. Click **Add user to groups**.
6. A list of existing groups will appear. Select the appropriate group. It is possible to select one or more groups. One user can be a member of one or more groups, and inherits all the permissions from those respective groups.
7. Finally, click on **Add to Groups**.

IAM role

An IAM role is an AWS identity. Every IAM role has its own permission policy that defines what that role can do and what it cannot do. It is like an IAM user, only without a password or an access key and a secret key. An IAM policy can be associated with an IAM user or group, while an IAM role cannot be associated with a user or a group. It can be assumed by a user, application, or service to delegate access to an AWS resource within the same or another account. It dynamically generates a temporary access key and secret key, which can be assumed by an entity for authentication. Once a role is assumed, an entity can make API calls to AWS services to which access is permitted to the role assumed by the entity.

For example, a role can be assigned to an EC2 instance with permission to access DynamoDB and RDS databases. An application hosted on the EC2 can assume the role and make API calls to access DynamoDB or databases on RDS.

Similarly, if you want to allow your web or mobile application to access AWS resources, but you don't want to hardcode an access key and secret key in the application code, an IAM role can come to the rescue. IAM roles can also be used to provide federated access to AWS services using Microsoft **Active Directory (AD)**, LDAP, or similar identity providers. In subsequent sections, we will get into the details of these aspects.

In a nutshell, AWS resource permissions, in the form of IAM policies, are attached to the IAM roles rather than being attached to IAM users or groups. IAM roles can be assumed by the following:

- An IAM user in the same AWS account
- An IAM user in a different AWS account
- AWS web services (for example, EC2)
- External user authentication software that uses an external **identity provider (IdP)** that is compatible with **Security Assertion Markup Language (SAML) 2.0** or **OpenID Connect (OIDC)**, or a custom identity broker

Let's start by understanding some of the important terminology with respect to IAM roles. These terms are as follows:

- Role
- AWS service role
- AWS service role for an EC2 instance
- AWS service-linked role
- Role chaining
- Delegation
- Federation
- Policy
- Trust policy
- Permissions policy
- Permissions boundary
- Principal
- Role for cross-account access

Each of these concepts is explained individually as follows:

- **Role:** A role is a set of permissions that grant access to perform specific actions on one or more resources. Rather than attaching set of permissions to the IAM users or groups, it is attached to the IAM role. Roles can be assumed by the following:
 - An IAM user in the same AWS account
 - An IAM user in a different AWS account
 - By AWS services such as EC2 or RDS
 - An external user authenticated by an external IdP service that is compatible with SAML 2.0 or OpenID Connect, or a custom-built identity broker
- **AWS service role:** These types of IAM roles are assumed by AWS services to perform actions in your AWS account on your behalf. Before using some of the AWS services, the environment may require you to define such AWS service roles. They are documented in AWS service documentation. It can be customized to match with the enterprise compliance requirements, but it must include all the required permissions for the service to access the AWS resources that it needs. Such service roles vary from service to service, and can be used to grant permissions only within the AWS account and not across the AWS accounts.
- **AWS service role for an EC2 instance:** This is a special type of service role, which can be attached to (that is, assumed by) an EC2 instance. At the time of creating an EC2 instance, or throughout its life cycle at any time, such a role can be attached to or detached from the EC2 instance. Ideally, it is best practice to attach appropriate roles to an EC2 instance at the time of the launching the instance, as AWS creates temporary security credentials that are attached to the role, and after that, makes them accessible for the EC2 instance and uses them to the benefit of its applications. Consider an example: a deployed application on EC2 requires access to the S3 bucket. In this case, rather than embedding an access key and secret key to the EC2 instance, it is best practice to create an AWS service role for an EC2 instance with sufficient privileges to access the S3 bucket, and attach this role to the EC2 instance.

- **AWS service-linked role:** This unique role is directly linked to the AWS service. The roles are defined in advance and include all the permissions required by the service to call other AWS services. A service might automatically create or delete the role. It might allow you to create, modify, or delete the role as part of a wizard or process in the service. Regardless of the method, setting up a service is easy with service-linked roles as you do not have to add the required permissions manually. AWS services, such as application Auto Scaling, Amazon EC2 Auto Scaling, and AWS Lambda, use such service-linked roles.



More details about the AWS services can be obtained from the official documentation, at https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_aws-services-that-work-with-iam.html.

- **Role chaining:** This occurs when a second role is assumed through the AWS CLI or API. A very good example on role chaining can be found in the official documentation at https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_terms-and-concepts.html. The primary purpose of role chaining is to provide a maximum of a one-hour session for AWS CLI or API requests. In general, when using the AssumeRole API operation to assume a role, it is possible to specify the duration of the role session with the DurationSeconds parameter. This value is defined in seconds and can be up to 43,200 seconds (that is, 12 hours). However, if you assume a role using role chaining, and provide a value for the DurationSeconds parameter as greater than one hour, then the operation fails.
- **Delegation:** Delegation is a way to extend an entity's permission on AWS resources to other users or applications, allowing them to perform certain operations on the resources. It involves creating a trust between the account where the AWS resources are hosted and the account that contains the user that needs to access these resources. The source account where the AWS resources are available is called the *trusting account*, and the account from where the user wants to access those source resources is called the *trusted account*.

Trusting (source) and trusted (destination) accounts can be the following:

- The same AWS account
- Two different accounts managed by the same organization
- Two different accounts managed by different organizations

To delegate permission, you need to attach two policies to the IAM role. One policy defines the permissions to be given and another is a trust policy that defines trusted accounts that are allowed to grant permission to the user to assume the role.

- **Federation:** Identity federation is a mechanism through which applications can use external **Identity Providers (IdPs)** for authenticating users rather than writing custom sign-in code for authenticating the users. These external IdPs include Amazon, Facebook, Google, or any IdP that is compatible with **Open ID Connect (OIDC)**, MS AD, or LDAP, and any that support SAML 2.0 to configure token-based authentication mechanisms between external IdPs and AWS-hosted applications. Web identity federation is explained in more detail in a subsequent section.
- **Policy:** A policy is a JSON-formatted document written in line with IAM policy notation. It defines the permissions to be granted by an IAM role. Policies can also be written to attach policies to IAM users and groups. IAM roles have the following two types of policies attached to them:
 - **Trust policy:** This policy defines the entities that can assume a role. Each role can have only one trust policy. A trusted entity can be one of a number of possible AWS services, including EC2, EMR, an EC2 container service, RDS, Lex, and Glue.
 - **Permissions policy:** The resources and actions a role can perform on those resources are defined by the permission policies. These policies can be AWS-managed policies, customer-managed policies, or inline policies attached directly to the role. More information can be found at https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_managed-vs-inline.html.
- **Permissions boundary:** This is an advanced feature that makes it easier to delegate permissions management to trusted employees. Usually, when an organization has seen growth, a situation may arise where trusted employees can be allowed to configure and manage IAM permissions in order to make permissions management faster. An IAM administrator can create one or more permissions boundaries using managed policies to allow your employees to create a principal with this boundary. Permissions boundaries can be applied to AWS organizations, IAM users, or roles. They cannot be applied to a service-linked role.

- **Principal:** This is an element generally used in a policy to denote a user (that is, an IAM user, federated user, or assumed role user), AWS account, AWS service, or other principal entity that is allowed or denied access to a resource. Specified users are allowed or denied access to perform actions on AWS resources.
- **Role for cross-account access:** When AWS resources existing in one account are accessed from another account based on a trust relationship, it is called cross-account access. IAM roles are the primary way to grant cross-account access.

As we have gone through the various elements of IAM roles, let's look at how we can create a role.

Creating roles for an AWS service

To create a role for an AWS service, perform the following steps:

1. Log in to the AWS Management Console with the appropriate credentials. An IAM user must have sufficient privileges to create IAM resources (that is, users, groups, policies, and so on).
2. Select **IAM** under the **Security, Identity & Compliance** group from the AWS dashboard. This will take the user to the IAM dashboard.
3. Select **Roles** from the IAM dashboard.
4. Select **Create New Role**, as shown in the following screenshot:

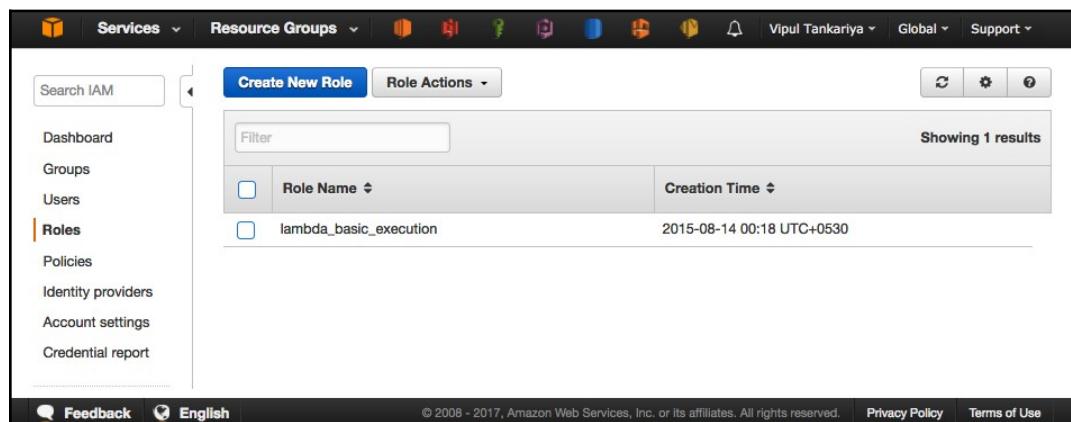


Figure 3.17: IAM – the Create New Role screen

5. Provide a meaningful role name with a maximum of 64 characters, as shown in the following screenshot:

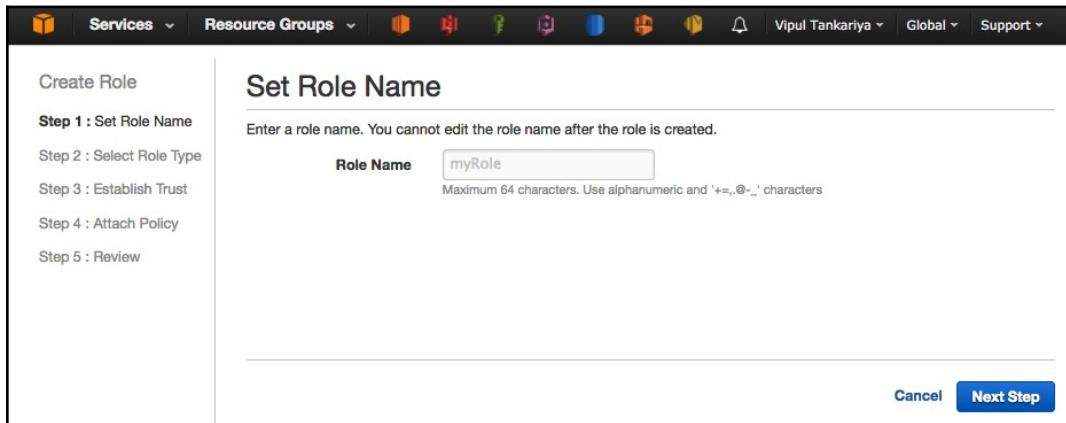


Figure 3.18: IAM – Setting the Role Name

6. Select role type as **AWS Service Roles**, as shown in *Figure 3.18*:
- **AWS Service Roles** are assigned to AWS resources such as EC2, RDS, and Redshift. This grants them privileges to perform various operations on required AWS services, based on permissions granted on the role.
 - **Role for Cross-Account Access** is used for establishing a trust relationship between multiple AWS accounts.
 - **Role for Identity Provider Access** is used by external IdPs for federated authentication.

7. Subsequent steps appear based on the selection in the previous step. Since we are exploring the role for AWS service (EC2), select **AWS Service Roles** and click on **Select** against Amazon EC2, as follows:

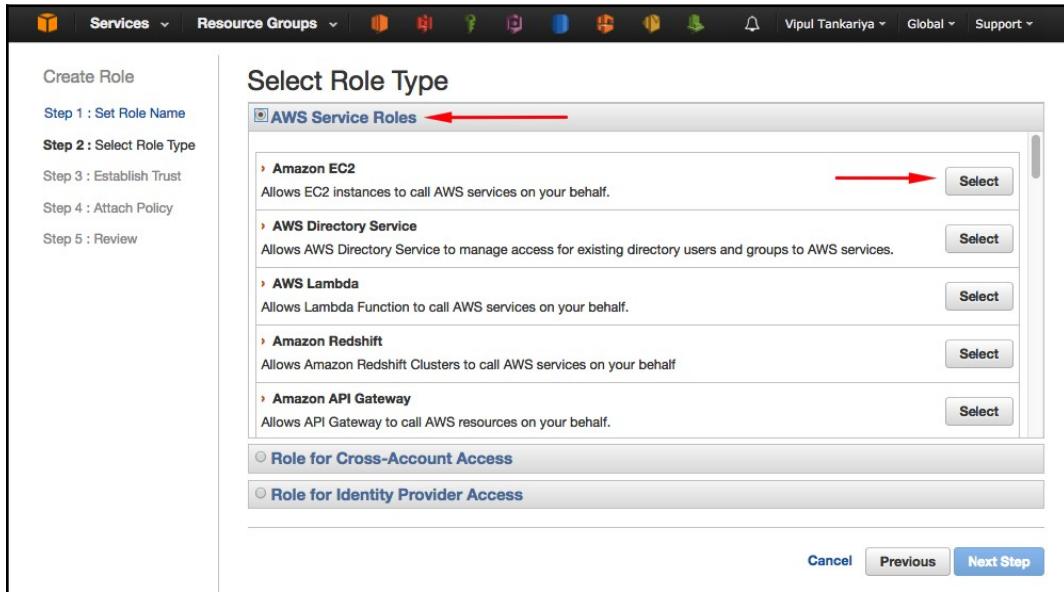


Figure 3.19: IAM role settings – selecting the role type

8. **Attach Policy**, as per the permissions required by the application hosted on EC2. The policy may contain permissions for accessing an S3 bucket, RDS, DynamoDB table, or any other AWS services as per the application requirements. Policies can be selected on screen, as shown in the following screenshot:

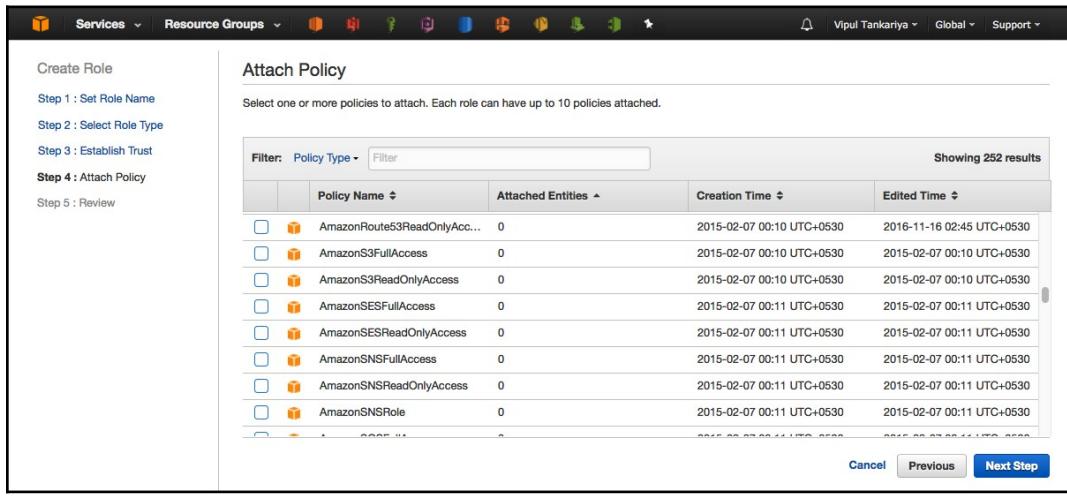


Figure 3.20: Creating an IAM role – attaching the policy

9. Finally, review and click on **Create Role**, as shown in the following screenshot. This is the concluding step in creating an IAM role for an AWS service. You can assign this role to an EC2 instance while launching a new instance:

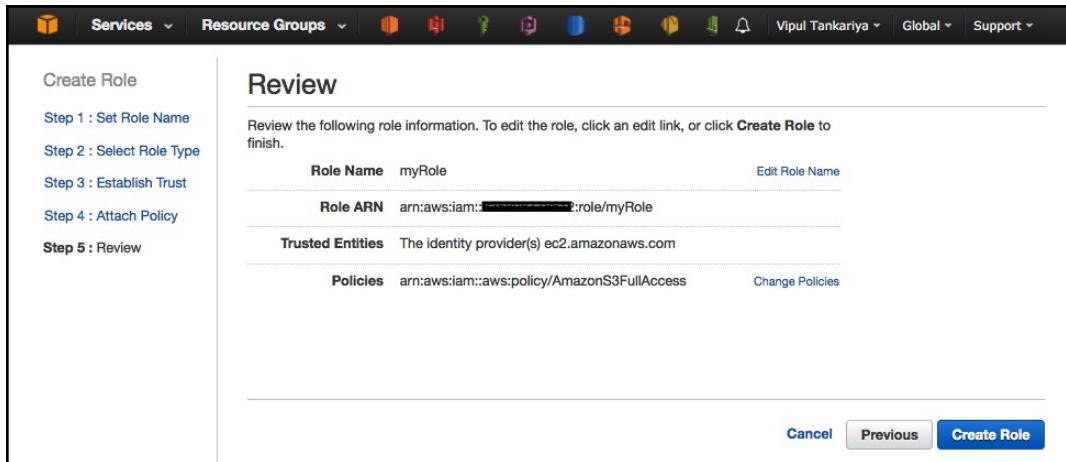


Figure 3.21: Creating an IAM role – reviewing the settings

Creating IAM roles using the AWS CLI

A new IAM role can be created using the following command; the essential parameters are `--role-name` and `--assume-role-policy-document`, which define a JSON-documented policy:

```
$ aws iam create-role --role-name Test-Role --assume-role-policy-document file://Test-Role-Trust-Policy.json
```

Policy

A policy is a document that formally states one or more permissions. Policies are written to explicitly allow or deny permission to access one or more AWS resources. Policies can be associated with one or more IAM users, groups, roles, or resources, based on their type.

IAM policy types based on their day-to-day usage are as follows:

- **Identity-based policies:** These policies include user-managed and inline (that is, directly embedded) policies for IAM identities, such as users, groups, and roles.
- **Resource-based policies:** These policies also include user-managed and inline (that is, directly attached) policies. But, rather than being attached to the identity, they are attached to the resource in question, such as an S3 bucket or IAM role trust policies.
- **Organizations SCPs:** With the help of AWS organizations **service control policy (SCP)**, apply a permissions boundary to an AWS organization or **organizational unit (OU)**.
- **Access Control Lists (ACLs):** ACLs are used to control the principal's capability to access AWS resources. These are similar to the resource-based policies, the only difference being that ACLs don't use the JSON-policy-document structure.

ACLs and identity-based and resource-based policies are called **permissions** policies. They are attached to an object in a given AWS account to define the permissions for the object.

Permissions boundaries define the principal's boundaries. They control the maximum permissions that a principal can have. AWS organizations SCP and permissions boundary falls under this category.

Broadly, IAM policies can be classified as follows:

- Managed policies:
 - AWS-managed policies
 - Customer-managed policies
- Inline policies
- Resource-based policies

Managed policies

Built-in policies that are managed by AWS or policies that are created and managed by customers are called **managed policies**. These policies can be attached to multiple users, groups, and roles. Managed policies cannot be attached to resources. Managed policies are further classified as follows:

- **AWS-managed policies:** As the name suggests, these are built-in policies that are created and managed by AWS. They are also updated from time to time, and updates are automatically applied to the attached IAM principal entities.
- **Customer-managed policies:** These are policies that are created and managed by customers in their AWS account. These policies can be updated by customers as and when required. The effects of such changes are applied immediately to the principal entities to which the policies are attached.

The main difference between these two policy types is that AWS-managed policies are generic, while customer-managed policies are precisely in line with actual permission requirements. The similarity between AWS-managed and customer-managed policies is that an amendment to the policy doesn't overwrite it; IAM creates a new version of the managed policy every time it is updated. A managed policy can have up to five versions. Beyond that, it is necessary to delete one or more of the existing versions.

Inline policies

Inline policies are also customer-managed policies, except that these policies have a one-to-one relationship between policies and principal resources. These policies are created and managed to be directly attached only to a single user, group, or role. Such policies are useful to make sure that permissions in a policy are precisely granted as per organizational requirements. Such policies are also automatically deleted when an underlying resource is deleted using the AWS Management Console. In contrast to customer-managed policies, changes are immediately applied to principal resources where policies are attached.

Resource-based policies

Resource-based policies are also an inline policy type, as they are written inline for attachment to a particular resource. Not all AWS services support resource-based policies. At present, only S3 buckets, SNS topics, Amazon Glacier vaults, AWS OpsWorks Stacks, AWS Lambda functions, and SQS queue support resource-based policies.

Now that we understand the different types of policy, let's find out how to write a policy. Every policy is JSON-formatted and carries at least one statement. Usually, a policy consists of multiple statements to grant permission on different sets of resources. The following is an example of a basic customer-managed policy:

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



An **Amazon Resource Name (ARN)** is a unique identifier for each AWS resource. It is used in IAM policies, API calls, and wherever it's required to identify AWS resources unambiguously.

A basic example of an ARN is as follows:

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

Here, the following terms are used:

- **partition:** This specifies which AWS partition the AWS resource belongs to. Various AWS partitions are as follows:
 - aws: Public AWS partition
 - aws-cn: AWS China
 - aws-us-gov: AWS GovCloud

- **service**: Specified AWS service name (that is, EC2, S3, IAM, and so on) region, specifies the AWS region where the resource resides. As some AWS services are global, such as IAM, the ARN for such resources doesn't have a region.
- **account-id**: Specifies the 12-digit AWS account number.
- **resource, resourcetype:resource , or resourcetype/resource**: This part of the ARN varies from service to service, as some services allow paths for resource names (that is, slash (/) or colon (:)).

Examples of some ARNs are as follows:

- The ARN for an EC2 resource is formatted as such:

```
arn:aws:ec2:us-east-1:123456789012:dedicated-host/h-12345678
```

- The ARN for an IAM role looks like this:

```
arn:aws:iam::123456789012:role/application_abc/component_xyz/  
S3Access
```

The preceding customer-managed policy allows us to perform any possible operations on the S3 example_bucket. By attaching this policy to the IAM entity (user, group, or role), it gets permission to read, write, delete, or perform any possible operations on the specified S3 bucket. Various elements of IAM policy are explained as follows:

- **Version**: This element specifies the IAM policy language version.



The latest and current version is 2012-10-17. It should be used for all the policies (that is, managed or resource-based policies). For inline policies, the version element can be 2008-10-17, but it is highly recommended to keep it as 2012-10-17.

- **Effect**: The effect element either allows or denies actions on the specified resources. It defines whether the list of actions, specified in action elements against the resources mentioned in the Resource elements, are allowed or denied. By default, every service and resource is denied access. Usually, policies are written to allow resource access.

- **Actions:** This defines a list of actions. Each AWS service has its own set of actions. As per the policy written for the resource, this list of actions varies.
- **Resources:** This section specifies a list of resources on which the preceding specified list of actions is allowed.



The major difference between a managed policy and a resource-based policy is that a resource-based policy specifies who has access to the resource (principal) and the list of permitted actions, while in a managed policy, only a list of actions is specified, and **not** the principal entity.

An IAM resource-based policy can also be generated with the help of the AWS policy generator. The URL for the AWS policy generator is <https://awspolicygen.s3.amazonaws.com/policygen.html>.

An example of a resource-based policy can be found at <https://docs.aws.amazon.com/IAM/latest/UserGuide/iam-ug.pdf>.

IAM policy simulator

Writing an IAM policy can be a lengthy and error-prone process. There might be a few human or logical errors that appear while writing a policy. It can be a very time-consuming and tedious process to find and rectify such policy errors. An IAM policy simulator provides a platform to simulate and test policies before using them in an AWS account. Using the IAM policy simulator, an existing AWS or custom managed policy can be copied and modified as required. This makes writing new policies easier. The IAM policy simulator uses the same engine as used in an AWS account to evaluate policies. The only difference is that the IAM policy simulator is safe to test, as it doesn't make actual AWS service requests; it just simulates them. The policy simulator can be accessed at <https://policysim.aws.amazon.com>.

Active Directory Federation Service (AD FS)

As you have seen in this chapter so far, you can access AWS resources either using an IAM user ID and password or using an access key and secret key combination. Let's consider a scenario in which a user uses IAM credentials to access AWS resources and an AD user ID and password to access resources hosted within the on-premises environment. In such a scenario, in an organization where there are a number of users, it becomes increasingly difficult to maintain credentials in multiple systems. Users and operations teams have to maintain user details not only on their organization's identity provider, but also on IAM. To cater to such scenarios, IAM also supports identity federation for delegated access to the AWS Management Console or AWS APIs. Along with identity federation, secure access to resources in your AWS account is granted to external identities such as federated users. This is done without us having to create any IAM users. You can access your AWS resources with your organization's AD services using a method called Directory Federation (AD FS).

Let's take a look at a couple of scenarios demonstrating what you can do with AD FS with AWS:

- You can access the AWS Management Console by authenticating against your organization's AD instead of using your IAM credentials. Once authenticated against AD, AWS generates temporary credentials for the user and allows access on the AWS console.
- You can access a web application hosted on AWS by authenticating against your organization's AD. Once authenticated against AD, it provides **single sign-on (SSO)**, which provides users with seamless access to applications without re-prompting for credentials after initial authentication.

Such a setup that can authenticate as an identity provider and communicate with AWS is achieved using Windows AD, ADFS, and SAML. Let's understand these terms before getting into configuration details for such a setup:

- All the user accounts and passwords in your organization are tracked by Windows AD.
- AD FS is part of Windows AD services in the form of a web service that provides a token-based SSO service for accessing systems and applications located across organizational boundaries.
- SAML is an XML-based, open-standard data format for exchanging authentication and authorization data between an IdP and a service provider.

Integration between AD FS and the AWS console

Now that we know the terminology for enabling ADFS with AWS, let's take a look at how ADFS authentication works to access the AWS console:

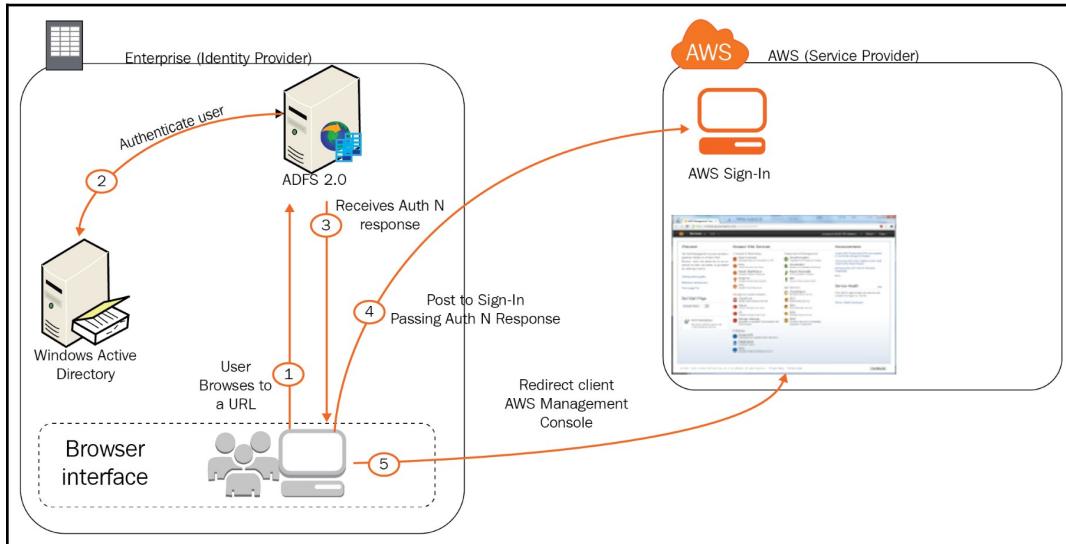


Figure 3.22: Accessing the AWS console

As indicated in *Figure 3.21*, accessing the AWS console using ADFS is a five-step process. These steps are described here:

1. A user initiates a request to an AD FS web URL in an internet browser. This URL may look something like `https://HostName/adfs/ls/IdpInitiatedSignOn.aspx`. When you enable an AD FS role on a Windows instance, it creates a virtual directory named `adfs` in your default site. The sign-on page is automatically created under the `adfs` virtual directory. This page provides an interface to enter a user ID and password to authenticate against the site.
2. When a user submits the credentials, AD FS authenticates the user against Windows AD.
3. In the third step, the ADFS sends a SAML assertion in the form of an authentication response to the user browser.

4. The SAML assertion is posted to the AWS sign-in endpoint by the user browser for SAML (<https://signin.aws.amazon.com/saml>). Behind the scenes, the AssumeRoleWithSAML API is used to request temporary security credentials, and then a sign-in URL is constructed for the AWS management console.
5. The sign-in URL is received and redirected to the console by the user browser.

The preceding details are sufficient to understand AD FS as far as the scope of the exam is concerned. If you need more details about setting up and configuring AD FS with an AWS console, you can refer to the AWS documentation at <https://aws.amazon.com/blogs/security/enabling-federation-to-aws-using-windows-active-directory-adfs-and-saml-2-0/>.

Web identity federation

When you create a web application or a mobile application, creating a user repository and authenticating users against the repository is one of the core tasks of the application development life cycle.

If you are creating a mobile application and that mobile application needs access to AWS resources such as S3 and DynamoDB, how would you enable the application to access these AWS resources? One way is to use an access key and a secret key in the application that provides access to S3 and DynamoDB. However, the application would work with such an approach, but it is not recommended to embed or distribute long-term AWS credentials to an app that a user downloads to a device. Even if these credentials are stored in an encrypted format on the device, it would pose a security risk.

In such a scenario, AWS recommends using web identity federation. By using web identity federation, applications can request temporary security credentials dynamically when required. The credentials are generated based on an AWS role that carries permissions to perform only required operations on specific AWS resources, as permitted by the role.

Web identity federation enables you to use many well-known identity providers, such as Amazon, Facebook, Google, LinkedIn, or any other OIDC-compatible IdP for authenticating users. You don't need to create your own custom sign-in code or manage your own user identities.

The application can authenticate a user against such an IdP. If a user is authenticated, the application receives an authentication token. This token is exchanged in AWS for temporary security credentials. Since these temporary security credentials are based on an AWS role, the token carries permission only to perform specific tasks mentioned in the role permissions. Using web identity federation, you don't have to embed or distribute credentials with applications, and you thus keep your AWS account secure.



A perfect example for a sample workflow using web identity federation can be found at <https://aws.amazon.com/blogs/aws/aws-iam-now-supports-amazon-facebook-and-google-identity-federation/>

AWS recommends using Amazon Cognito with web identity federation instead of writing custom code for authentication. You can easily sign up or sign in to your mobile and web applications using Amazon Cognito. You can authenticate users through social IdPs such as Facebook, Twitter, or Amazon; with SAML identity solutions; or by using your own identity system with Amazon Cognito.

The following diagram shows a sample workflow using web identity federation with Cognito:

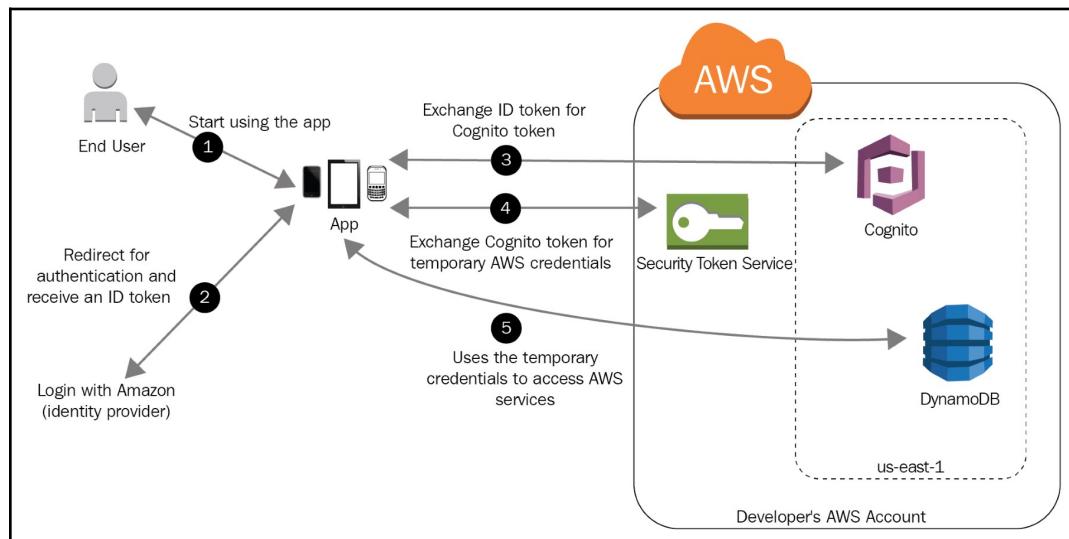


Figure 3.23: A workflow using web identity federation with Cognito

1. As you can see in the preceding diagram, an end user starts your app on a mobile device. The application prompts the user to sign in.
2. When a user enters the credentials and submits the authentication request, the application redirects it to a **Login with Amazon** third-party IdP for verifying the credentials. If the user is authenticated, they receive an authentication token.
3. The application forwards the authentication token to Cognito using Cognito APIs and exchanges the authentication token for a Cognito token.
4. The application passes the Cognito token to AWS STS to request temporary security credentials. AWS STS provides temporary security credentials.
5. The app uses temporary security credentials for accessing AWS resources required to operate. Which resources can be accessed is determined by the policies assigned to the role, along with any temporary security credentials associated with the role.

Security Token Service (STS)

STS is a web service that enables an application to dynamically generate *temporary security credentials* with restricted permissions based on an IAM role. These temporary credentials can be generated either for an IAM user or for a federated user, as we have seen in the *Web identity federation* section.

Temporary security credentials generated using AWS STS for a trusted user can control access to your AWS resources. Temporary security credentials and the long-term access key credentials used by IAM users work in almost the same way, except for a few differences:

- Temporary security credentials, as the name suggests, are for short-term use only. These credentials expire after a specific time.
- Temporary security credentials can be configured to expire from within a few minutes to several hours.
- AWS does not recognize the credentials after they expire. API requests for access made with expired credentials are not allowed.
- Temporary security credentials are generated dynamically and not stored with the user.
- They are provided to the user or application based on the request.
- New credentials should be requested before old ones expire. Once your temporary security credentials expire, you can request new credentials as long as you still have the permission to do so.

These differences lead to a few advantages over using temporary credentials, which can be found at https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_temp.html.

Using temporary security credentials to request access to AWS Resources (https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_temp_use-resources.html) can be done in the following scenarios:

- Using temporary credentials in Amazon EC2 instances
- Using temporary security credentials with the AWS SDKs



For more information on AWS STS and AWS regions, please visit https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_temp.html

AWS account ID and alias

A user's AWS account ID and alias play vital roles in the user login process. While an account ID is generated automatically when a new user is registered, the alias provides a login page URL. Both these topics are explained in this section.

AWS account IDs

Every AWS account has a unique 12-digit account number. It can be obtained by clicking on **Support Center**, under the **Support** drop-down menu situated at the right-hand side of the top menu bar. A 12-digit account number will appear on the screen along with the support plan.



The appearance and availability of such information may vary from time to time without any prior notice.

AWS account aliases

The general URL for IAM users to sign in to the AWS account includes a 12-digit account number as follows:

`https://Your_AWS_Account_ID.signin.aws.amazon.com/console/`

Sometimes, it may be difficult for users to remember such long numbers. To make it more user-friendly, it is possible to create an account alias. Once the AWS account alias is created, the sign-in page URL looks like the following:

`https://Your_Alias.signin.aws.amazon.com/console/`

It is important to know that, even after creating an AWS account alias, it is possible to use the original URL containing the 12-digit account number. At any given time, only one alias can be created for the AWS account.

The steps to create an AWS account alias are as follows:

1. Sign in to the AWS Management Console and navigate to IAM console at `https://console.aws.amazon.com/iam/`.
2. Choose **Dashboard** from the navigation pane.
3. You can find the **IAM users sign-in link** on the dashboard. Click on **Customize** to the right of the link.
4. It provides you the option to type the name you want to use for your alias. After typing in the desired name for your alias, you can click on **Yes, Create**.
5. Similarly, for removing the alias, you can click on **Customize**, and subsequently click **Yes, Delete** to delete the alias. When you delete the alias, the sign-in URL reverts to the default URL of your AWS account ID.

Controlling user access to the AWS Management Console

Access to the AWS Management Console is controlled by the IAM policy associated with a user. The following list describes a few important points for controlling user access to the AWS Management Console:

- IAM users can only access AWS resources to the extent of the permissions granted to them.
- AWS IAM is a free and global (that is, region-independent) service.
- To sign in to the AWS management console, a password must be configured for each IAM user.
- By default, IAM users do not have any privileges to perform any action in the AWS account.
- To enable IAM users to perform day-to-day activities, it is essential to attach a suitable policy.

- When many users are required to have the same level of privileges, it is recommended to create a common IAM group and attach a policy to the group, rather than creating policies for each individual user.
- Any IAM user can read the posts on the AWS Discussion Forums. When a user posts for the first time, they are prompted to give the nickname and email address to be used exclusively by that user in the AWS Discussion Forums.
- Optionally, permission can be granted to access AWS account billing and usage information for a given IAM user.

It is also important to understand the authentication mechanism associated with the various methods through which AWS resource can be accessed. The following table describes this:

AWS resources access methods	Authentication mechanism
Web console	Username and password.
CLI	Access key and secret key.
API/SDK	Query request over HTTP or HTTPS required to include Signature Version 2 Sign-in Process and Signature Version 4 Sign-in Process. It is recommended to use Signature Version 4 Sign-in Process.
Federated users	With the help of an IAM role.
IAM roles	Temporary security credentials are dynamically created with the help of AWS STS and assigned to a user who assumes a role.

IAM best practices

The security of your AWS resources can be maintained by following these best practices:

- Never share credentials (that is, the password or access key and secret key). Specifically, sharing root user credentials can pose a very serious security threat, as they carry the highest level of access in the relevant AWS account.
- Never use the root account for day-to-day tasks. Create individual IAM users for designated roles and responsibilities.
- Until and unless it is essential, do not create an access key and a secret key. Also, keep rotating the password and keys periodically.

- It is not best practice to hardcode the access key and secret key in any program or application.
- Keep your access key and secret key secured so that they do not fall into the hands of any unauthorized person. A secret key is only generated once paired with a relevant access key. If a secret key is lost, then there is no mechanism in AWS to retrieve it. You need to discard the existing key and generate a new one. When a secret key is lost and discarded, the user must update the new key pair in the application, API, SDKs, CLI, and wherever the old key pair is used. Updating the new key pair ensures the smooth functioning of any relevant applications, programs, and services.
- Periodically remove unused IAM accounts, access keys, and secret keys. Implement a reasonably strong password policy to avoid compromising user passwords. A password policy can be configured from **Account settings** in the IAM dashboard.
- Implement MFA for all users and possible SDKs.
- Always grant the lowest level of privileges to the users. Only grant the essential permissions required for users to perform their day-to-day tasks. While inspecting privileges for the user, group, role, and policy, accessing the **Advisor** tab from the IAM dashboard can help.
- To grant permissions, use groups rather than applying the permissions at user level. Groups make it easier to manage permissions. If a user requires extra permissions, a separate policy can be attached to a specific IAM user based on need.
- Periodically audit existing users, groups, policies, and roles. Remove unwanted privileges from policies. Remove unused users, groups, and roles.
- To provide credentials to an application running on EC2, create an IAM role and attach it to an EC2 instance. Roles don't have a username and password or an access key and secret key. Temporary credentials are dynamically generated for roles, and such temporary credentials are automatically rotated.
- Apply policy conditions for an extra layer of security. For example, conditions can be specified for an allowed range of IP addresses.
- Monitor AWS account activity (that is, the creating, deleting, accessing, and modifying of resources) using various AWS services such as Amazon CloudTrail, Amazon CloudWatch, and AWS Config.

- It is suggested that you customize the IAM user sign-in link with an easy-to-remember name as it is used by AWS IAM users to log in to AWS. The sign-in link contains a 12-digit account number in the URL, which may be difficult for users to remember. It can be customized with a meaningful and unique name.
- AWS recommends that you use managed policies where possible. The main advantage of using these policies is that they are maintained and updated by AWS, as and when new services or API operations are introduced. However, you need to exercise caution while using managed policies, as they are generally broad in nature. You should always adopt least privilege policies. That means that you should ensure that a policy contains only essential privileges.

Exam tips

The following are some exam tips for AWS IAM:

- The AWS IAM service is a global service. This means it is not region-specific. IAM entities such as users, groups, roles, and policies are the same across all regions. Once they are created, they are the same for all AWS regions.
- By default, newly created IAM users do not have any privileges to perform any tasks on AWS accounts. Users must be granted permission to access any service or perform any operation in AWS. User permissions are granted by either adding the user to a group with required permissions or by directly attaching an access policy to a user.
- IAM users can be a member of any IAM group, but an IAM group cannot be a member of any other IAM group. In other words, an IAM group cannot be nested.
- One user can be part of multiple policies and multiple policies can be attached to a single user.
- An IAM user password is used for an AWS dashboard login, and an access key and secret key pair are used for API, CLI, and SDK authentication. It is not possible to use these credentials for other AWS services.

- By default, for any IAM users, groups, or roles, permission to access any AWS resource is denied, until and unless explicitly allowed in IAM policies. When multiple IAM policies are attached to an IAM entity, they explicitly deny access to any AWS resource overrides explicitly allowed. For example, if a user is granted permission to access an S3 bucket in one policy and the same user is explicitly denied permission to access that S3 bucket, that user cannot access the specified S3 bucket, as explicit denial overrides explicit allow permissions.
- Configuring or reconfiguring a given password policy does not impact existing user passwords. Password policies come into effect only when a new user is created, or when an existing user updates their password.
- It is important to remember that IAM policies are JSON documents, written as per the IAM policy standards. By default, access to AWS resources is denied, so usually, IAM policies are written to grant permission to any user, group, or service.
- It is very important to understand policy structure (that is, effect, action, and resource), and how to both write and interpret a policy.
- It is also important to understand IAM elements such as users, groups, and roles. As we know, users are end users, groups are collections of logically similar users, and roles are IAM entities that can be assumed by AWS resources and federated users. Roles can also be assigned to application or program resources such as EC2 or Lambda. A role attached to an AWS resource has its own policy. The permission associated with the policy defines which AWS resources can be accessed by the associated role.
- IAM roles don't have permanent credentials. As and when roles are assumed, a temporary access key and secret key are dynamically generated and automatically rotated.
- IAM roles can be of three types:
 - IAM roles for AWS resources (EC2 and Lambda)
 - IAM roles for cross-account access (granting permissions to IAM users across AWS accounts)
 - IAM roles for federated identity
- Before running the AWS CLI, it is important to install and configure a default access key, secret key, region, and output format. The default output format is JSON.

Summary

- The root user has complete, unrestricted access to all resources, including billing information, on the account. This is a superuser, and its permissions cannot be altered by any other user on the account.
- Never share your credentials with other users, especially root credentials, as they give unrestricted access within an AWS account.
- Usually, an individual user is authenticated by a username and password. Similarly, programmatic access through applications and CLIs are authenticated using an access key ID and secret key.
- An access key ID and secret key come in a pair. An access key ID is a 20-digit key and a secret key is a 40-digit key.
- AWS does not provide any mechanism to retrieve an access key ID or a secret key if these keys are lost.
- Password policies specify the complexity requirement of a password, and define a mandatory rotation period for a password associated with IAM users.
- MFA is an extra layer of security protection for user authentication that requires users to enter a six-digit token on top of the username and password.
- MFA can be enabled using a hardware-based MFA device or software/virtual MFA device.
- An IAM group is a collection of IAM users. A group lets you add, change, or remove permissions for multiple users altogether.
- There is a soft limit of a maximum of 300 IAM groups and 5,000 users. If more users are required, it is advisable to use the identity federation service to create temporary security credentials.
- An IAM role can be assumed by a user, application, or service to delegate access to an AWS resource within the same or another account.
- IAM roles can also be assumed by external user authentication software that uses an external IdP, compatible with SAML 2.0 or OIDC, or a custom identity broker.
- IAM roles can also be used to provide federated access to AWS services using Microsoft AD, LDAP, or similar identity providers.

- Identity federation is a mechanism through which applications can use external IdPs for authenticating users, rather than writing custom sign-in code for user authentication.
- A policy is a JSON-formatted document written in line with IAM policy notation.
- Trust policy defines the entities that can assume a role. Each role can have only one trust policy.
- A policy is a document that formally states one or more permissions. Policies are written to explicitly allow or deny permissions to access one or more AWS resources.
- ACLs are used to control the principal's capability to access an AWS resources.
- AWS-managed policies are built-in policies that are created and managed by AWS.
- Customer-managed policies are policies that are created and managed by customers in their AWS account.
- AWS console authentication can be integrated with AD FS.
- Web identity federation enables you to use many well-known identity providers, such as Amazon, Facebook, Google, LinkedIn, or any other OIDC-compatible IdP for authenticating users.
- STS is a web service that enables an application to dynamically generate temporary security credentials with restricted permissions based on an IAM role.
- By default, AWS STS is a global service with a single endpoint at <https://sts.amazonaws.com>. However, you can also choose to make AWS STS API calls to endpoints in any other supported region.
- An AWS account can be accessed using the 12-digit account ID, along with the user ID and password or an account alias and the respective credentials.

4

Virtual Private Clouds

Before we look at what **Virtual Private Clouds (VPCs)** are, let's understand what a computer network is. In very simple terms, when two or more computers are interconnected to share resources and communicate between each other, it is called a computer network.

When we talk about interconnected computers in an organization, there are some distinct requirements around the usage of these computers:

- Some computers are restricted to be accessed from within the organization only.
- Some computers need to be accessed from within the organization as well as from outside of it.

Based on the usage of the resources in a network, the network is subdivided into a number of segments. For example, the resources that are required to be accessed from within the organization are kept in a private segment of the network. Similarly, the resources that are required to be accessed from within the organization as well as from outside of it are kept in a public segment of the network.

The segment that logically isolates the resources in a network is called a **subnet**. In an organization, a network may be subdivided into multiple subnets based on the needs of the organization. In other words, there may be more than one public or private subnet in a network. Resources in a network need to communicate with one another, so they need to have a unique identity through which these resources can be distinctly identified and communication can be initiated between them. Such an identity for resources in a network is called an **IP address**.

So far, we have looked at networks in simple terms. Let's revisit our understanding about networks in a few simple points:

- When two or more computers are interconnected for communication between each other, it is called a **network**.
- Resources in a network need to communicate with each other based on requirements.
- Resources in a network are subdivided into logical segments called subnets.
- Resources in a private subnet are accessible within the network or organization.
- Resources in a public subnet are accessible from outside the organization as well.
- We have also seen that resources in a network are logically isolated in subnets.

Introduction to VPCs

As we now have a basic understanding of what a computer network is, let's look at what an AWS VPC is. A VPC is similar to a computer network that we can create in an on-premises data center. In the same way that we create dedicated and private networks within an organization, where computers in a network share network devices, such as routers and switches, we can create a VPC when we create a new account in AWS. A VPC has a similar network infrastructure as we can shape it in our own data center. The difference is that it is a virtual environment within a public cloud wherein the virtual network is logically isolated from other similar networks within the public cloud.

This chapter covers the following VPC components. Each of these components is described in subsequent pages of the chapter:

- **Elastic Network Interface (ENI)**
- Route table
- **Internet Gateway (IGW)**
- Egress-only IGW
- **Network Address Translation (NAT)**
- **Dynamic Host Configuration Protocol (DHCP)** option sets DNS

- VPC peering
- VPC endpoint
- ClassicLink

Unlike a traditional data center, a VPC can be created on demand without buying any hardware. You just need to create an AWS account and you are ready to get started.

Let's look at VPCs a little closer. As we have seen in [Chapter 2, Understanding the Fundamentals of Amazon Web Services](#), AWS has multiple regions. At the time of writing this chapter, there are 16 AWS regions, out of which 3 random regions are illustrated in the following diagram for visualizing the regions. We will see in the subsequent diagram how a VPC looks within a region:

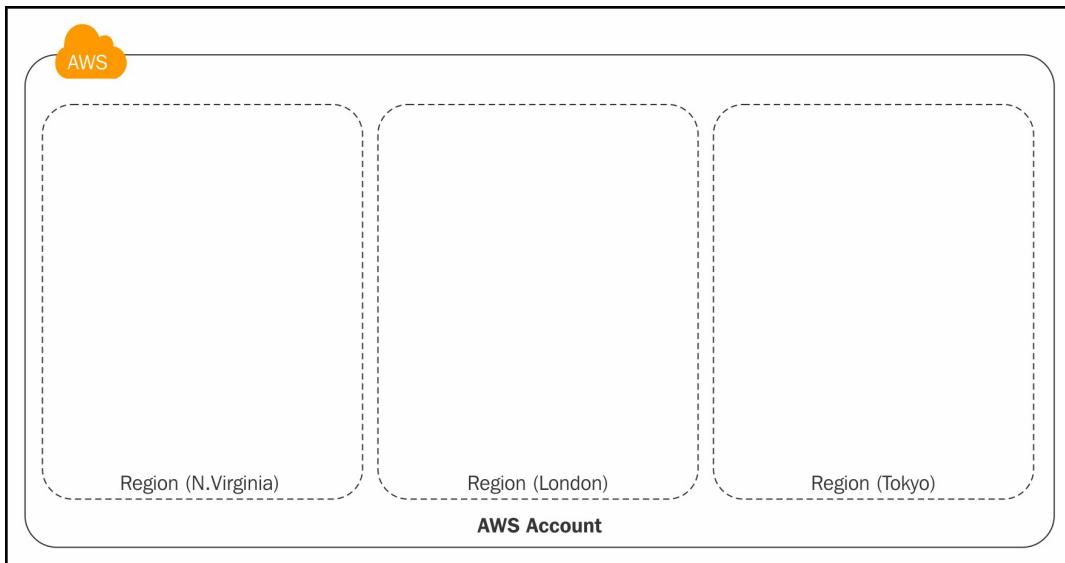


Figure 4.1: AWS account with regions

Every AWS region has two or more **Availability Zones (AZs)**. In [Figure 4.2](#), we have considered one region to visualize multiple AZs and how a VPC spans a region and multiple AZs.

Each AWS account can access multiple regions and each region has two or more AZs. A VPC can be created in any region and can span multiple AZs. A VPC's scope is limited to a single region; however, it can span multiple AZs. To explain this concept, *Figure 4.2* represents an AWS account, one region out of multiple AWS regions, a default VPC with **Classless Inter-Domain Routing (CIDR)** ($172.31.0.0/16$), and two AZs:

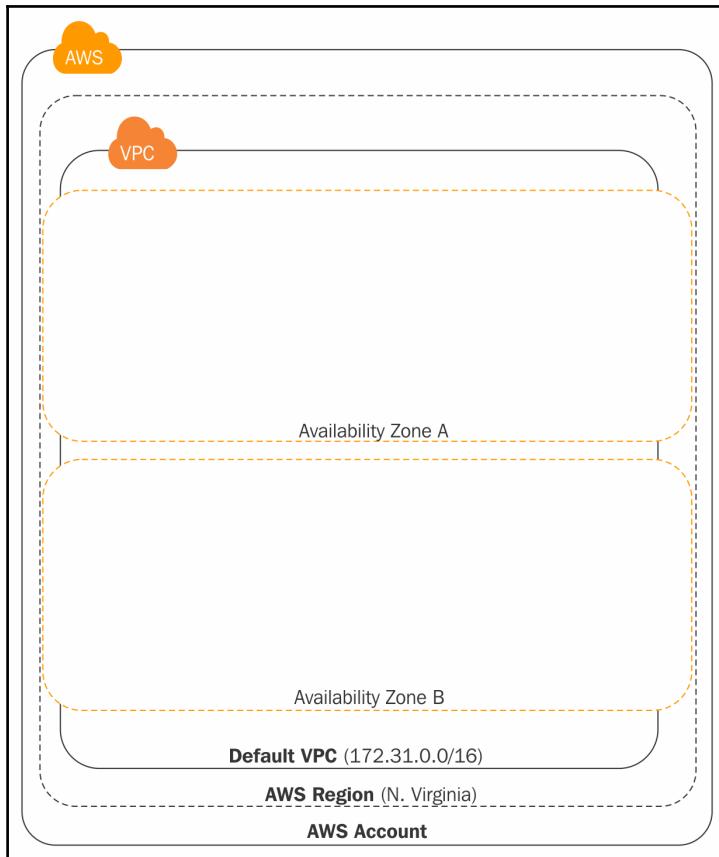


Figure 4.2: VPC visualization in a region with multiple AZs

Within one AWS account, several networks may exist (such as $10.0.0.0/16$ and $10.1.0.0/16$). For example, an organization can work on several different projects at the same time, and, for each of the projects, they can create individual network VPCs to isolate traffic and AWS resources. Each VPC spans the region in which it is created. An organization may also create multiple VPCs for each of the regions where their offices are located.

When you create a VPC, you need to specify an IP range for that VPC. This IP range is called **CIDR**, for example, `10.0.0.0/16`. CIDR is a set of IP standards that is used to create a unique identity for a network. It defines a set of IPs that can be allocated to resources within a network. You can look at the RFC 4632 standards at <https://tools.ietf.org/html/rfc4632> for more information on CIDR.

Now let's look at what a subnet inside a network is with respect to a VPC.

Subnets

Subnet is short for **subnetwork**. As we saw at the beginning of this chapter, a network is subdivided into multiple logical parts for controlling access to individual logical subparts of the network. When we create a subnet, we need to specify a unique CIDR block for the subnet. This CIDR block has to be a subset of the VPC CIDR block. Each subnet must reside entirely within a single AZ as a subnet cannot span multiple AZs.

Subnets are categorized as public and private subnets based on their security profile, or, in other words, based on their route table. We will now discuss different types of subnets.

Private subnets

A private subnet is a subset of a network wherein resources within a subnet are isolated and restricted for access from within the VPC. Any incoming traffic from the internet cannot directly access the resources within a private subnet. Similarly, outgoing traffic from a private subnet cannot directly access the internet. Outgoing traffic to the internet is either restricted or it is routed through NAT. We will learn more about NAT in subsequent sections of this chapter.

Resources in a private subnet are assigned a private IP that is accessible only from within the VPC. A route table defines the routing of the traffic to and from the subnet and ultimately determines whether a subnet is a private or public subnet based on whether it has a direct route to an IGW or not. This chapter discusses IGWs in subsequent sections.

Public subnets

A public subnet is a subset of a network wherein the resources within the subnet are isolated and can be accessed from within the VPC as well as from the internet. Any incoming traffic can directly access the resources located in a public subnet. Resources in a public subnet are assigned a public IP that is accessible from the internet.

Similarly, traffic originating from a public subnet can directly access the internet. Unlike a private subnet, traffic going out from a public subnet is not routed through NAT, but directly routed to an IGW. Such routing makes it possible for the resources to directly access the internet from a public subnet.

The following diagram describes how public and private subnets are segregated within a VPC:

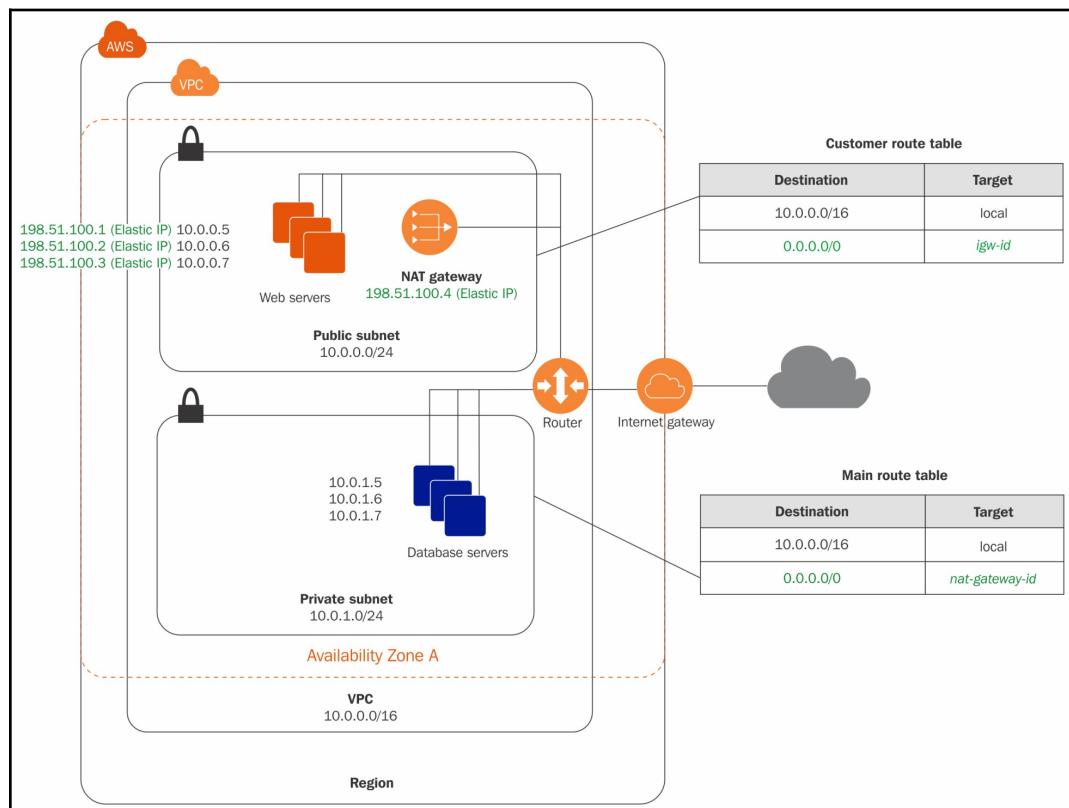


Figure 4.3: VPC with public and private subnets

As you can see in *Figure 4.3*, VPC CIDR is 10.0.0.0/16 and there are two subnets, one each for public and private resources. Both the subnets have their own CIDR. The main differentiator is the route table. As you can see in the route table, for destination 0.0.0.0/0, the target in the private subnet is an NAT gateway ID and the target for the public subnet is IGW ID. CIDR 0.0.0.0/0 denotes any traffic over the internet.

IP addressing

When an EC2 instance is launched, it carries an IP address and an IPv4 DNS hostname. The IP address and DNS hostname vary depending on whether the instance is launched in an EC2-Classic platform or a VPC. When an instance is launched in Amazon VPC, it supports both IPv4 and IPv6 addresses. An EC2-Classic platform supports only IPv4; it does not support IPv6.

By default, Amazon uses IPv4 addressing for the instance and VPC CIDR. This is the default behavior of EC2 instances and VPCs. Alternatively, a user can assign the IPv6 addressing protocol to VPCs and subnets, which would consequently allow you to assign IPv6 addresses to instances in a VPC.

We will be looking at the following IPs in the upcoming sections:

- Private IPs
- Public IPs
- Elastic IPs

Private IPs

As we have seen in earlier sections, a private IP address cannot be reached over the internet. Private IP addresses can be used for communication between instances within the same network. When an instance is launched, Amazon uses the DHCP to assign a private IP address to the instance. Apart from a private IP address, an instance is also assigned an internal DNS hostname that ultimately resolves to a private IPv4 address of the instance.

Here is an example of an internal DNS hostname: ip-10-5-200-21.ec2.internal. This DNS name can be used for communication between instances within the same network, but it cannot be resolved outside of the network or over the internet since it is a private DNS address.

When an instance is launched in a VPC, it is assigned a primary private IP address. This IP address is automatically selected from the IPv4 range of the subnet. Alternatively, you can also specify a custom IP address out of the IPv4 CIDR range of the subnet. If you specify a custom IP, that IP must not be already in use by any other instance. A primary IP address is assigned to a default Ethernet interface. This default Ethernet interface is named **eth0**. A primary IP address cannot be changed once an instance is launched. You can also assign a secondary IP address to an instance. Unlike a primary IP address, a secondary IP address can be changed and assigned to other instances.

EC2 instances launched in VPC retain their IP addresses even if the instances are stopped or restarted. The IP address is released only when the instance is terminated. Instances in EC2-Classic release their IP address as soon as they are stopped or terminated. If an instance in EC2-Classic is restarted, it is assigned a new IPv4 address.

Public IPs

Unlike a private IP address, a public IP address can be reached over the internet. Public IP addresses can be used for communication over the internet. When an instance is assigned a public IP address, it also receives an external DNS hostname.

Here is an example of an external DNS hostname: ip-XXX-XXX-XXX-XXX.compute-1.amazonaws.com, where XXX-XXX-XXX-XXX indicates the public IP address of the instance.

This public DNS name can be used for communication between instances within the same network or outside of the network over the internet. When this external hostname is used within the network, it is resolved to a private IP address of the instance. If this external DNS name is used outside the network, then it resolves to a public IP address. This public IP address is mapped to a primary private IP address using NAT.

In an EC2-Classic platform, when an instance is launched, it is automatically assigned a public IP from the public IPv4 address pool. Instances launched in an EC2-Classic platform must have a public IP. This is the default behavior of the EC2-Classic platform and it cannot be changed.

When an instance is launched in a VPC subnet, there is a subnet attribute that determines whether an instance launched in the subnet automatically gets a public IP or not. Public IPs for VPC instances are assigned from the EC2-VPC public IPv4 address pool. Amazon automatically assigns a public IP to an instance launched in a default VPC, while instances in a non-default VPC do not get a public IP automatically. This behavior can be controlled using the VPC subnet settings.

Here's how you can control public IPv4 addressing attributes for a VPC subnet:

1. Go to the VPC console through the dashboard or go to <https://console.aws.amazon.com/vpc/>.
2. From the VPC console, go to **Subnets**.
3. Select a subnet from the list of subnets in your VPC and choose **Actions**.
4. Select **Modify auto-assign IP settings**.
5. Select the **Enable auto-assign public IPv4 address** checkbox.
6. When this checkbox is selected, it assigns a public IPv4 address every time an instance is launched in the selected subnet. You can select or clear this checkbox based on the requirement and then save the settings.

You can override this behavior while launching an instance in a VPC subnet. Amazon provides an option to choose whether you want to auto-assign a public IP to the instance or not while launching an EC2 instance.

After an instance is launched, you cannot assign or release a public IP address manually to an instance. Amazon releases the public IP when an instance is stopped or terminated. When the same instance is stopped and restarted, it automatically gets a new public IP address. Again, you cannot control this behavior. If you want to use a persistent IP address or need to manually assign/release the IP address, you need to use an **Elastic IP address**.

When you assign an Elastic IP address to the primary Ethernet interface, Amazon automatically releases the public IP address of the instance. An Elastic IP address is equivalent to a public IP address that can be controlled by users.

Elastic IP addresses

An Elastic IP address is a public IPv4 address that can be allocated to an AWS account. An Elastic IP can be assigned or released from an instance as needed. Once an Elastic IP is allocated, it remains in the account until it is explicitly released from the account. As far as the use of an Elastic IP is concerned, it is similar to a public IP specified in the previous point. Here's some of the important aspects of an Elastic IP:

- To use an Elastic IP, you first need to allocate it in your account. Once an Elastic IP is allocated, you can assign it to an EC2 instance or a network interface.
- If an Elastic IP is associated with a primary network interface of an instance, Amazon automatically releases the public IP address associated with the instance or interface. An instance with a primary network interface cannot have an Elastic IP and a public IP at the same time.
- You can associate an Elastic IP to an EC2 instance and disassociate it as needed. Once the Elastic IP is disassociated from the instance, the same IP can be associated to any other instance.
- If you disassociate an Elastic IP from an instance, it remains in the account until you explicitly release it from the account.
- If an Elastic IP is not associated to an instance, Amazon charges the account on an hourly basis for that IP. Amazon also charges if an Elastic IP is associated to a stopped instance. In short, Amazon charges small fees for any unutilized Elastic IP addresses to ensure efficient usage of Elastic IPs. There is no charge for one Elastic IP address if it is associated with a running instance. If there is more than one Elastic IP address associated with an instance, you pay for additional Elastic IP addresses.

- An Elastic IP is associated with a specific region. You cannot use the same Elastic IP in different regions.
- When an Elastic IP is associated with a primary network interface of an instance, its public IPv4 address is released and its public DNS hostname is updated to reflect the Elastic IP address.
- When a public DNS hostname is accessed from outside the VPC or network, it resolves to a public IP or Elastic IP of the instance. If a public DNS hostname is accessed from within the VPC or network, it resolves to a private IP of the instance.

Creating a VPC

By default, when you create an AWS account, Amazon automatically provisions a default VPC for you. You can customize the default VPC based on your needs. You can add more subnets, remove any existing subnets, change the default route table, attach network gateways, or change the **Network Access Control List (NACL)** as per your requirements.

You can configure the default VPC and use it as needed, or you can create additional VPCs using either the VPC wizard or by creating custom VPCs manually. The VPC wizard provides four predefined categories of VPCs, which can help you quickly build the VPCs; they are these:

- VPC with a single public subnet
- VPC with public and private subnets
- VPC with public and private subnets and hardware VPN access
- VPC with a private subnet only and hardware VPN access

Let's understand each of these VPC types and the steps involved in creating the respective VPCs.

VPCs with a single public subnet

To create a VPC with a single public subnet using a wizard, follow these steps:

1. Go to <https://console.aws.amazon.com/vpc> or select **VPC** from the AWS dashboard. It brings up the VPC dashboard:

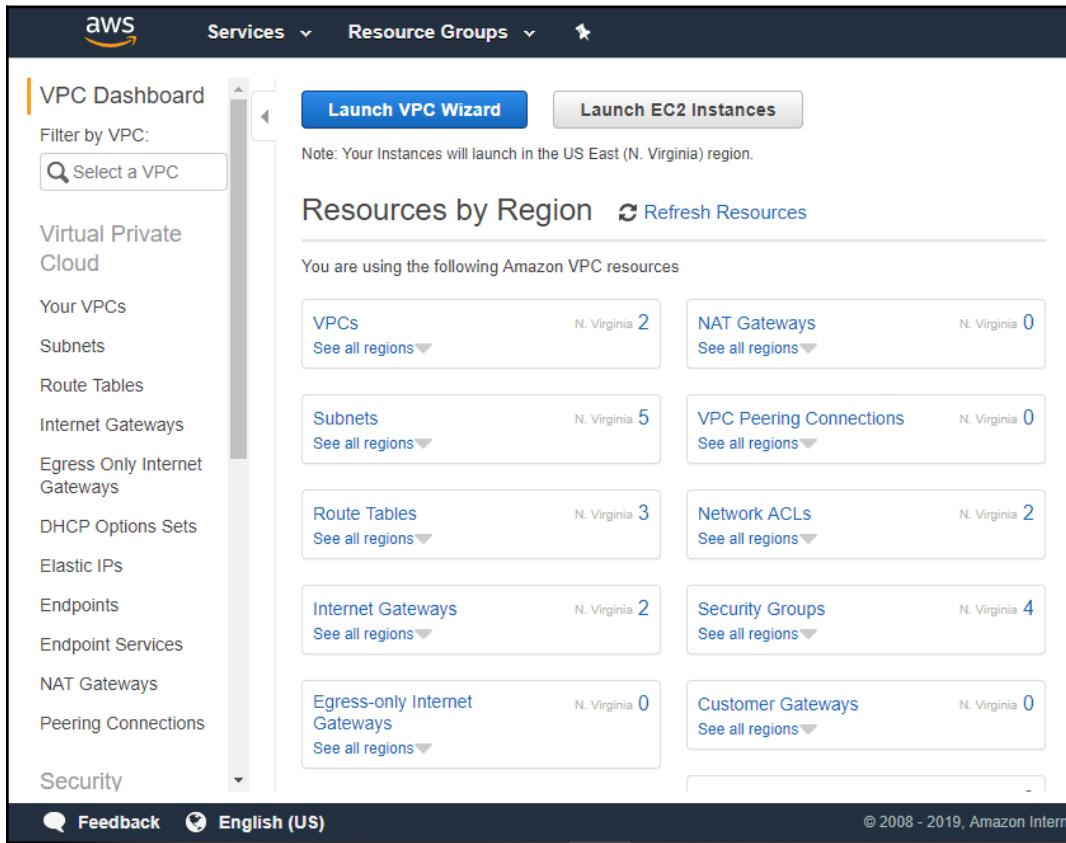


Figure 4.4: VPC dashboard

2. Ensure that you have selected the right region where you want to create the VPC. Regions can be selected from the top-right corner of the screen, just as **N.Virginia** is selected in the preceding screenshot.
3. Click on the **Launch VPC Wizard** button, and the following screen will appear:

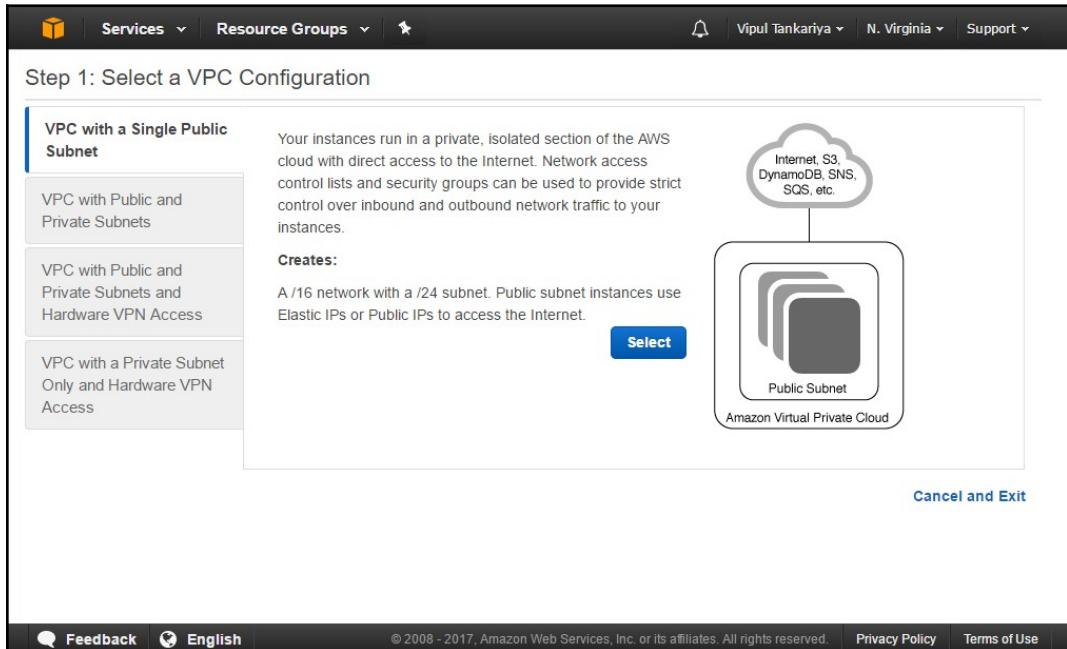


Figure 4.5: VPC wizard

4. Click on the **Select** button. It will show you the following:

The screenshot shows the 'Step 2: VPC with a Single Public Subnet' configuration page. The top navigation bar includes 'Services', 'Resource Groups', and 'Support'. The user is signed in as 'Vipul Tankariya' in the 'N. Virginia' region.

IPv4 CIDR block: 10.0.0.0/16 (65531 IP addresses available)

IPv6 CIDR block: No IPv6 CIDR Block
 Amazon provided IPv6 CIDR block

VPC name: [Input field]

Public subnet's IPv4 CIDR: 10.0.0.0/24 (251 IP addresses available)

Availability Zone: No Preference

Subnet name: Public subnet

You can add more subnets after AWS creates the VPC.

Service endpoints: Add Endpoint

Enable DNS hostnames: Yes No

Hardware tenancy: Default

Enable ClassicLink: Yes No

Buttons at the bottom: Cancel and Exit, Back, Create VPC

Footer links: Feedback, English, © 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved., Privacy Policy, Terms of Use

Figure 4.6: VPC wizard

5. Provide an **IPv4 CIDR block**. The default is 10.0.0.0/16, which provides 65,531 IPs in the VPC. This is the maximum size of VPC you can create in AWS by using the /16 CIDR range. The following table shows the IP range for CIDR 10.0.0.0/16. If you are not familiar with how to calculate the CIDR range, you can visit sites such as <http://ipaddressguide.com/cidr> and play around with the CIDR calculator. As you can see in the following table, the total hosts or IP address range in this CIDR is 65,536; however, only 65,531 IPs are usable. Amazon reserves five IPs in any subnet for various purposes:

CIDR range	10.0.0.0/16
Netmask	255.255.0.0

Wildcard bits	0.0.255.255
First IP	10.0.0.0
Last IP	10.0.255.255
Total IPs	65,536

Details of the reserved IPs are given in the following table. From this table, you can see that whenever you create any subnet, the respective five IPs in that subnet are reserved by AWS, that is, $x.x.x.0$, $x.x.x.1$, $x.x.x.2$, $x.x.x.3$, and $x.x.x.255$:

	Network address
10.0.0.0	Reserved for VPC router
10.0.0.1	Reserved for DNS server
10.0.0.2	Reserved for future use
10.0.0.3	Network broadcast address
10.0.0.255	

6. You can keep **No IPv6 CIDR Block** selected or select **Amazon provided IPv6 CIDR block** if your VPC needs IPv6 addresses as well. For clarity, keep **No IPv6 CIDR Block** as selected.
7. Enter the VPC name as required again in the **VPC name** field. The VPC name can have a maximum of 256 characters.
8. Specify the public subnet's CIDR range. The default is $10.0.0.0/24$. This CIDR range spans 256 IPs, out of which 5 IPs are reserved, as described in step 5, leaving reserved IPs aside; this gives you 251 usable IPs in the subnet.
9. You can choose to specify any specific AZ for the subnet or keep **No Preference** selected against the **Availability Zone** field.
10. Enter any subnet name against the **Subnet name** field. A subnet name can have a maximum of 256 characters. You can add more subnets after the VPC is created.
11. You can add **Service endpoints** by clicking on the **Add Endpoint** button. With service endpoints, you can directly access the respective service from within the VPC. Currently, this feature supports the S3 endpoint. Adding an S3 endpoint to VPC allows direct access from VPC to S3. If an S3 endpoint is not added, any request to S3 goes through IGW. That means your data or traffic goes through the internet for any interaction with S3 service. Adding an S3 endpoint routes the traffic directly to S3 and keeps the communication within the VPC network. For now, do not add any endpoints to ensure clarity.

12. Keep **Enable DNS hostnames** selected as **Yes**. When the DNS hostnames attribute is enabled in a VPC, any EC2 instance provisioned in the VPC is automatically assigned a DNS hostname.
13. In the next step, you need to select **Hardware tenancy**. You can select either **Default** tenancy or **Dedicated** tenancy. **Default** tenancy specifies that a single physical AWS machine may run instances provisioned by multiple AWS customers. If you select **Dedicated** tenancy, it ensures that EC2 instances launched in this VPC are launched on a dedicated hardware. Tenancy of a VPC cannot be changed after a VPC is created. **Dedicated** tenancy costs more than **Default** tenancy. Select this option according to your requirements.
14. The **Enable ClassicLink** option allows a VPC to communicate with the EC2 instances launched in EC2-Classic. Without this option enabled, resources in the VPC need to use a public IP address of the EC2-Classic instance or tunneling for communication. If you have any resources in EC2-Classic, you can choose **Yes** to enable the ClassicLink, otherwise, you can keep **No** selected.
15. Click on the **Create VPC** button to create the VPC. This final step pops up a small progress bar while the system creates your VPC. Once the progress completes 100%, it shows the newly created VPC in the **Your VPCs** list from the VPC dashboard:

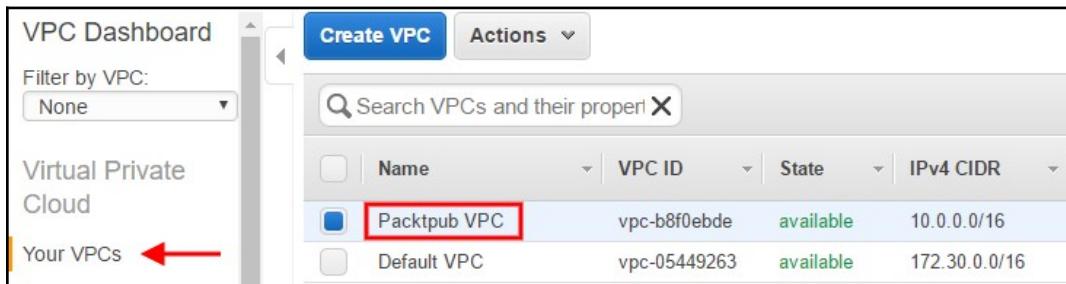


Figure 4.7: Your VPCs in the VPC dashboard

The previously mentioned steps for creating a VPC generate a VPC with a single public subnet. The following diagram shows the architecture of the VPC:

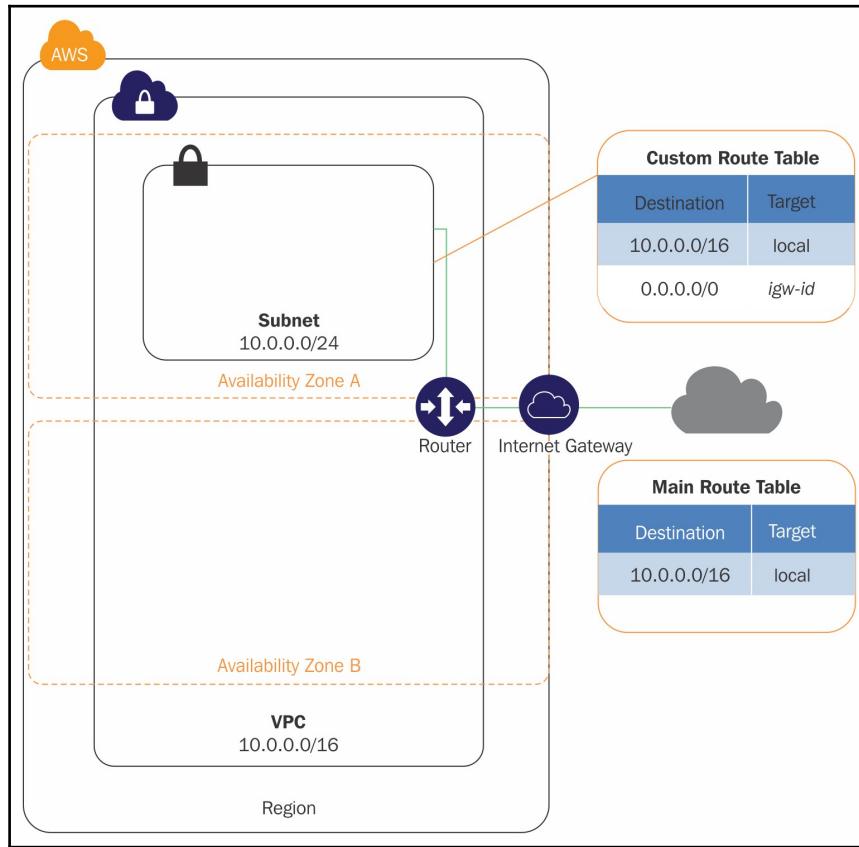


Figure 4.8: Architecture diagram – VPC with single public subnet

The wizard creates a VPC with the following list of features:

- It creates a VPC with a 10.0.0.0/16 CIDR block and 65,536 IPv4 addresses. It associates an IGW to the VPC.
- It also creates a public subnet with the 10.0.0.0/24 CIDR range. It encompasses 256 IPv4 addresses. As was elaborated on earlier, Amazon reserves 5 IPs out of the subnet, which leaves 251 usable IPv4 addresses.
- It creates a custom route table and attaches it to the subnet. This route table enables traffic between the subnet and the IGW.

VPCs with private and public subnets

The steps for creating a VPC with private and public subnets are almost identical to those for creating a VPC with a single public subnet, except the wizard offers to create private subnets and it also offers to attach a NAT gateway for routing traffic from private subnets to the internet.

To create a VPC with single, private, and public subnets using the wizard, follow these steps:

1. Go to <https://console.aws.amazon.com/vpc> or select **VPC** from the AWS dashboard.
2. Ensure that you have selected the right region where you want to create the VPC and click on **Start VPC Wizard**.
3. From the subsequent screen, select **VPC with Public and Private Subnets** and click on the **Select** button:

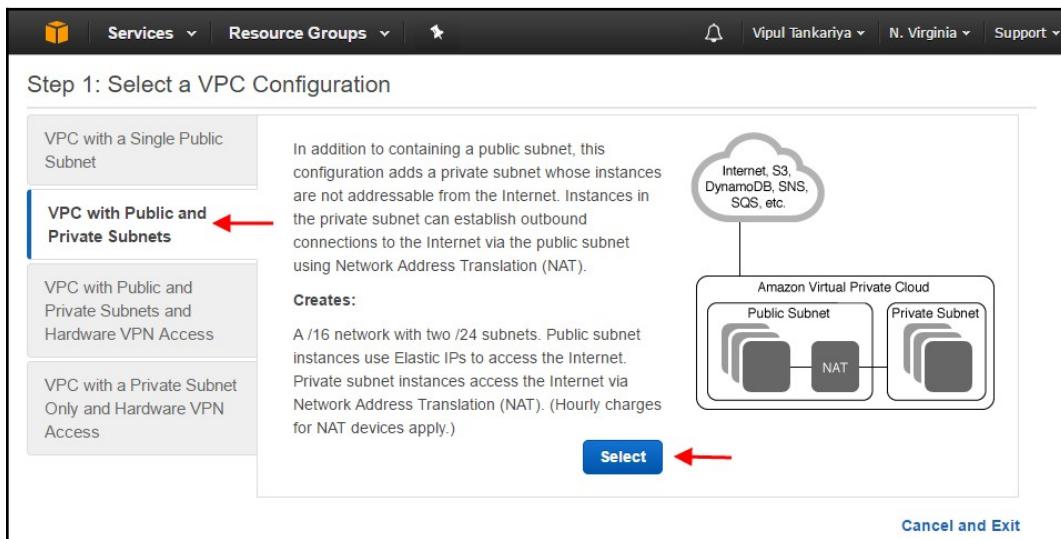


Figure 4.9: VPC wizard for VPC with public and private subnets

4. On the subsequent screen, you can see that the wizard is almost identical to the wizard for creating a VPC with a single public subnet. This wizard offers to create private subnets and it also offers to attach a NAT gateway for routing traffic from private subnets to the internet:

Step 2: VPC with Public and Private Subnets

IPv4 CIDR block*: 10.0.0.0/16 (65531 IP addresses available)

IPv6 CIDR block: No IPv6 CIDR Block Amazon provided IPv6 CIDR block

VPC name: []

Public subnet's IPv4 CIDR*: 10.0.0.0/24 (251 IP addresses available)

Availability Zone*: No Preference ▾

Public subnet name: Public subnet

Private subnet's IPv4 CIDR*: 10.0.1.0/24 (251 IP addresses available) ←

Availability Zone*: No Preference ▾

Private subnet name: Private subnet

You can add more subnets after AWS creates the VPC.

Specify the details of your NAT gateway (NAT gateway rates apply). ←

Elastic IP Allocation ID*: []

Use a NAT instance instead ↑

Service endpoints

Add Endpoint

Enable DNS hostnames*: Yes No

Hardware tenancy*: Default ▾

Enable ClassicLink*: Yes No

[Cancel and Exit](#) [Back](#) [Create VPC](#)

Figure 4.10: VPC wizard – NAT and private subnet detail

5. Specify the details in all the fields as required, including **Elastic IP Allocation ID** in the **Specify the details of your NAT gateway** section. You can allocate a new Elastic IP address and obtain its allocation ID from the listing of **EC2 dashboard | Elastic IPs**.
6. Click on the **Create VPC** button after providing all the required details on the screen. It creates a VPC with private and public subnets.

The following diagram shows the architecture of the newly created VPC:

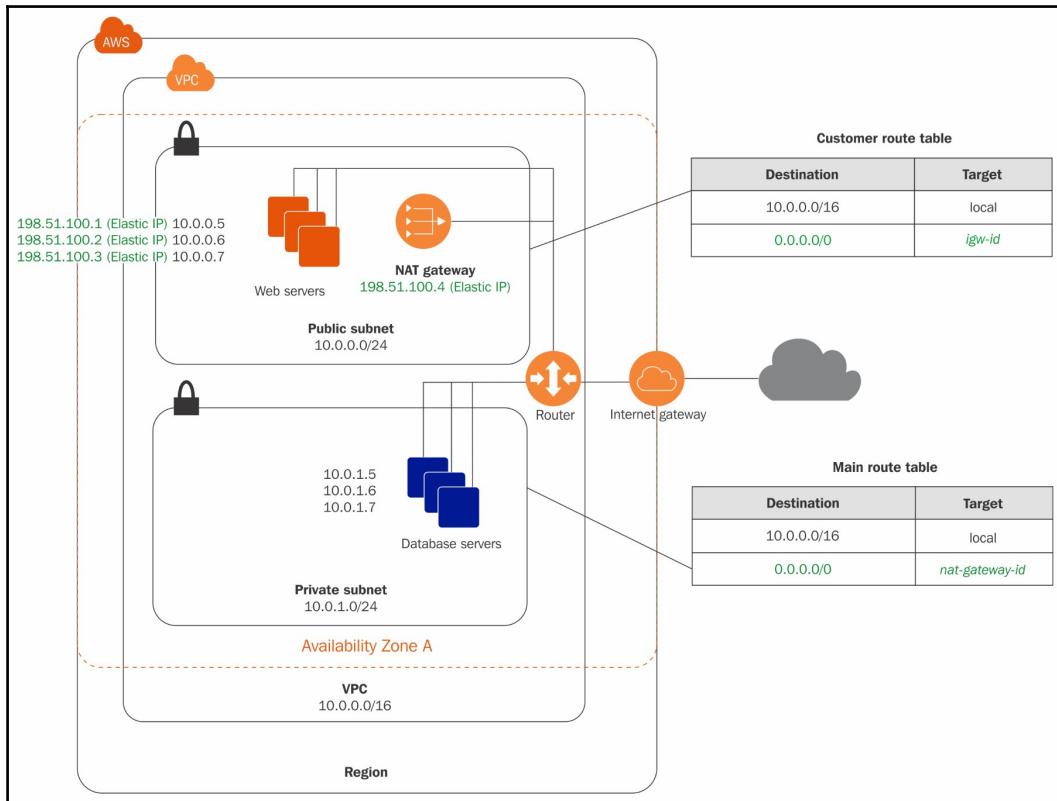


Figure 4.11: Architecture diagram – VPC with private and public subnets

Here are the details of the VPC created using the wizard:

- It creates a VPC with a /16 CIDR range and 65,536 IPv4 addresses.
- It also creates a public subnet with a /24 IPv4 CIDR range and 256 IPv4 addresses. Since this is a public subnet, it also creates a route table entry with a route to an IGW. It creates a private subnet with a /24 IPv4 CIDR range and 256 private IPv4 addresses.
- It also adds an IGW and connects the VPC to the internet as well as other AWS services.
- It creates an NAT gateway with an Elastic IPv4 address supplied during the wizard. This allows instances in the private subnet to communicate over the internet through the NAT gateway.

- It adds a custom route table and associates it with the public subnet. This route table enables the instances in the public subnet to communicate with each other and over the internet through an IGW.

VPCs with public and private subnets and hardware VPN access

The steps for creating a VPC with private and public subnets and hardware VPN access are almost identical to those of the previous wizards, except for an additional screen that requires details for creating hardware VPN access. For ease of use and better understanding, only the last part of the wizard is elaborated on here.

The steps, in brief, are as follows:

1. Go to <https://console.aws.amazon.com/vpc> or select **VPC** from the AWS dashboard.
2. Ensure that you have selected the correct region where you want to create the VPC and click on **Start VPC Wizard**.
3. From the subsequent screen, select **VPC with Public and Private Subnets and Hardware VPN Access** and click on the **Select** button.
4. Fill in all the fields in the subsequent screen, which are identical to what we have gone through in the previous wizard.
5. After filling in all the fields, click on the **Next** button. It will bring up the following screenshot:

The screenshot shows the 'Step 3: Configure your VPN' screen of the AWS VPC Wizard. At the top, there's a navigation bar with 'Services', 'Resource Groups', a user icon, 'Vipul Tankariya', 'N. Virginia', and 'Support'. Below the title, there are two sections: 'Specify the public IP Address of your VPN router (Customer Gateway)' and 'Specify the routing for the VPN Connection'. In the first section, there are three input fields: 'Customer Gateway IP:' with a placeholder '192.168.1.1', 'Customer Gateway name:' with a placeholder 'CustomerGateway', and 'VPN Connection name:' with a placeholder 'VPNConection'. A note below says 'Note: VPN Connection rates apply.' In the second section, there's a dropdown menu for 'Routing Type:' with 'Dynamic (requires BGP)' selected. At the bottom right are three buttons: 'Cancel and Exit', 'Back', and a large blue 'Create VPC' button.

Figure 4.12: Create VPC wizard – configuring your VPN

6. Specify the **Customer Gateway IP**. This is an IP address of the anchor from the on-premises or target location where we need to establish a VPN tunnel from our AWS VPC. The anchor can be a hardware or software appliance or router.
7. Provide the **Customer Gateway name**. This is optional but recommended for easily identifying the customer gateway.
8. Provide the **VPN Connection name**. This is also optional but it is recommended to give a name to this connection.
9. Select the **Routing Type**. There are two route types, dynamic and static. You need to enter the IP prefix for your network manually while creating the VPN connection in static routing. In dynamic routing, we can use **Border Gateway Protocol (BGP)** to advertise the IP prefix automatically to the VPN gateway. Ensure that your target appliance supports BGP if you select **Dynamic (requires BGP)**.
10. Click on the **Create VPC** button after providing all the required details on the screen. This creates a VPC with private and public subnets and hardware VPN access.



To establish a VPN connection between a VPC and a target network, an administrator on the target network needs to configure the anchor appliance at the target. This is generally not required for a developer to understand. Also, it is out of the scope of the AWS Certified Developer – Associate exam.

If you're interested in knowing more about how a network administrator needs to configure the target anchor appliance for VPN connectivity, you can refer to <http://docs.aws.amazon.com/AmazonVPC/latest/NetworkAdminGuide/Welcome.html>.

The following diagram shows the architecture of the newly created VPC:

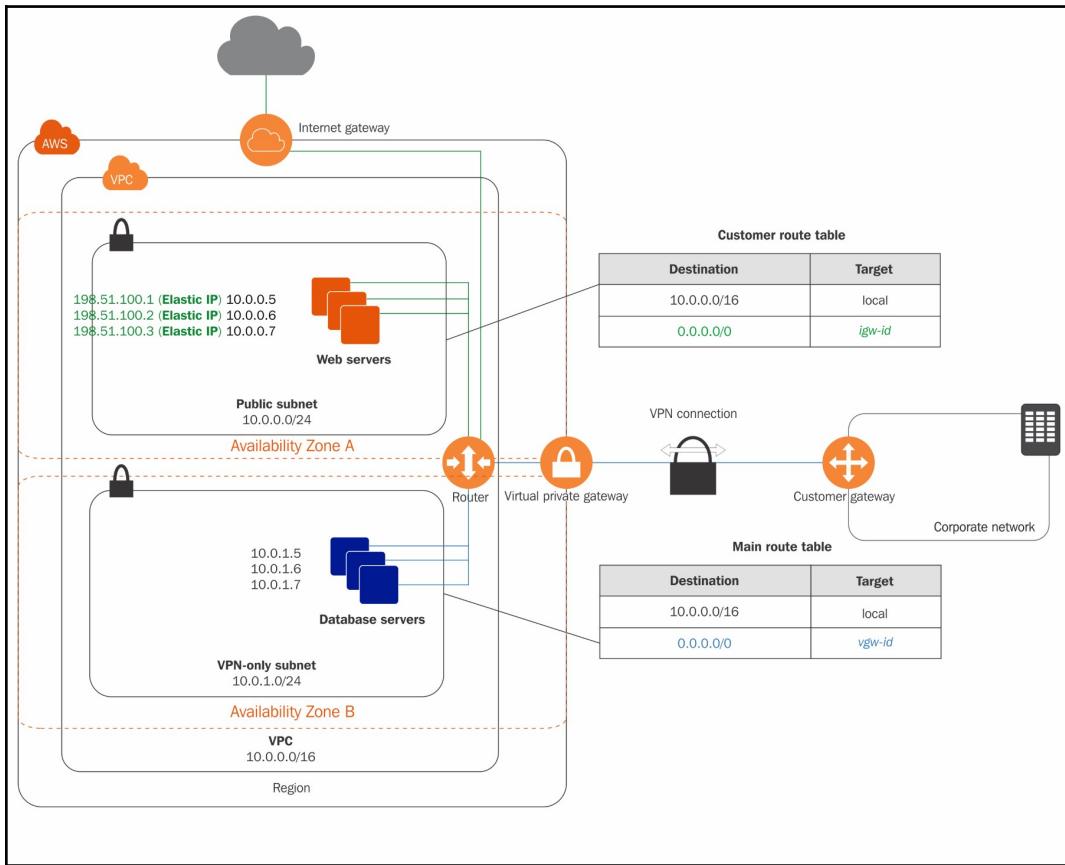


Figure 4.13: VPC with private and public subnets and VPN access

The following points describe the VPC created using the wizard:

- It creates a VPC with a size /16 IPv4 CIDR range with 65,536 private IPv4 addresses.
- It creates a public subnet /24 IPv4 CIDR block and 256 private IPv4 addresses.
- It creates a VPN-only subnet with IPv4 CIDR block of size /24 and 256 private IPv4 addresses.
- It also creates an IGW that connects the VPC to the internet and other AWS services.

- It also creates a VPN connection between the VPC and target network. The VPN connection needs a **Virtual Private Gateway (VGW)** that points to the VPC endpoint on AWS, and a **Customer Gateway (CGW)** that points to the target network endpoint.
- It creates an NAT gateway with an Elastic IPv4 address supplied during the wizard. This allows instances in a private subnet to communicate over the internet through the NAT gateway.
- It adds a custom route table and associates it with the public subnet. This route table enables the instances in the public subnet to communicate with each other and over the internet through an IGW.
- A route is created for routing the traffic to a VPN connection that is ultimately routed to a CGW. This entry enables the resources on the target network to communicate with the instances in the VPC.

VPCs with a private subnet only and hardware VPN access

The steps for creating a VPC with a private subnet only and hardware VPN access are almost identical to those of the previous wizards except for one change: no public subnets are created in this wizard. This wizard creates a VPN connection between the VPC and target network over an IPSec VPN tunnel. There is no IGW for any communication with the internet. This scenario is recommended for extending a target or an on-premises network to AWS VPC for a secure communication without exposing it to the internet.

The steps (in brief) for creating a VPC with a private subnet only and hardware VPN access are as follows:

1. Go to <https://console.aws.amazon.com/vpc> or select **VPC** from the AWS dashboard.
2. Ensure that you have selected the correct region where you want to create the VPC and click on **Start VPC Wizard**.
3. From the subsequent screen, select **VPC with Private Subnets Only and Hardware VPN Access** and click on the **Select** button.
4. Fill in all the fields in the subsequent screen, which are identical to what we have gone through in the previous wizard.

5. After filling in all the fields, click on the **Next** button. This will bring up the screen that we saw in the previous wizard.
6. Specify the **Customer Gateway IP**. This is an IP address of the anchor from an on-premises or target location where we need to establish a VPN tunnel from our AWS VPC. The anchor can be a hardware or software appliance or router.
7. Provide the **Customer Gateway name**. This is optional but recommended for easily identifying the CGW.
8. Provide the **VPN Connection name**. This is also optional, but it is recommended to give a name to this connection.
9. Select the **Routing Type**. There are two route types, dynamic and static. If static routing is selected, you need to enter the IP prefix manually for your network while the VPN connection is being created. If dynamic routing is selected, the IP prefix is advertised automatically to the VPN gateway using BGP. Ensure that your target appliance supports BGP if you select **Dynamic (requires BGP)**.
10. Click on the **Create VPC** button after providing all the required details on the screen. It creates a VPC with a private subnet only and hardware VPN access.



To establish a VPN connection between a VPC and the target network, an administrator on the target network needs to configure the anchor appliance at the target. This is generally not required for a developer to understand. Also, it is out of the scope of the AWS Certified Developer – Associate exam.

If you're interested in knowing more about how a network administrator needs to configure the target anchor appliance for VPN connectivity, you can refer to <http://docs.aws.amazon.com/AmazonVPC/latest/NetworkAdminGuide/Welcome.html>.

The following diagram shows a VPC created for a private subnet only with hardware VPN access:

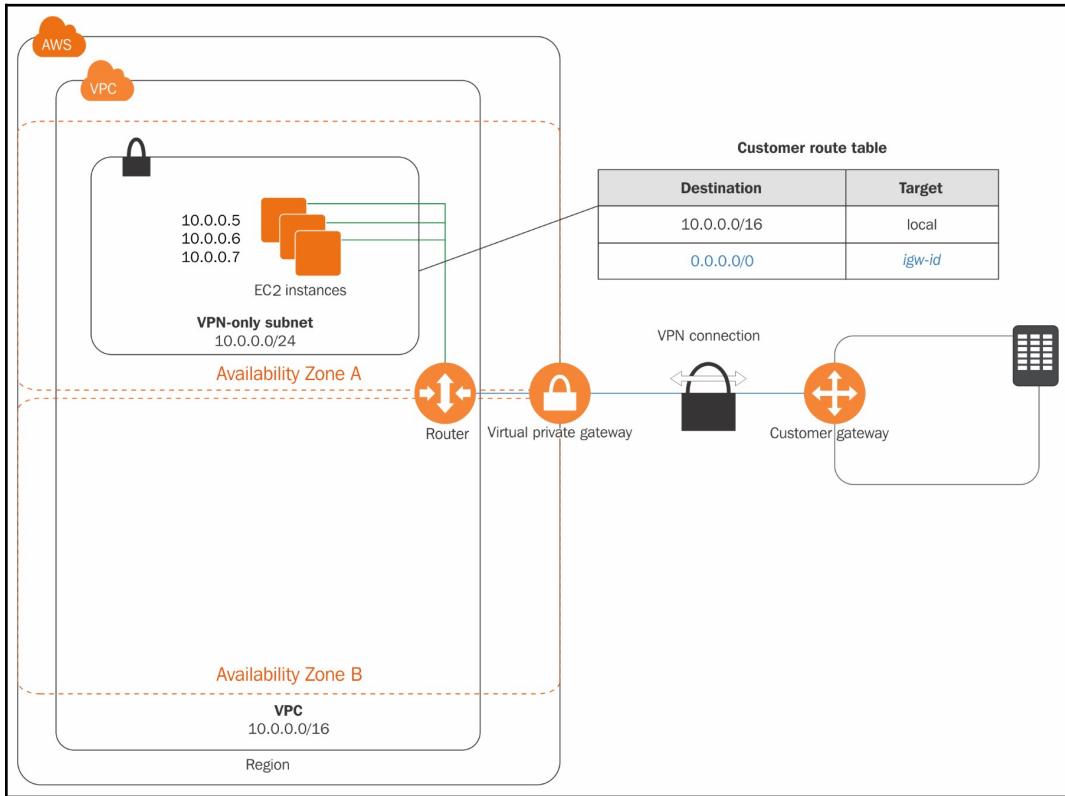


Figure 4.14: Architecture – VPC with private subnet only and hardware VPN access

The architecture created using this wizard includes the following:

- It creates a VPC with a /16 CIDR range and 65,536 private IP addresses.
- It creates a VPN-only subnet with a /24 CIDR range and 256 private IP addresses.
- It creates a VPN connection between your VPC and target network. The VPN connection includes a VGW pointing to the VPC and a CGW that points to the anchor appliance on the target network.
- It creates a custom route table associated to the subnet. The route table contains route entries for communication between instances within the VPC and also an entry for communication with other resources on the target network.

Now that you have understood how to create a VPC, let's see how we can make it more secure.

Security

While creating a VPC, security is one of the most critical aspects of the VPN of an organization. As AWS states in many of its official communications, the security of the customer network is one of its highest priorities. Keeping security at the top of the AWS charter, Amazon provides two features for taking care of network security and one feature for monitoring the network.

Security groups and **NACLs** are for network security and **flow logs** are for network monitoring. Security groups act as an EC2 instance-level firewall, while NACLs act as a subnet-level firewall. Flow logs provide insight into network traffic. In the following sections in this chapter, these features will be described in detail.

To start with, let's take a high-level overview of the difference between security groups and NACLs. The layers of communication, as shown in the following diagram, help us to understand the significance of security groups and NACLs in a VPC:

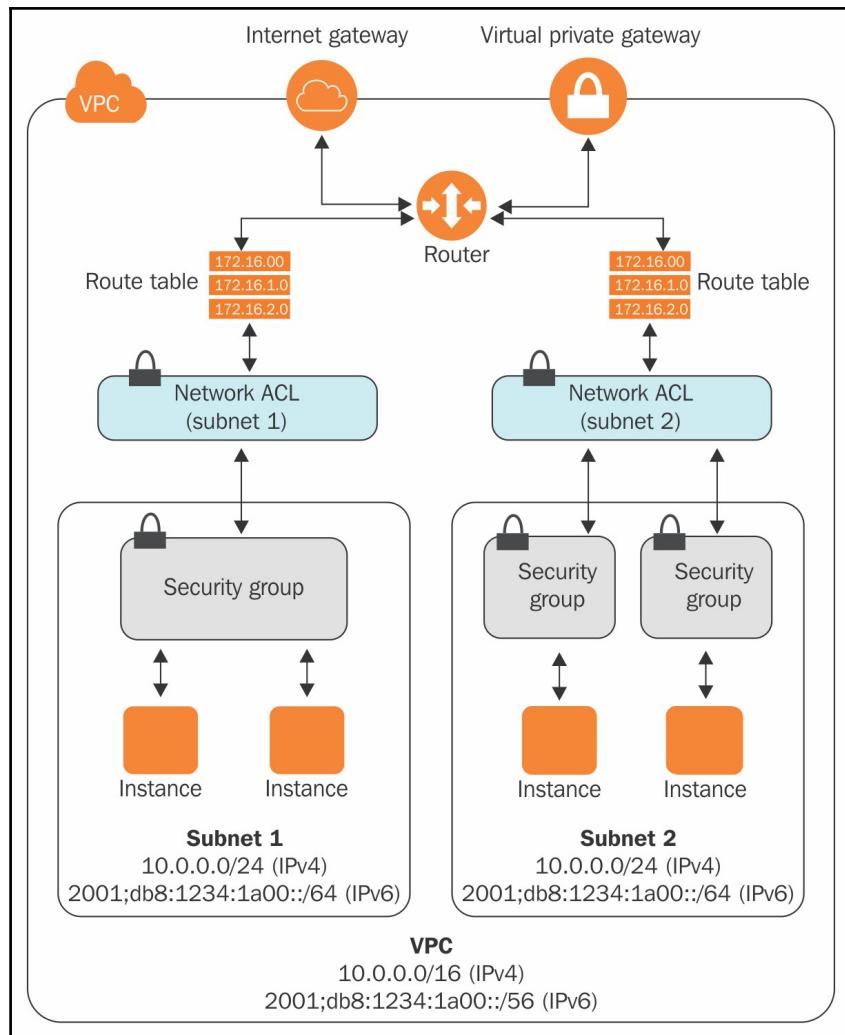


Figure 4.15: Security groups and NACLs

Security groups

A security group can be described as a virtual firewall that controls any traffic coming in or going out of the instances associated with the security group. When an EC2 instance is launched, that instance is associated with one or more security groups. Each of the security groups contains rules to allow traffic to or from its associated instances.

A security group can have multiple rules for inbound and outbound traffic. Inbound and outbound traffic is also called ingress and egress traffic, respectively. Security groups can be attached to an EC2 instance to restrict unsolicited traffic. Each security group can be attached to multiple EC2 instances. It is best practice to create an application-specific security group and attach it to one or more EC2 instances hosting the same application, using AWS to modify the security group and fine-tune it as per the changing business need from time to time. As and when required, you can allow or block communication on certain protocols, ports, and source IPs, without disturbing communication for other applications. In general, security group rules are implemented as per the hosted application's communication requirement for protocol, ports, and source IPs, on relevant EC2 instances.

To meet an organization's advanced security needs, third-party EC2-based firewall applications can be deployed in an AWS account on top of the security groups attached to EC2 instances. While launching an EC2 instance in a VPC, if a security group is not attached explicitly, then a default security group of the VPC is automatically attached to the EC2 instance. Security groups attached to an EC2 instance can be changed at any time based on need.

With the help of a CIDR notation, a source IP can be fixed to the particular IP, such as 10.108.20.107/32. Also, any source IP can be allowed by a 0.0.0.0/0 CIDR notation. It is best practice to allow ingress communication from a specified source IP rather than allowing it from everywhere, until or unless project-specific requirement is there. For example, communication on HTTP (port 80) or HTTPS (443) should be allowed from anywhere, but for SSH (22) and RDP (3389), it should be allowed only from trusted source IPs.

Also, a security group ID can be specified as a source IP to allow communication from all the instances that are attached to that security group. For example, in the case of autoscaling, the number of EC2 instances (that is, a hosted web application) and their IP addresses keeps changing. In such situations, it is best practice to attach a security group to such EC2 instances with the help of an autoscaling template and place a security group ID as a source IP in another security group. This other security group may be attached to the EC2 instance hosted with a database.



A security group attached to an EC2 instance should have at least one rule to allow SSH (22) or RDP (3389) to log in and perform maintenance activities from the trusted IP source.

Some important points about security groups are as follows:

- Each security group can have a maximum of 50 separate rules for inbound and outbound.
- One security group can be attached to more than one EC2 instance.
- At any given time, a maximum of five security groups can be attached to a network interface.
- When multiple security groups are attached to a single network interface, rules from each security group are aggregated as one rule and evaluated for any inbound or outbound traffic request.
- In situations where multiple rules for the same protocol and ports are given for an EC2 instance, the most permissive rules are applied. For example, say one rule allows communication on TCP port 22 (that is, SSH) from 10.108.123.65, and another rule allows communication on TCP port 22 from everywhere, 0.0.0.0. In that situation, communication from everywhere is allowed.
- As soon as any modification in security group rules is saved, it immediately gets applied to all associated EC2 instances.
- By default, when a security group is created, it allows all outgoing communication and blocks all incoming communication.
- In a security group, each rule is defined only to allow communication. There is no provision in a security group to create a rule for explicitly denying any traffic. The security group allows incoming traffic from all the ports and protocols defined in the rules. All other traffic is implicitly denied. This way, it eliminates the need for any explicit deny rule.
- Security groups are stateful. That means that any traffic allowed on a specific inbound port and protocol is automatically allowed for outbound traffic on the same port and protocol. You don't need to create outbound traffic rules for ports and protocols described in inbound rules.
- Instances associated with the same security group cannot communicate with each other unless there is a rule placed in any security group associated with these instances that allows communication between these instances.

NACLs

An NACL acts as a virtual firewall at the subnet level. It is an optional layer of security. Every VPC has a default NACL. Creating a VPC automatically creates a default NACL. Every subnet, whether it is private or public in a VPC, must be associated to one NACL. By default, NACLs block all inbound and outbound IPv4 traffic. At any given time, NACL rules can be modified and be put into immediate effect as they are saved. In the same VPC, different subnets can be associated with different NACLs.

Some important points about NACLs are as follows:

- Each subnet in a VPC must be associated with at least one NACL. At the time of creating a subnet, if it is not explicitly associated with any NACL, then it automatically gets associated with the default NACL.
- One NACL can be associated with one or more subnets; however, at any given time, only one NACL can be associated to a subnet.
- NACL rules are evaluated based on its rule numbers. It evaluates the rule starting from the lowest number to the highest number. The highest number can be 32,766. It is best practice to create rules with the sequence numbers having increments of 10, 50, or 100. It gives freedom to insert rules in between in future, if required.
- Separate rules to allow or deny can be created for inbound and outbound traffic.
- Unlike security groups, NACL rules are stateless. That means if a port is open for allowing inbound traffic, it does not automatically allow outbound traffic. Similarly, if a port is allowed for outbound traffic, it does not automatically allow inbound traffic. Rules for inbound and outbound traffic have to be defined separately.
- The default NACL for any VPC contains a rule numbered as * in inbound and outbound. This rule appears and executes last. While evaluating a traffic request, if any of the rules do not match whether to *allow* or *block* the network traffic, it blocks that traffic. This is the default behavior of NACL and it cannot be changed.
- Adding more rules to any NACL may bring network performance implications. Add NACL rules carefully.

Security groups versus NACLs

We will now compare security groups with NACLs:

Security group	NACL
It is stateful, which means that opening any inbound traffic on a port/protocol automatically allows outbound traffic on that port and protocol.	It is stateless, which means that you need to explicitly define inbound and outbound traffic rules. It does not allow inbound traffic for a rule defined for outbound traffic and vice versa.
Acts as a firewall at EC2 level. You need to associate a security group with one or more EC2 instances.	Acts as a firewall at the subnet level. NACL rules get applied to all EC2 instances within the related subnet.
By default, all incoming requests are denied and all outgoing requests are allowed.	By default, all incoming and outgoing requests are denied.
Rules do not execute in a sequence.	Rules execute in a sequence, as they are numbered.
Up to five security groups can be attached to a network interface.	Only one NACL can be attached to a subnet.

Flow logs

Flow logs are a feature that enable you to track incoming and outgoing traffic from a VPC's network interfaces. Flow logs can be enabled on a network interface, subnet, or VPC. Flow logs make it possible to audit network traffic. They contain information about source and destination IP addresses, ports, the **Internet Assigned Numbers Authority (IANA)** protocol number, and many more details for allowed and denied traffic based on the security groups and NACL. It stores data in CloudWatch Logs. It is also possible to create alarms that notify you of the occurrence of certain network traffic. Once a flow log is enabled on the AWS resources, it may take a little while to start collecting data and appear on CloudWatch Logs. It does not capture real-time log streams for the network interfaces. Flow logs help in fault diagnostics and traffic monitoring.

Flow logs for each network interface get stored in separate log streams. It is possible to create multiple flow logs publishing data to the same log group in CloudWatch Logs. For the same network interface, there could be a separate flow log just to capture rejected or accepted traffic. In such situations, flow logs are automatically combined as one log stream for the same network interface appearing multiple times in a log group.

Flow logs can be enabled for the network interfaces, which are either created for custom AWS resources such as EC2 instances, or AWS services such as Elastic Load Balancing, ElastiCache, RDS, and so on. Once a flow log is enabled at subnet or VPC level, adding a new instance in that subnet or VPC automatically starts collecting the flow logs for newly created network interfaces on that EC2 instance.

Once a diagnostic or network audit has been done, flow logs can be deleted at any time. Deleting flow logs does not delete existing flow log records or log streams. It just disables the flow log service and stops collecting new logs for respective resources such as network interfaces, subnets, or VPCs. Existing log streams can be deleted from the CloudWatch Logs console.

The limitations of a flow log are as follows:

- It cannot be enabled for a VPC that is peered with a VPC in another AWS account. It can only be enabled when both the peered VPCs are in the same AWS account.
- It is not possible to tag a flow log.
- Once a flow log is created, it is not possible to modify its configuration. You need to delete the existing flow log and create a new one.
- If any network interface has multiple IPv4 addresses, then in a flow log, traffic sent to a secondary IP displays the primary private IPv4 address in the destination IP address field.
- It captures traffic for custom-created DNS servers in the VPC, but does not capture traffic for an Amazon DNS server and DHCP traffic.
- It does not capture traffic generated by Microsoft Windows instances for Amazon Windows license activation.
- It does not capture traffic for instance metadata to and from 169.254.169.254. This is a dynamically generated link-local address that is used by the instance for collecting its instance metadata.
- It does not capture traffic to the reserved IP address for the default VPC router.



Creating a flow log must be associated with an IAM role having sufficient permissions to publish flow logs to the specified log group in CloudWatch Logs. Also, an IAM role must have a trust relationship, which should allow the flow log service to assume the role.

Controlling access

While designing the architecture of a VPC, it is recommended you at least use a privilege policy. Access to various resources and services in the VPC should be granted to individuals based on their role. There may be a number of roles in an organization, such as network administrator, developer, database administrator, and tester. All these individuals with different roles should have access to specific resources and services as their organizational role demands. AWS IAM makes it possible to define different access policies for various users rather than giving full control to all of them. By defining access policies, we can restrict the users to using specific APIs and take limited actions against various AWS services.

The following is an example of an access policy for a network administrator. This policy grants an administrator access to create and manage the VPC:

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {"Effect": "Allow",  
     "Action": ["ec2:*Vpc*", "ec2:*Subnet*", "ec2:*Gateway*", "ec2:*Vpn*",  
               "ec2:*Route*", "ec2:*Address*", "ec2:*SecurityGroup*",  
               "ec2:*NetworkAcl*", "ec2:*DhcpOptions*", "ec2:RunInstances",  
               "ec2:StopInstances", "ec2:StartInstances", "ec2:TerminateInstances",  
               "ec2:Describe*"],  
     "Resource": "*"  
    }  
  ]  
}
```

The following is an example of a read-only policy. A user with this policy can list all VPCs and various resources associated with the VPC:

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {"Effect": "Allow", "Action": ["ec2:DescribeVpcs",  
                                  "ec2:DescribeSubnets", "ec2:DescribeInternetGateways",  
                                  "ec2:DescribeEgressOnlyInternetGateways", "ec2:DescribeVpcEndpoints",  
                                  "ec2:DescribeTransitGateways", "ec2:DescribeTransitPeeringConnections"]}  
  ]  
}
```

```
"ec2:DescribeNatGateways", "ec2:DescribeCustomerGateways",
"ec2:DescribeVpnGateways", "ec2:DescribeVpnConnections",
"ec2:DescribeRouteTables", "ec2:DescribeAddresses",
"ec2:DescribeSecurityGroups", "ec2:DescribeNetworkAccls",
"ec2:DescribeDhcpOptions", "ec2:DescribeTags",
"ec2:DescribeInstances"],
"Resource": "*"
}
]
}
```

Similarly, you can create more such policies based on your needs and assign them to specific users for controlling access to the VPC.

VPC networking components

A VPC network consists of certain components. We will now look briefly at these components.

ENI

ENI stands for **Elastic Network Interface** and is a virtual network interface. It is a communication hub for an EC2 instance that enables network communication on an instance. An EC2 instance can have one or more network interfaces. When any EC2 instance is created inside a VPC, a network interface is also created by default and attached to it. The default network interface created while launching an instance is called a primary network interface of the instance. This primary network interface also gets one primary IPv4 address from the subnet's available IP range.

You cannot detach a primary network interface from an EC2 instance and attach it to another. However, although you cannot detach a primary network interface, AWS allows us to create additional network interfaces that can be attached to the EC2 instance. Additional network interfaces are also called secondary network interfaces. Secondary network interfaces can be detached from one EC2 instance and attached to another EC2 instance. During this transition from one EC2 instance to another, its attributes remain intact.

A network interface can have the following attributes:

- Usually, a default network interface (that is, a primary network interface) is referred to as **eth0**.
- One primary private IPv4 address is automatically associated with it from the available range of IPs in its respective subnet.
- One Elastic IPv4 address per private IPv4 address can be attached to a network interface. Similarly, one public IPv4 address can be auto-assigned to a primary private IPv4 address.
- One or more IPv6 addresses can be attached to an ENI.
- One or more security groups can be attached to a single network interface. Currently, AWS allows a maximum of five security groups with a single network interface.
- A **media access control (MAC)** address is associated with each network interface.
- For easy identification, you can also specify a description of each network interface.
- The number of network interfaces and secondary private IPs attached to an ENI depends on the EC2 instance type. In general, the bigger the instance type, the more network interfaces and the more secondary private IPs that can be attached to it.
- You can attach multiple network interfaces to an instance for creating a management network or for implementing network and security appliances in a VPC.

Route tables

A route table is a set of rules that determines how the network traffic is routed. These sets of rules are also called routes. Routes determine how data packets travelling over an IP network are directed. A route table is a very important VPC resource for controlling the flow of network traffic.

For every VPC, there should be a main route table. While creating a custom VPC automatically, a main route table is also created. Each subnet in the VPC must be associated with a route table that controls routing for it. A subnet can only be associated with one route table, while a route table can be associated with many subnets. While creating a subnet, if it is not explicitly associated with any route table, then it is associated with a VPC's main route table. The main route table controls the network traffic for all the subnets that are explicitly not attached to any route tables. A route table attached to subnets and its respective route entries can be changed any time as per requirements.

It is recommended that you create a custom route table rather than directly make permanent changes in a VPC's default route table. Even if it is required to change the default route table, it is best practice to perform tests in a custom route table before updating the default route table. Once the results are satisfactory and are not disrupting network communication, then the change can be applied to a VPC's default route table.

Some important points about route tables are as follows:

- Every VPC has an implicit router.
- Every VPC has a main route table. It cannot be deleted, but it can be complemented with other additional route tables. Also, route entries in a main route table can be changed.
- Each route entry in a route table specifies a destination CIDR and a target. A target indicates that traffic for this specific CIDR block should be routed through this specific VPC resource such as an IGW, a VGW, a NAT device, a peering connection, or a VPC endpoint. For example, internet traffic (`0.0.0.0/0`) should be routed through an IGW (`igw-id`).
- Every route table contains one route entry to communicate within the VPC over IPv4. It allows communication from one subnet to another subnet, such as communication between web or application servers deployed in a public subnet and DB servers or RDS hosted in a private subnet.
- Currently, AWS allows a hard limit of 200 route tables per VPC and a maximum of 50 routes per table.

Selection of the optimum route for network traffic is done based on the longest prefix match; in other words, the most specific routes that match the network traffic. This is explained in the following example route table:

Destination	Target	Description
172.168.0.0/16	local	Network traffic for the 172.168.0.0/16 IPv4 address range; destination will be routed only within the VPC.
0.0.0.0/0	igw-abcd1236	Network traffic for the 0.0.0.0/0 internet IPv4 address to be routed to the IGW (igw-abcd1236).
172.31.0.0/16	pcx-1234abcd	Network traffic for the 172.31.0.0/16 IPv4 address points to a peering connection (pcx-1234abcd).



If route entries for a VPN connection or AWS Direct Connect overlap with local routes for VPC, local routes are given priority.

IGWs

An IGW is a virtual router in a VPC and an important stopping point for data on its way to and from the internet. An IGW makes it possible for EC2 instances to communicate with the internet. As a result, we can take a remote access (that is, RDP or SSH) of the EC2 instance available in public subnets. By deploying NAT instances in a public subnet, it can also provide an internet connection to the EC2 instance in a private subnet. In subsequent topics, we will see NAT in more detail. An NAT instance prevents unsolicited access from the internet to the instances in a private subnet. IGWs are highly available, redundant, and horizontally scaled in nature. This ensures that there is no bandwidth constraint and no availability risk.

The following points are critical for enabling internet access in a VPC subnet:

- The VPC must be attached to an IGW.
- All public route tables must point to the IGW.
- Instances in a subnet need to have a globally unique IP address, be it an IPv4 address, Elastic IP address, or IPv6 address.
- NACLs and security groups should allow traffic to and from the internet.

Figure 4.16 illustrates how traffic from public subnet 1 ($10.0.0.0/24$) is routed to the internet. A custom route table entry for internet traffic ($0.0.0.0/0$) pointing to `igw-id` enables the subnet to access the internet:

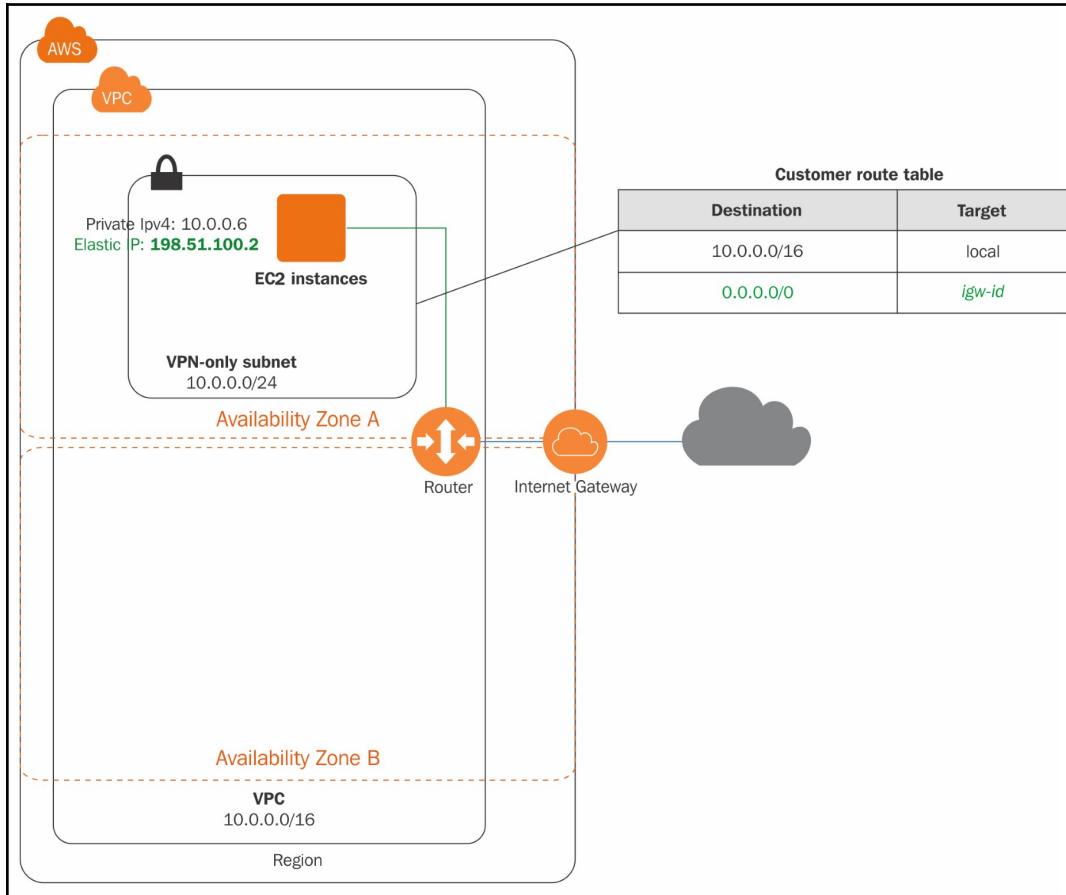


Figure 4.16: IGW

Egress-only IGWs

As with IGWs, egress-only IGWs are also highly available, redundant, and horizontally scaled. They only work with IPv6; they don't work with IPv4 addresses. For IPv4 network traffic, you can use an NAT gateway instead of egress-only internet traffic.

An egress-only IGW ensures that EC2 instances can communicate with the internet; however, these instances are not directly accessible from the internet. The main purpose of an egress-only IGW is to secure the subnet against internet traffic, and, at the same time, enable the instances in a subnet to access the internet.

For enabling egress-only internet traffic, you need to create an egress-only IGW in VPC and add a route to the route table pointing all IPv6 traffic to (`::/0`) to the egress-only IGW. It is a stateful VPC resource. An NACL can be used to control network traffic for an egress-only IGW.

NATs

An NAT can be defined as a virtual router or a gateway in a VPC that enables instances in a private subnet to interact with the internet. It's an important stopping point for data on its way from private subnets to the internet without directly exposing the instances to the internet. It acts as a firewall, dynamically assigns a temporary public address to an instance, and routes the traffic between the requesting instances and the internet.

There are two types of NAT devices:

- **NAT gateway:** This is the gateway service provided and managed by AWS.
- **NAT instance:** This is a custom-provisioned EC2 instance hosting NAT services.

These NAT devices only support IPv4 network traffic. EC2 instances in a private subnet do not have a public or an Elastic IP, and a subnet's route table does not have route entry to send traffic directly to the internet through an IGW. The NAT device acts as an intermediate point between instances and IGWs. It receives traffic from an EC2 instance residing in a private subnet and, before forwarding the traffic to the internet, it replaces the reply-to IPv4 address with its own public or Elastic IP address. When a reply is received from internet, again it changes the reply-to address from its IP address to the EC2 instance private IP address.

NAT devices do not support IPv6 traffic. If you need to use an IPv6 address for an EC2 instance, you can use an egress-only IGW instead of an NAT device, restricting unsolicited connection requests from the internet.

In an enterprise's AWS cloud infrastructure, either an NAT gateway or an NAT instance is deployed in a public subnet. An NAT instance can have a public or Elastic IP, while the NAT gateway requires an Elastic IP. To route traffic from a private subnet to the internet, there should be a route entry in a private subnet's route table, pointing internet traffic to the NAT instance or NAT gateway.

An NAT gateway is provisioned in a public subnet of an AZ in a VPC. Multiple private subnets in a single AZ can use the same NAT gateway. If an AZ fails, the NAT gateway residing in that AZ fails and all the subnets using the NAT gateway cannot reach the internet. It is advisable to create an NAT gateway in multiple AZs for failover.

Unlike an NAT gateway, there is an additional setting when you create an NAT instance. To facilitate internet requests, you need to disable the source and destination check on the NAT instance. By default, before sending traffic to the internet, every EC2 instance checks that its public address is in the source or destination address. But, in the case of an NAT instance, it sends and receives traffic on behalf of EC2 instances residing in a private subnet. In this scenario, if the source/destination check is not disabled on the NAT instance, it cannot serve the requests from other EC2 instances. If you ever find that EC2 instances in your private subnet are not able to connect to the internet, do check whether the source/destination check on the NAT EC2 instance is disabled.

The following are the steps to disable the source and destination check on an NAT instance:

1. Log in to the AWS web console using the appropriate credentials.
2. Navigate to the EC2 dashboard.
3. In the left-hand navigation pane, select **Instances**.
4. On the right-hand side upper pane, from the list of EC2 instances, select **NAT instance**.
5. From the top toolbar, select **Actions** or right-click on the selected instance. Select **Networking** and then **Change Source/Dest. Check** from the available options.
6. The subsequent dialog box gives provision to enable or disable the source/destination check.

The characteristics of an NAT gateway are as follows:

- An NAT gateway supports burstable bandwidth up to 10 GBps. Higher than 10 GBps bandwidth can also be achieved by deploying AWS resources into multiple subnets and creating an NAT gateway for individual subnets.
- An NAT gateway supports the TCP, UDP, and ICMP protocols.
- Creating an NAT gateway automatically creates a network interface and allocates a private IP from the subnet's address range. One Elastic IP address also has to be attached. Once an NAT gateway is deleted, the attached Elastic IP gets released from the NAT gateway, but remains reserved in the AWS account.

The following diagram helps us to visualize an NAT gateway:

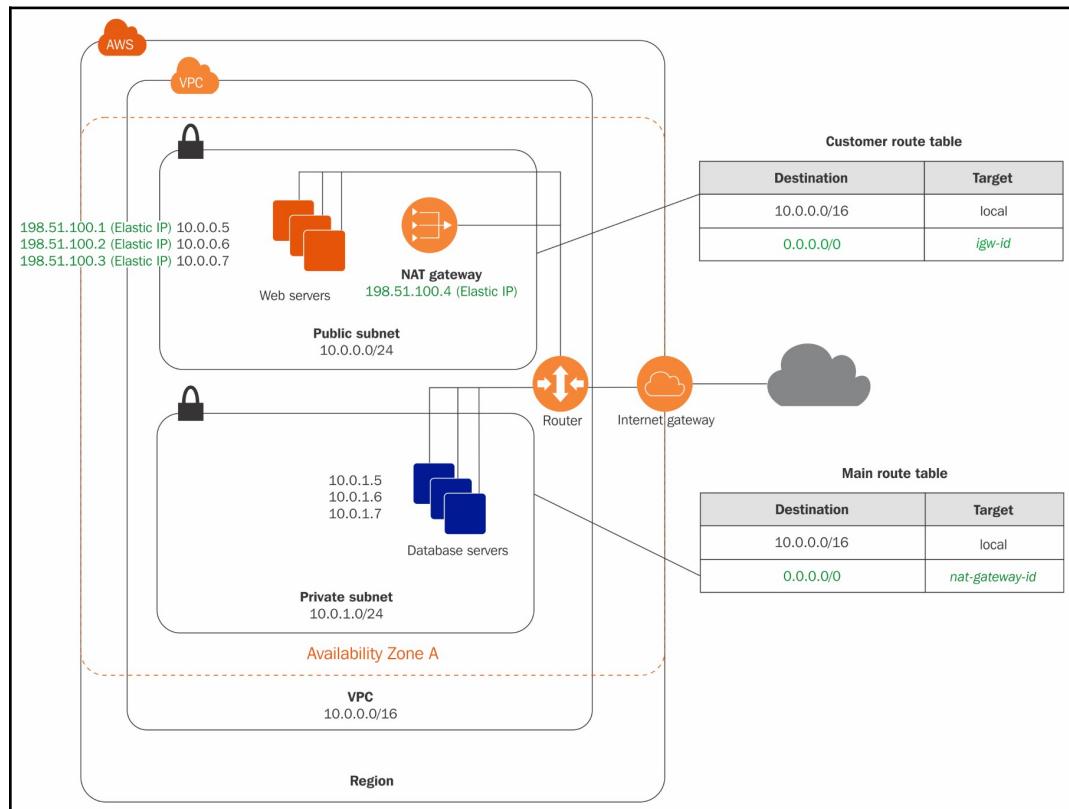


Figure 4.17: NAT gateway architecture

As you can observe from *Figure 4.17*, `10.0.1.0/24` is a private subnet. A route table entry is added to the main route table with `0.0.0.0/0` as the destination is routed to **nat-gateway-id**. Unlike private subnets, the public subnet's route table has **igw-id** as the target for internet traffic. This means the private subnet routes the internet traffic to an NAT gateway and the public subnet routes the traffic directly to the IGW.

Comparison of NAT instances and NAT gateways

It is highly recommended that you use an NAT gateway rather than an NAT instance as it provides better availability and higher bandwidth compared to an NAT instance. If you opt for an NAT instance, bandwidth depends on the EC2 instance type. Also, you have to design a custom failover logic for NAT instances, unlike an NAT gateway. An NAT gateway is managed by AWS and does not require you to perform maintenance activity. An NAT gateway being a managed service, Amazon takes care of maintenance activities. Amazon optimizes NAT gateways for performance and cost-effectiveness.

The following troubleshooting steps may help in certain situations where some instances in a VPC are unable to route traffic though an NAT instance:

- Ensure that there is a route entry in the route table of each private subnet for routing internet traffic to `nat-gateway-id`.
- Ensure that the source/destination check is disabled on the NAT instance.
- If some instances from a private subnet are able to access the internet and some instances are not, it may be because of the NAT instance's network bandwidth limitation. Ensure that the NAT instance type provides the expected bandwidth required to serve the traffic. You may need to change the instance type for better network bandwidth.



It is recommended that you use an NAT gateway over an NAT instance. An existing AWS cloud infrastructure with an NAT instance can easily migrate to an NAT gateway by creating an NAT gateway and changing the route table entries to point to `nat-gateway-id`.

DHCP option sets

DHCP is a network protocol that dynamically assigns IP addresses to instances in a VPC from the respective subnet's CIDR block. It also passes configuration information such as domain name, DNS, and NTP server. This configuration information is called a DHCP option set. A DHCP option set is essential for any newly created EC2 instance in a VPC. Every VPC has one DHCP option set. When creating a custom VPC, it automatically associates a default DHCP option set for the VPC. Once the DHCP option set is created, its parameters cannot be modified. You need to create new DHCP option set and associate it with the VPC. If you do not want to automatically assign configuration to a VPC, no DHCP option set can also be configured. As soon as the DHCP option set changes for any VPC, existing and new EC2 instances start using new DHCP option sets. Changing the DHCP option set does not require restarting EC2 instances. A DHCP option set allows the configuration of the following parameters:

DHCP option name	Description
domain-name	Domain names are very useful in various networking and application-specific naming and addressing purposes. Usually, an organization's name or abbreviated name is used as a high-level domain name. By default, an Amazon-provided DNS is <code>ec2.internal</code> for <code>us-east-1</code> (N. Virginia) and in other regions it is <code>region.compute.internal</code> . For example, in the Tokyo region, it is <code>ap-northeast-1.compute.internal</code> .
domain-name-servers	This is also referred to as a DNS. It can be either an Amazon-provided DNS or a maximum of four IP addresses of a custom DNS separated by commas. It is essential to have a custom DNS when you want to use custom domain names for instances in a VPC.
netbios-name-servers	Network Basic Input/Output System (NetBIOS) is a legacy networking protocol. A NetBIOS name server is required in case an enterprise runs some legacy applications requiring WINS. A maximum of up to four comma-separated IPv4 addresses can be specified for the NetBIOS server.

ntp-servers	A Network Time Protocol (NTP) is used to synchronize the clocks of computers to some time reference. In an organization, it is essential to have all the computers using the same clock settings. Having the same clock on all computers not only helps for log auditing and understanding event sequence, but also fulfills a base requirement to run many enterprise-grade applications. You can specify a maximum of up to four NTP servers' IPv4 addresses, separated by commas.
netbios-node-type	This is a method that computers use to resolve a NetBIOS name into an IP address. In general, a node type can be any of the following types: 1: This is for a broadcast node, also called B-node. 2: This is for a peer-to-peer node, also called P-node. 4: This is for a mixed node, a combination of B and P nodes, but, by default, it functions as a B-node. 8: This is for a hybrid node, a combination of a P-node and a B-node, but, by default, it functions as a P-node. It is recommended you use value 2 (peer-to-peer) as value 1, 4, and 8 with broadcast and multicast are not currently supported.

DNS

Usually, DNS and DHCP work together to translate a domain name to an IP address and vice versa. It also provides configuration information to instances within the network. Every EC2 instance in a VPC requires a unique DNS name. A DNS name is a combination of a hostname and a domain name. For any instance to communicate over the internet, it requires a public IP, and it requires a private IP to communicate within a local network.

When launching a new EC2 instance inside a default VPC, it gets a private and public DNS hostname corresponding to private and public IPv4 addresses by default. However, when an EC2 instance is launched in a custom VPC, it gets only a private DNS hostname and a corresponding IPv4 address by default. An internal DNS (private Amazon-provided) provides a DNS hostname to the EC2 instances in the form of `ip-<private-ipv4- address>.ec2.internal` in N. Virginia (us-east-1), and `ip-<private-ipv4- address>.region.compute.internal` for other regions. A public DNS (external Amazon-provided) is in the form of `hostname ec2-public-ipv4- address.compute-1.amazonaws.com` in N. Virginia (us-east-1), and `ec2-public-ipv4- address.region.amazonaws.com` for other regions.

For a custom VPC, the behavior of a DNS can be modified with the help of the following parameters:

Attribute	Description
enableDnsHostnames	A parameter value can be true or false. When it is true, an EC2 instance in the VPC gets public DNS hostnames. When this parameter is true, it also requires the enableDnsSupport attribute to be true.
enableDnsSupport	A parameter value can be true or false. It indicates whether DNS resolution (host to an IPv4 address and vice versa) is supported or not. If it is set to false, it does not resolve public hostnames to an IPv4 address.

By default, in a default VPC, both of the preceding parameters are set to true. However, in the case of a custom VPC, only the enableDnsSupport parameter is, by default, set to true. When both of the preceding parameters are set to true, then an EC2 instance gets a public DNS hostname and it can resolve an Amazon-provided private DNS hostname. When both parameters are set to false, the instance does not get a public DNS hostname, the Amazon-provided private DNS server cannot resolve the private hostname to IP, and the EC2 instance gets a custom private DNS hostname if the custom domain name is specified in a DHCP option set.

At the time of writing, AWS VPC does not provide DNS for IPv6 addresses.

VPC peering

VPC peering is a way to connect two different VPCs for routing traffic between them using IPv4 or IPv6 addresses. Once a VPC peering connection is established between two VPCs, instances in either of these VPCs can communicate with each other as they communicate with local instances within the same VPC.

By default, network traffic either flows within the same VPC or to and from the internet, but it does not route to other VPCs. If there is a need to route traffic between two VPCs, a VPC peering connection can be established. VPC peering can be used between two VPCs within the same region or different regions, irrespective of whether they belong to the same AWS account or a different AWS account. Communication among peered VPCs takes place through routing.

Network traffic does not flow through any separate VPC resources such as gateway or VPN connections.

Let's look at the following critical points for enabling VPC peering:

- The owner of VPC1 initiates a peering request for VPC2.
- VPC1 and VPC2 can belong to the same AWS account or to a different account. The two VPCs can belong to the same owner or to different owners.
- The two VPCs cannot have CIDR blocks that overlap with each other. The VPC2 owner accepts the peering request.
- For enabling traffic between the two VPCs, both the VPCs need to add a router entry in their respective route tables pointing to the IP address range of peering VPCs.
- Update the security groups of each VPC, where required, to enable traffic from the peering VPCs.
- If there is a need to resolve DNS hostnames from peering VPCs, you need to enable the DNS hostname resolution in the respective VPC configuration.
- By default, when instances in both sides of the VPC peering connection refer to each other using a public DNS name, it resolves to a public IP address of the target instances.

VPC endpoints

Generally, AWS services are different entities and do not allow direct communication with each other without going through either an IGW, an NAT gateway/instance, a VPN connection, or AWS Direct Connect. A VPC endpoint is an AWS service that enables you to create a private connection between different AWS services without going through the previously mentioned communication gateways.

Let's understand this scenario with some examples. In an enterprise infrastructure, an EC2 instance residing in a private subnet often needs to communicate with resources in other AWS services, for example, for storing and retrieving objects in S3. Before the launch of a VPC endpoint, you need to deploy an NAT device in a public subnet with an Elastic IP and route entry in the private subnet's route table. Such communication used to take place through the internet. Now, with the help of a VPC endpoint, there is no need to route traffic through the internet. It routes the traffic within the AWS infrastructure. A VPC endpoint is highly available, horizontally scaled, redundant, and easy to configure. It does not use any other VPC resources, such as VPN connection, NAT devices, or Direct Connect, for communication. It simply works using basic VPC components. There are no additional charges for using VPC endpoints. Only standard charges for resource usage and data transfer apply.

When creating a VPC endpoint, you need to define a source VPC and target AWS service. At the same time, a custom policy can be applied to fine-grain the access in a targeted AWS service. Enabling a VPC endpoint automatically identifies associated route tables with the source VPC and also automatically adds a route to each of the route tables. In route entry, the destination specifies the prefix list ID of the service (`p1-xxxxxxxx`), which represents the range of public IPv4 addresses used by the service. It indicates a target with the endpoint ID (`vpce-xxxxxxxx`). As a result, all the EC2 instances in all the subnets using any of the route table associated with VPC automatically start using this route to communicate with specific AWS services.

VPC endpoint limitations are as follows:

- It is not possible to tag VPC endpoints.
- It is not possible to transfer a VPC endpoint from one VPC to another VPC. You need to create new VPC endpoints for the desired AWS services in a related VPC and delete old, unwanted VPC endpoints.
- VPC endpoints support only within the same region. The source VPC and target resources of AWS services both must be in the same region.
- VPC endpoint connections just work within the VPC. They do not work with the VPN, VPC peering, Direct Connect, or ClassicLink connections.
- In order to use a VPC endpoint, it is essential to enable a DNS resolution in a VPC, whether using custom or Amazon-provided DNS servers.
- VPC endpoints prefix list IDs cannot be used in an NACL's outbound rule to allow or deny network traffic.

ClassicLink

Since December 4, 2013, AWS has supported EC2-VPC only. But, before that, it was possible to create EC2-Classic. EC2-Classic and EC2-VPC are both totally different in many ways. ClassicLink is the only way to make communication possible between them within the same region. Enabling the ClassicLink option allows a VPC to communicate with the EC2 instances launched in EC2-Classic. Without this option enabled, resources in the VPC need to use the public IP address of the EC2-Classic instance or tunneling for communication. If you have any resources in EC2-Classic that require direct communication with VPC resources, ClassicLink can help.

Enabling ClassicLink allows you to use VPC security groups on the EC2-Classic instances, and, in turn, this enables communication between instances in EC2-Classic and VPC using private IPv4 addresses. It is available in all AWS accounts that support EC2-Classic instances. There is no additional charge for using ClassicLink; however, standard data transfer charges apply.

VPC best practices

The following list summarizes VPC best practices:

- Before starting to design and implement AWS VPC, it is essential to understand present and future needs. It is recommended that you plan your VPC architecture, considering the minimum requirement for the next two years. Once the infrastructure is created on a VPC, making any changes in the VPC requires redesigning and recreating infrastructure. Lateral changes in the design and infrastructure can be very time-consuming and expensive.
- It is suggested that you use a CIDR range as per RFC 1918. Also, make sure that a sufficient number of IP addresses are available in each subnet to match with present and future needs. Ensure that the CIDR range in AWS does not conflict with the CIDR range used in any other data center or VPC where you may have to establish a VPN or Direct Connect connection.
- Remember, AWS reserves five IP address for internal purposes. The first four and the last one are in an IP range.
- Create subnets to isolate resources as per the project requirement (that is, DMZ/Proxy, ELB, web applications, databases, and more); for example, have a public subnet to host internet-facing resources and a private subnet for databases that do accept web requests.
- Create multiple subnets (that is, public or private) in multiple AZs to host a multi-AZ infrastructure and avoid a single point of failure. By default, each subnet can communicate with every other subnet in the same VPC.
- Make sure that only required ports and protocols from trusted sources are allowed to access AWS resources using security groups and NACLs. Create individual security groups for each resource or the same type of resources. For example, create a single security group for web servers.
- Make sure that unwanted outgoing ports are not open in security groups. For example, a security group for a web application does not need to open incoming mail server ports.

- Control access to AWS VPC services using appropriate user segregation, groups, and policies.
- Use NACLs wisely. More rules in the NACL may bring performance implications to overall networks.
- While adding rules to the NACL, number each rule in increments of 10, 50, or 100. This gives the flexibility to add rules in between when required.
- Preferably, use an NAT gateway over an NAT instance. An NAT gateway provides redundancy and better network bandwidth.
- Rather than modifying a default route table and NACL, create a custom route table. By default, a route table is used for newly created subnets.
- Use Elastic IPs wisely. Unused Elastic IPs and IPs attached to stopped EC2 instances may incur charges to the AWS account.
- It is essential to disable the source/destination check when an NAT instance is configured.
- Use a bastion host to access private machines hosted in a private network in a VPC.
- Enable VPC flow logs for audit purposes. Studying flow logs from time to time highlights unauthorized attempts to access the resources.
- Never create security group rules allowing `0.0.0.0/0` unless it is unavoidable. It is highly recommended that you create customized security groups for each resource or group of resources.
- It is recommended that you do not allow UDP/ICMP ports for private instances in security groups.
- While defining a security group, use a target security group ID instead of using an IP address for restricting access, and instead of using CIDR as a target. Such approaches ensure that your environment security is not compromised or mismanaged, even if IP addresses change.
- VPC peering can be used to make communication between VPCs within the same account, different AWS accounts, or any two VPCs within the same region or different regions. Initially, VPC peering was supported only within the same region, but lately AWS supports VPC peering across the regions.

Summary

- Every AWS region comprises two or more AZs.
- Subnet is short for subnetwork. A network is subdivided into multiple logical parts for controlling access to individual logical subparts of the network, which are called subnets.
- A private subnet is a subset of a network wherein resources are isolated and restricted for access from within the VPC.
- Any incoming traffic from the internet cannot directly access the resources within a private subnet.
- Similarly, outgoing traffic from a private subnet cannot directly access the internet.
- A public subnet is a subset of a network wherein resources within a subnet are isolated and can be accessed from within the VPC as well as from the internet.
- A private IP address cannot be reached over the internet.
- When an instance is launched, Amazon uses DHCP to assign a private IP address to the instance.
- Unlike a private IP address, a public IP address can be reached over the internet.
- An Elastic IP address is a public IPv4 address that can be allocated to an AWS account.
- If you want to use a persistent IP address or need to manually assign/release the IP address, you need to use an Elastic IP address.
- By default, when you create an AWS account, Amazon automatically provisions a default VPC for you.
- Instances in a private subnet cannot access internet directly. You can configure an NAT gateway or an NAT instance for routing the private subnet traffic to the internet.
- Security groups and NACLs are for network security and flow logs are for network monitoring.
- A security group can be described as a virtual firewall that controls any traffic coming in or going out of the instances associated with the security group.
- Each security group can have a maximum of 50 separate rules for inbound and outbound traffic.

- An NACL acts as a virtual firewall at subnet level.
- Each subnet in a VPC must be associated with at least one NACL.
- Flow logs are a feature that enables you to track incoming and outgoing traffic from a VPC's network interfaces.
- ENI stands for **Elastic Network Interface** and is a virtual network interface. It is a communication hub for an EC2 instance that enables network communication on an instance.
- A route table is a set of rules that determines how the network traffic is routed.
- An IGW is a virtual router in a VPC and an important stopping point for data on its way to and from the internet.
- An NAT can be defined as a virtual router or a gateway in a VPC that enables instances in a private subnet to interact with the internet.
- DHCP is a network protocol that dynamically assigns IP addresses to instances in a VPC from the respective subnet's CIDR block.
- Usually, DNS and DHCP work together to translate a domain name to an IP address and vice versa.
- VPC peering is a way to connect two different VPCs for routing traffic between them using IPv4 or IPv6 addresses.
- A VPC endpoint is an AWS service that enables you to create a private connection between different AWS services without going through the previously mentioned communication gateways.

5

Getting Started with Elastic Compute Cloud (EC2)

In the last decade of the 20th century, the world started rapidly moving toward computerization. Now that we are nearing the end of the second decade in the 21st century, the word *compute* has become a necessity in almost every walk of life. Be it mobiles, phablets, tablets, laptops, desktops, high-end servers, cars, GPS systems, toys, digital advertisement boards, point-of-sale terminals, or a number of **Internet of Things (IoT)** devices, they all use compute power. *When computing has such a deep impact on our lives, how can any organization stay away from it?* For any enterprise, of any scale, computers are one of the most essential IT resources. They provide a mechanism to host and run business applications.

The following topics will be covered in this chapter:

- Introducing EC2
- Pricing for EC2
- EC2 instance life cycle
- **Amazon Machine Images (AMIs)**
- Introducing **Elastic Block Store (EBS)**
- EC2 best practices

Introducing EC2

Technology is evolving, demand is increasing, competition is increasing, and organizations are continuously coming up with innovative solutions to serve their customers. This entire ecosystem is dependent on computing power. It has become critical for organizations to arrange highly available and reliable computing resources to run their businesses. Amazon's EC2 is a compute service that provides an on-demand and scalable computing service in the cloud. It eliminates the need for upfront investment on hardware with the *pay as you go* model. With such flexibility in provisioning computing resources, it makes it possible to develop and deploy applications faster. You can provision as many EC2 instances as you want, be it a single instance or tens or hundreds or thousands of servers based on your requirements. You pay only for what you use. If you do not require the provisioned instances, you can terminate them at will. You can scale up your fleet of servers or scale them down, based on what your business demands. You can configure security, manage networking, and add or remove storage as your business demands.

Some of the important aspects of EC2 are as follows:

- Amazon EC2 is a virtualized environment in the cloud.
- A provisioned EC2 resource is called an **instance**.
- You can create new instances based on AMIs.
- AMIs are preconfigured templates that include base operating systems and any additional software in a single package.
- It provides various combinations of CPU, memory, storage, and networking capacity for provisioning instances. These combinations are called **instance types**.
- It provides a highly secured mechanism to log in to your instances using key pairs. A key pair is a combination of private and public keys. When an instance is provisioned, a public key resides on the EC2 instance and a private key is provided to the user who provisions the server. A key pair is used for login to an instance associated with the key pair.
- Amazon provides temporary as well as persistent storage for your EC2 instances. Temporary storage is also called an **instance store**.
- Data on an instance store vanishes when the instances are stopped or restarted. Amazon also provides persistent storage volumes on EC2, which is called an **Elastic Block Store (EBS)**.

- Amazon provides multiple physical locations for provisioning EC2 instances, known as **Availability Zones (AZs)**.
- It provides a firewall to your instances using security groups.
- Security groups can control what port, protocol, and source IPs can access your instance.
- It provides a static address for your instances, which is called an **Elastic IP address**.
- It allows you to assign metadata to your instances, known as tags. Tags can be used to identify an instance. A tag can be a name, environment, department, cost center, and more.
- EC2 instances are created in an isolated virtual environment and can be associated with a **Virtual Private Cloud (VPC)**.

This chapter gradually touches upon all the critical points of EC2 in subsequent pages.

Pricing for EC2

If you just want to get started with working with EC2 and learning, Amazon provides EC2 in a free tier. It offers a `t2.micro` instance type to run for up to 750 hours per month. You can use the Amazon free tier for 12 months from the date of opening a new account. These 750 hours can be utilized either by one instance for 30 days around the clock, or by running 10 instances for 15 hours as you require.

When using instance types other than the free tier, charges apply on a per-hour basis and vary based on instance type, region, and payment option. A small instance type, with a smaller number of vCPUs and less memory, costs less compared to an instance type with more vCPUs and memory.

Amazon charges EC2 instances on a per-hour basis, and actual EC2 pricing depends on instance type, size, and payment model. There are four ways to pay for Amazon EC2 instances, as follows:

- On-demand
- Spot instances
- Reserved instances
- Dedicated hosts

On-demand

By default, EC2 hourly charges are applied at the on-demand rate. In this mode, compute (CPU and memory) is used as and when required. There is no need to have any long-term commitment. Compute capacity can be increased or decreased on the fly in order to meet business requirements.

Here are some examples of when to use on-demand instances:

- Usually, this payment mode is suitable when you create a new infrastructure in the cloud and are not sure what instance type and number of instances are required.
- You can even use on-demand instances for carrying out tests when low-cost and flexible compute is required without any upfront payment or long-term commitment.
- You can use on-demand instances for short-term applications with unpredictable workloads.

Here are some things to remember while using on-demand pricing:

- On-demand instances are charged on an hourly basis. It is the costliest pricing option available in AWS.
- When an instance is stopped, partial EC2 hours are rounded up for billing and you pay for a full hour.
- On-demand instances can be launched through the AWS Management Console.
- You can launch up to 20 instances at a time using the **RunInstances API**.

Spot instances

This allows us to bid for spare Amazon EC2 compute capacity. Usually, these computes can be up to 90% cheaper than on-demand instances. It requires you to bid, and it specifies the maximum price you want to pay per instance along with instance type and AZ. You get your spot instance, based on availability and current spot pricing in the specified AZ. The prices for instance types vary on the basis of the availability of spare capacity in the specified AZ.

In other words, it's all about the supply and demand ratio of spare capacity. As the demand for a specific instance type in spot instances increases, spot instance prices also increase. When the current spot instance price goes above your bid price, AWS terminates your spot instance. Before terminating EC2 spot instances, AWS gives a notification, two minutes prior to termination.

Here are some examples of when to use spot instances:

- When you need cheap resources for a temporary purpose
- When your application runtime is flexible and application workload can be interrupted
- When you need a large amount of additional computing capacity

Here are some things to remember while using spot instances:

- If a spot instance is terminated by AWS before completing an hour, you are not charged for that hour.
- If you terminate a spot instance in-between an hour, you're charged for that incomplete hour.
- Spot instances cannot be stopped and restarted. If a spot instance is stopped, it gets terminated.

Reserved instances

This provides a significant discount vis-à-vis the on-demand per-hour price. You can reserve an instance for a period of one or three years for your predictable resource demand. It offers three types of reservation request—**All Upfront**, **Partial Upfront**, and **No Upfront** payment options:

- With the All Upfront payment option, you can save up to 75% compared with on-demand pricing.
- In the Partial Upfront method, a partial amount of total billing is paid upfront and the remaining amount is paid on a monthly basis.
- The third option for reservation is with No Upfront cost, where you pay only on a monthly basis for your reserved instances. Even with a No Upfront reservation, you can save around 35% to 40% compared with on-demand pricing.

Here are some examples of when to use reserved instances:

- When you need to run a predictable and consistent workflow for an extended period.
- All applications require steady-state infrastructure.
- Users or organizations who can commit to a one- or three-year duration.

Here are some things to remember regarding reserved instances:

- When you reserve an EC2 instance, you reserve an instance type and not a specific instance. Discounted prices are automatically applied on a monthly bill basis on an instance type usage in the account.
- EC2 instances can be reserved at the AZ or at the region. A reserved instance at a regional level affords the flexibility to select an instance type and AZ.
- Payment mode (that is, All Upfront, Partial Upfront, or No Upfront) and term (one year or three years) can be selected. It is recommended to reserve for one year only, since AWS periodically reduces resource pricing.

Scheduled reserved instances

AWS allows you to purchase reserved EC2 instance capacity for a pre determined recurring schedule such as a daily, weekly, or monthly basis. You can use this reservation type for instances that you require only for a specific time in a particular day, week, or month. Each scheduled reserved instance purchase has a specified start time and duration. Scheduled reserved instances can be purchased for a one-year term. Once you reserve a scheduled instance capacity, you are charged for the scheduled time even if you do not use the instance:

- **When to use scheduled reserved instances:** The purchase of scheduled reserved instances is best suited to a workload that doesn't run continuously, but on a periodic schedule; for example, running batch processing that runs either at the end of the week or out of business hours.
- **Things to remember about scheduled reserved instances:** At the time of writing, only C3, C4, C5, M4, and R3 instance types are supported for scheduled reserved instances. Scheduled reserved instances can be reserved for a minimum of a one-year period and for a minimum utilization of 1,200 hours over the course of a year. You are charged for such reservations on your monthly billing cycle. It is important to note that scheduled reserved instances are automatically terminated when the scheduled time ends.

Dedicated hosts

In a normal scenario, when you launch an EC2 instance, it is provisioned in a virtual environment hosted on shared hardware. Though each instance has its own dedicated resources, it shares the hardware platform with other accounts. When using dedicated hosts, EC2 instances from the same AWS account are physically isolated on dedicated hosts. A dedicated EC2 instance using the same architecture may share hardware within the AWS account. It gives additional control over host hardware. It helps to meet corporate compliance and regulation requirements. The pricing model for the dedicated hosts can also be an on-demand, reserved (save up to 70%), or spot instance (save up to 90%) model.

A dedicated EC2 instance has two pricing components:

- Applicable price per hour, based on the pricing model chosen.
- Additionally, dedicated per-region fees. This figure is \$2, applicable per hour for at least one dedicated EC2 instance of any type running per region.

Here are some examples of when to use dedicated hosts:

- Use dedicated hosts when regulatory requirements mandate the use of dedicated hardware.
- Use dedicated hosts if you require more visibility in the physical host, especially for the **Bring Your Own License (BYOL)** model.
- Use dedicated hosts if your app requires minimal latency between instances. Though placement groups can also provide minimal latency, dedicated hosts are another way to address this requirement.

Here are some things to remember about dedicated hosts:

- AWS charges an hourly dedicated host rate based on the instance type it can host.
- Each dedicated host can provision a specific number of instances, depending on the instance size.
- Dedicated host pricing is provided in the following URL: <https://aws.amazon.com/ec2/dedicated-hosts/pricing/#on-demand>.
- Once you allocate a dedicated host, you are charged irrespective of the number of instances provisioned on the host.

- A dedicated host can only run the same instance type and size. You cannot mix and match instance types or even different instance sizes. If you allocate a C3 instance family-dedicated host, you pay for the physical host allocated, irrespective of how many C3 instances are provisioned on the host.
- AWS provides reserved pricing for dedicated hosts as well. You can use a reservation to reduce the total cost on dedicated hosts.

EC2 instance life cycle

An EC2 instance passes through various stages throughout its life cycle. It all starts with launching an EC2 instance using a specific AMI. The following diagram is an illustration of the EC2 instance lifecycle:

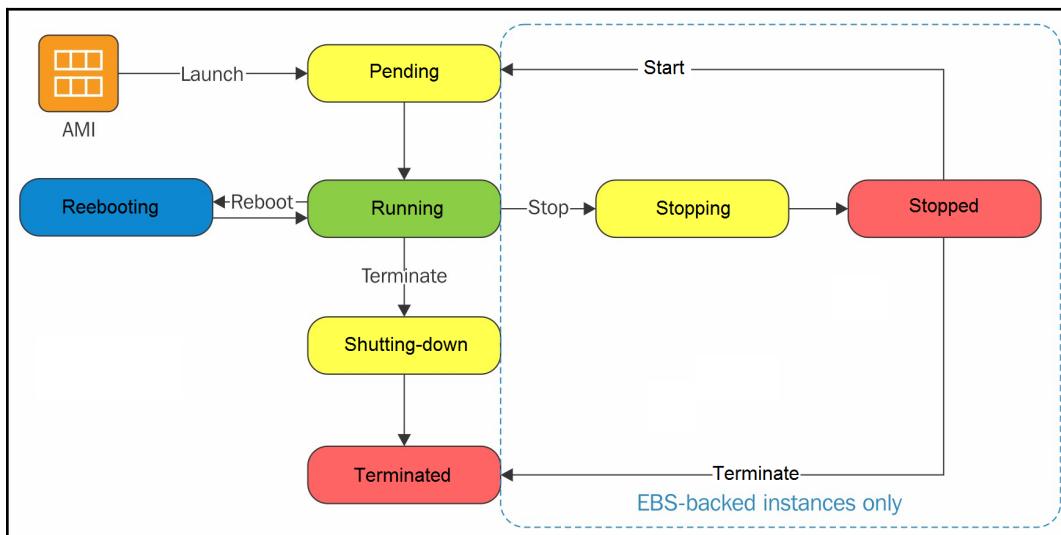


Figure 5.1: EC2 instance life cycle

The different stages of the instance life cycle that we will be looking into are as follows:

- Instance launch
- Instance stop and start
- Instance reboot

- Instance retirement
- Instance termination

Instance launch

When an instance is provisioned, it immediately gets into the **pending** state. Depending on what instance type you have selected, it is launched on a host computer inside AWS virtualized hardware. The instance is launched using the AMI you choose for provisioning. Once the instance is ready for use, it gets into the **running** state. At this moment, you can connect to your instance and start using it. AWS starts billing you for each hour that instance is used once it enters the running state.

Instance stop and start

If you have launched an EC2 instance with an EBS-backed volume, you can stop and start your instance as needed. If your instance fails any status check and is unresponsive, stopping and starting the instance again may help occasionally.

When you stop an instance, AWS initiates the operating system shutdown process and the instance enters the **stopping** state. As soon as the operating system shutdown process completes, the instance enters the **stopped** state. Once an instance is in the stopped state, you are not charged for that instance. However, AWS does charge you for any EBS volume or Elastic IPs associated with that instance. There are certain configuration options that can be used only when the instance is in the stopped state. When the instance is in the stopped state, you can change the instance type or disassociate/associate the root volume of the instance.

When an instance is started back up, it enters the pending state. AWS usually moves the instance to another host computer once it is stopped and started again. The instance may remain in the same host computer if there are no hardware-related issues on the host computer. AWS adapts this approach for automatically resolving hardware-related issues on an instance.

If you are running an instance on EC2-Classic, AWS allocates a new IPv4 address every time an instance is stopped and started again. However, EC2 on a VPC retains its IP address even if it is stopped and started again.

Every time an instance transitions from the running to the stopped state, AWS charges a full billing hour. That means that your billing hour count increases every time you stop and start an instance. In view of these hourly charges, it is recommended that you exercise the stop and start options wisely. However, with the introduction of per-second billing in October 2017, hourly charges apply only to Windows and licensed Linux distributions. Instances with free Linux distribution carry per-second billing and are charged accordingly.

Instance reboot

An EC2 instance can be rebooted in a variety of ways. You can use the AWS console, command line tools, the AWS API, or you can restart the instance from the operating system. AWS recommends rebooting the instance instead of rebooting the operating system. Rebooting the instance is equivalent to rebooting the operating system. When an instance reboots, it remains on the same host computer in the virtualized environment. It retains its IP addresses and public DNS name. It also retains the data on its instance store.

Unlike stopping and starting an instance, rebooting does not initiate a new billing hour.

Instance retirement

If there is any irreparable issue in underlying hardware where an instance is hosted, AWS schedules the instance for retirement. The instance is automatically stopped or terminated by AWS when it reaches the scheduled retirement date. If your instance is an EBS-backed instance, you can stop and start the instance. Stopping and starting the instance automatically changes the underlying host and you can continue using the instance. If your instance has a root volume with an instance store-based volume, the instance gets terminated and you cannot use it again.

Instance termination

If an EC2 instance is no longer required, you can terminate the instance. AWS stops charging you as soon as your instance's status changes to **shutting-down** or **terminated**.

AWS provides an option called **termination protection**. If this option is enabled, users cannot terminate an instance without disabling the termination protection. AWS provides this option to prevent accidental termination of an instance.

Once an EC2 instance is terminated, it remains visible with a terminated status on the console for a while, and then automatically disappears from the console. Once an instance is terminated, it cannot be recovered. If a safe backup of the instance is taken, you can launch a new instance from the backup.

Every EC2 instance with EBS-backed volume supports an attribute that controls its behavior on shutdown. This attribute is called

`instanceInitiatedShutdownBehavior`. While launching the instance, you can select what happens once you have shut down the instance. You can select to stop the instance on shutdown or terminate it. The default behavior of an EC2 instance is to *stop* on shutdown.

While launching an instance and associating an EBS volume, EC2 provides an option against each EBS volume called `DeleteOnTermination`. If this attribute is selected against an EBS volume, it is automatically deleted when the instance is terminated.

As we have seen various stages for an EC2 instance during its life cycle, it is also important to understand its impact on AWS' monthly billing. The following table helps us to understand this:

Instance state	Description	Instance usage billing
Pending	Usually, an EC2 instance comes to a pending state when it is being created for the first time, or when it is started after being in the stopped state.	Not billed
Running	The EC2 instance is running and ready for use.	Billed
Stopping	The EC2 instance is being prepared to be stopped.	Not billed
Shutting-down	This indicates that the EC2 instance is shut down and cannot be used. The instance can be restarted at any time as and when required.	Not billed
Terminated	The EC2 instance has been permanently deleted and cannot be restarted.	Not billed



Irrespective of the EC2 instance state, some of the AWS resources may incur charges, such as the EBS volume.

While maintaining IT infrastructure on a cloud, there may be a scheduled hypervisor maintenance from AWS. It may require you to shift resources (that is, EC2 instances) from one hypervisor to another.

Amazon Machine Images (AMIs)

While launching an instance, you may have a specific requirement, such as an operating system, preinstalled software, or a number of EBS volumes and their respective size. To cater to such a requirement, AWS uses a feature called an **AMI**.

An AMI contains a set of information to launch an instance:

- It contains a template that includes information such as the operating system, application server, and any other application software.
- It contains launch permissions describing which AWS account can use the AMI to spin up the new instance.
- It also contains block device mapping, describing the volume information to be attached to the instance while launching.

You can specify the AMI while launching an instance. An AMI can be used to launch as many instances as required; however, an instance can be based on a single AMI. You can also use multiple AMIs as required to launch different instances. The following diagram helps to explain how multiple EC2 instances from a single AMI can be created:

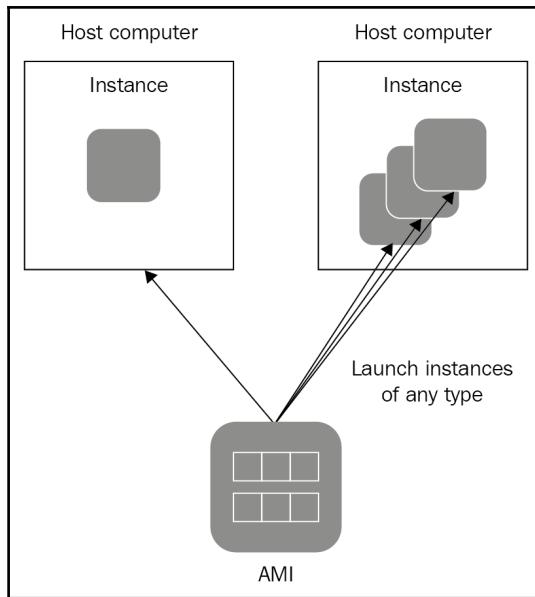


Figure 5.2: Concept of AMI and EC2 instance creation

Amazon provides a number of preconfigured AMIs in its marketplace. The AMIs in the marketplace include AMIs provided by Amazon with base configuration, community AMIs contributed by a large AWS community, and a number of AMIs with third-party software.

All AWS AMIs are internally stored in an S3 bucket. AWS protects these AMIs, and they are not directly visible on S3. You can choose the AMIs only when you launch an instance.

Apart from an underlying operating system and preconfigured software in the AMI, there are two more characteristics of AMI that are critical when choosing an AMI:

- Root device volume type
- Virtualization type

Before launching or planning an EC2 instance operating system in the enterprise architecture, it is necessary to understand what these characteristics are.

Root device types

While choosing an AMI, it is essential to understand the root device type associated with the AMI. A bootable block device of the EC2 instance is called a **root device**. As EC2 instances are created from an AMI, it is very important to observe the root device type at the AMI. An AMI can have either of two root device types:

- Amazon EBS-backed AMI (which uses permanent block storage to store data)
- Instance store-backed AMI (which uses ephemeral block storage to store data)

While creating an EC2 instance using a web console, we can see whether an AMI is EBS- or instance-backed. The following screenshot highlights the root device type while selecting an AMI:

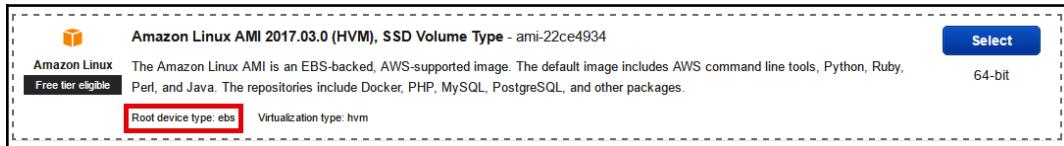


Figure 5.3: Root device type in an AMI

Amazon EBS-backed AMIs launch faster than instance-stored AMIs as you only need to create the volume from the snapshot, for booting the instance, while AMIs with ephemeral storage take a longer time to boot, as you need to load all the software on the ephemeral storage before booting the instance.

Ephemeral storage devices are directly attached to the host computer, which makes it faster in accessing the data; however, stored data gets wiped out in the event of restarting or shutting down the EC2 instance.

It is very important to remember that EBS-backed instances can be stopped; if ephemeral-based instances are stopped or terminated, the data stored on the ephemeral storage gets wiped from the storage.

EC2 instance virtualization types

Similar to a root device type, a virtualization type is another aspect of an AMI that it is critical to understand before choosing an AMI. An AMI can fall into either of the following two virtualization types:

- **Paravirtual (PV):** EC2 instance booted by PV-GRUB
- **Hardware Virtual Machine (HVM):** EC2 instance booted by the **Master Boot Record (MBR)**

The main difference between these two virtualization types is in their booting process and their ability to take advantage of special hardware extensions, for a better performance of CPU, network, and storage devices.

For the best EC2 instance performance, it is highly recommended that you use the latest instance type with HVM AMIs.

In the case of HVM, the operating system can run directly on virtual machines without any modifications. This makes HVM-based instances faster and more efficient. PV-based instances can run on hardware without virtualization support; however, they cannot take advantage of special hardware extensions. Special hardware extensions, such as enhanced networking or GPU processing, can make a huge difference when it comes to running certain application types. Prior to the advance in HVM technologies, PV-based instances used to perform better than HVM; however, as a result of technological improvements, HVM is now leading the race.

While creating an EC2 instance, you can see the type of virtualization against the AMI. The following screenshot highlights the virtualization type given against an AMI:

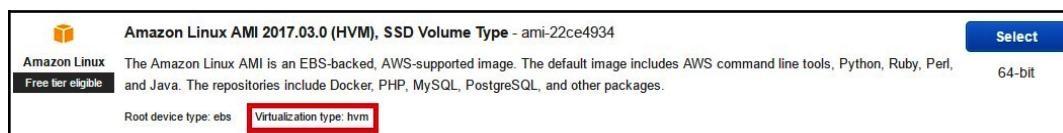


Figure 5.4: AMI virtualization type

Creating an EC2 instance

Now, since we have the basic information regarding EC2, its pricing model, and AMIs, let's understand how to create an EC2 instance using the AWS console.

The following steps describe the process of creating an EC2 instance:

1. Log in to the AWS console using valid credentials and go to the EC2 dashboard.
2. Click on the **Launch Instance** command button.
3. Select the appropriate AMI based on the operating system, root device type, and virtualization type. The screen that displays a number of AMIs to choose from is shown in *Figure 5.5*. By default, you can see a number of AMIs in **Quick Start**. If required, you can select **My AMIs**, which contains all the custom AMIs created in a user's account. If you select **AWS Marketplace** from the left-hand menu, it displays a number of third-party AMIs available in AWS Marketplace. Marketplace AMIs may not be free. You can also choose **Community AMIs** that contain a number of AMIs contributed by AWS community members. Alternatively, if you just want to create a free tier instance, you can select the checkbox against **Free tier only**:

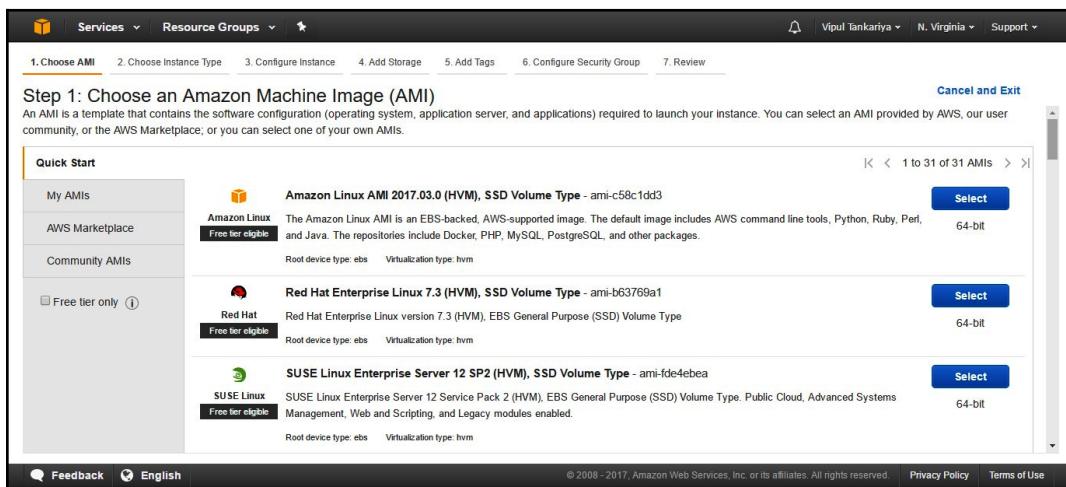


Figure 5.5: Selecting an AMI while launching an instance

4. Once you select an AMI, it displays on the screen, as shown in *Figure 5.6*. Select the appropriate instance type from the screen:

The screenshot shows the AWS EC2 instance creation wizard at Step 2: Choose an Instance Type. The 'Choose Instance Type' tab is selected in the top navigation bar. The main content area displays a table of available instance types, filtered by 'All instance types' and 'Current generation'. The table includes columns for Family, Type, vCPUs, Memory (GiB), Instance Storage (GB), EBS-Optimized Available, Network Performance, and IPv6 Support. The 't2.micro' row is highlighted with a blue background, indicating it is the currently selected instance type. This row also has a green 'Free tier eligible' badge. At the bottom of the table, there are buttons for 'Cancel', 'Previous', 'Review and Launch', and 'Next: Configure Instance Details'.

Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance	IPv6 Support
General purpose	t2.nano	1	0.5	EBS only	-	Low to Moderate	Yes
General purpose	t2.micro Free tier eligible	1	1	EBS only	-	Low to Moderate	Yes
General purpose	t2.small	1	2	EBS only	-	Low to Moderate	Yes
General purpose	t2.medium	2	4	EBS only	-	Low to Moderate	Yes
General purpose	t2.large	2	8	EBS only	-	Low to Moderate	Yes

Figure 5.6: Selecting an instance type

5. After selecting the instance type from the screen, click on **Next: Configure Instance Details**. The subsequent screen provides options to **Configure Instance Details**, such as the number of instances to launch, payment option (spot or on-demand), VPC and subnet, public IP, IAM role, shutdown behavior, termination protection, advanced monitoring, and user data.

The following screenshot displays the screen with options to configure instance details:

The screenshot shows the AWS EC2 instance configuration interface. At the top, there are tabs: 1. Choose AMI, 2. Choose Instance Type, 3. Configure Instance (which is selected), 4. Add Storage, 5. Add Tags, 6. Configure Security Group, and 7. Review. The main section is titled "Step 3: Configure Instance Details" with the sub-instruction "Configure the instance to suit your requirements. You can launch multiple instances from the same AMI, request Spot instances to take advantage of the lower pricing, assign an access management role to the instance, and more." Below this, there are several configuration groups:

- Number of instances:** Set to 1. [Launch into Auto Scaling Group](#)
- Purchasing option:** Request Spot instances
- Network:** vpc-7e50d016 (default) [Create new VPC](#)
- Subnet:** No preference (default subnet in any Availability Zone) [Create new subnet](#)
- Auto-assign Public IP:** Use subnet setting (Enable)
- Placement group:** Add instance to placement group.
- Capacity Reservation:** Open [Create new Capacity Reservation](#)
- IAM role:** None [Create new IAM role](#)
- Shutdown behavior:** Stop
- Enable termination protection:** Protect against accidental termination
- Monitoring:** Enable CloudWatch detailed monitoring
Additional charges apply
- Tenancy:** Shared - Run a shared hardware instance
Additional charges will apply for dedicated tenancy.
- Elastic Inference:** Add an Elastic Inference accelerator
Additional charges apply.
- T2/T3 Unlimited:** Enable
Additional charges may apply

At the bottom left is a link to "Advanced Details". At the bottom right are buttons: Cancel, Previous, **Review and Launch** (highlighted in blue), and Next: Add Storage.

Figure 5.7: EC2 instance configuration details

6. Add additional EBS volumes as required. Amazon allows up to 30 GB of general-purpose volume in the free tier. Also, while creating the EC2 instance, at this stage, it is possible to change the **Delete on Termination** option to true or false for each EBS volume, including the root volume. Once an instance is created, you can change the **Delete on Termination** option for EBS only through the CLI or API. After selecting the appropriate option, click on the **Next: Add Tags** button, as shown in the following screenshot:

Step 4: Add Storage

Your instance will be launched with the following storage device settings. You can attach additional EBS volumes and instance store volumes to your instance, or edit the settings of the root volume. You can also attach additional EBS volumes after launching an instance, but not instance store volumes. [Learn more about storage options in Amazon EC2.](#)

Volume Type	Device	Snapshot	Size (GiB)	Volume Type	IOPS	Throughput (MB/s)	Delete on Termination	Encrypted
Root	/dev/xvda	snap-0f2b695076fc43043	8	General Purpose SSD (GP2)	100 / 3000	N/A	<input checked="" type="checkbox"/>	Not Encrypted

[Add New Volume](#)

Free tier eligible customers can get up to 30 GB of EBS General Purpose (SSD) or Magnetic storage. [Learn more](#) about free usage tier eligibility and usage restrictions.

[Cancel](#) [Previous](#) [Review and Launch](#) [Next: Add Tags](#)

Figure 5.8: Adding additional EBS volumes

7. In the subsequent screen, you can add tags to your EC2 instance. Amazon assigns a distinct instance ID to every EC2 instance for uniquely identifying an instance. On top of that, you can also add additional tags to the instance to group them based on environment; that is, development, testing, preproduction or production, and more. These tags are key-value pairs and are case-sensitive.
8. While creating the tags, by checking against the **Volumes** column, AWS associates the same tags with each relevant EBS volume associated with the instance. You can see the **Volume** column in the next screenshot. Click on **Next: Configure Security Group** for the next screen:

Step 5: Add Tags

A tag consists of a case-sensitive key-value pair. For example, you could define a tag with key = Name and value = Webserver.
A copy of a tag can be applied to volumes, instances or both.
Tags will be applied to all instances and volumes. [Learn more about tagging your Amazon EC2 resources.](#)

Key	(127 characters maximum)	Value	(255 characters maximum)	Instances	Volumes
Name		Learn launch EC2 instance		<input checked="" type="checkbox"/>	<input type="checkbox"/>

[Add another tag](#) (Up to 50 tags maximum)

[Cancel](#) [Previous](#) [Review and Launch](#) [Next: Configure Security Group](#)

Figure 5.9: Adding tags

9. The following screen shows the options provided to **Configure Security Group**. You can open the required port on a specific protocol and source IPs. Generally, inbound rules are defined based on what ports and protocols are used by the application hosted on the server. You can either use an existing security group or you can create a new one based on the requirement. The following *Figure 5.10* shows security group configuration options. After configuring the security group, you can click on **Review and Launch**:

Step 6: Configure Security Group

A security group is a set of firewall rules that control the traffic for your instance. On this page, you can add rules to allow specific traffic to reach your instance. For example, if you want to set up a web server and allow Internet traffic to reach your instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. You can create a new security group or select from an existing one below. [Learn more about Amazon EC2 security groups.](#)

Assign a security group: Create a **new** security group Select an **existing** security group

Security group name:

Description:

Type	Protocol	Port Range	Source
SSH	TCP	22	Custom <input type="text" value="0.0.0.0/0"/> <input type="button" value="X"/>

Add Rule

Warning
Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

Cancel **Previous** **Review and Launch**

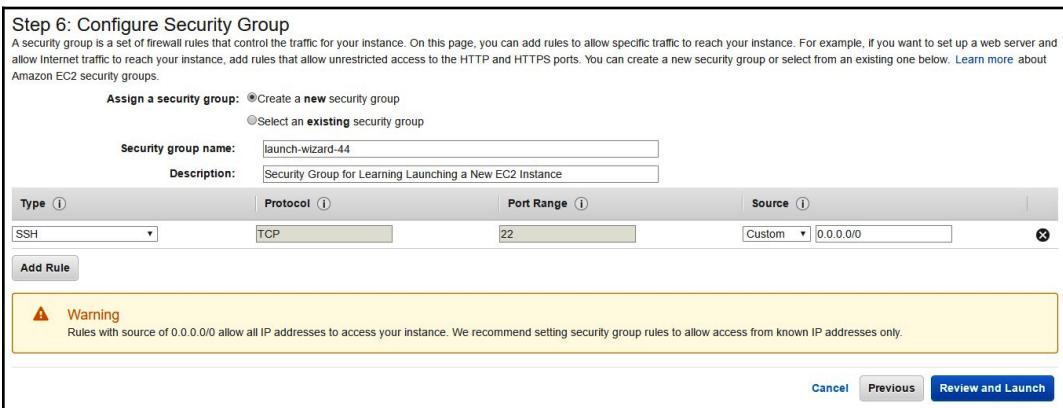


Figure 5.10: Configuring the security group

10. In the subsequent screen of the wizard, you can finally review the configuration options you have selected during the launch instance wizard. If required, you can click on the **Previous** button and modify the options as necessary:

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags 6. Configure Security Group 7. Review

Step 7: Review Instance Launch

Please review your instance launch details. You can go back to edit changes for each section. Click **Launch** to assign a key pair to your instance and complete the launch process.

▼ AMI Details Edit AMI

 **Amazon Linux AMI 2017.03.0 (HVM), SSD Volume Type - ami-22ce4934**
Free tier eligible The Amazon Linux AMI is an EBS-backed, AWS-supported image. The default image includes AWS command line tools, Python, Ruby, Perl, and Java. The repositories include Docker, PHP, MySQL, PostgreSQL, and other packages.
Root Device Type: ebs Virtualization type: hvm

▼ Instance Type Edit instance type

Instance Type	ECUs	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance
t2.micro	Variable	1	1	EBS only	-	Low to Moderate

▼ Security Groups Edit security groups

Security group name: launch-wizard-44
Description: Security Group for Learning Launching a New EC2 Instance

Type (i)	Protocol (i)	Port Range (i)	Source (i)
SSH	TCP	22	103.26.232.66/32

► Instance Details Edit instance details

► Storage Edit storage

► Tags Edit tags

Cancel **Previous** **Launch**

Figure 5.11: Reviewing the configuration options

11. After verifying all the options, you can click on the **Launch** button. Once you click on the **Launch** button, it asks you to **Select an existing key pair or create a new key pair**. Select an existing key pair or give a suitable key name to create a new key pair. Remember to download the key. The key is available to download during this wizard only. AWS does not provide an option to download the key later on. After providing the key pair detail, you can click on the **Launch Instances** button. It may take a few minutes for the instance to launch. The time for an EC2 instance to come to a running state depends on the AMI type and the instance type.

12. You can see all the instances on the EC2 dashboard. You can also see all the relevant EC2 properties by selecting a specific instance:

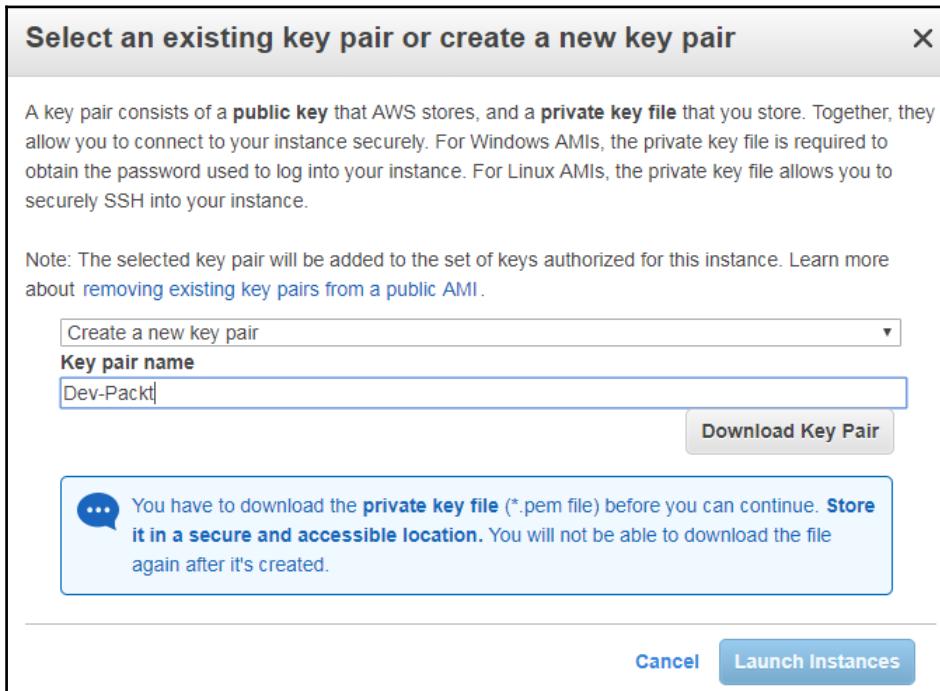


Figure 5.12: Downloading the key pair

Changing the EC2 instance type

Once an instance is launched, it may be required to change the instance type based on requirements. For example, you may need to change the instance type to accommodate high CPU and memory requirements. Perform the following steps to change the EC2 instance types. An instance can be changed only if the instance is in the stopped state. Shut down the instance from either the operating system or from the EC2 console and follow these steps:

1. Log in to the AWS dashboard using valid credentials and go to the EC2 dashboard.
2. Go to **Instances** and select the desired EC2 instance to change the instance type.

3. Shut down the EC2 instance. Once an EC2 instance is in the stopped state, right-click on the EC2 instance and change its type by going to **Instance Settings** | **Change Instance Type**, as shown in the following screenshot:

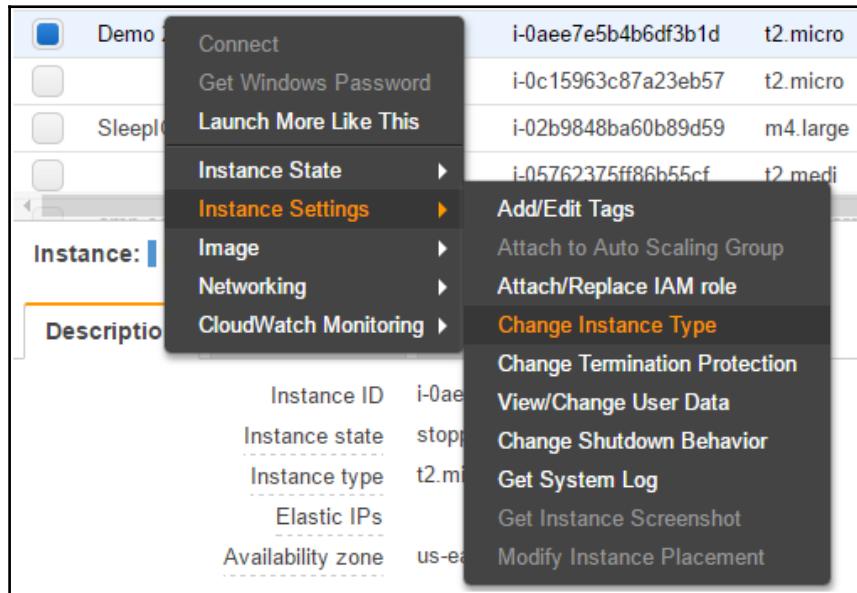


Figure 5.13: Changing the instance type

4. Once the instance type is changed, you can start the instance again. It may take some time for the instance to come back to the running state.

Connecting to the EC2 instance

To remotely connect to an EC2 instance in a public subnet, you need to know its public IP or Elastic IP address. To work with EC2 instances residing in a private subnet, you need to create a bastion host in a public subnet and attach an Elastic IP to access it. In order to connect to an instance in a private subnet, you first need to connect to a bastion host and then, from the bastion host, connect to the EC2 instances in a private subnet. By default, Linux-based EC2 instances can be connected on port 22 using tools such as PuTTY. Microsoft Windows EC2 instances can be connected on port 3389 using the Windows Remote Desktop utility. To connect to the Linux system, you need to pass a username, port, and private key. The public key is embedded inside the EC2 instance.

Default users for various Linux systems are given in the following table:

Operating system on EC2 instance	AMI (SSH username)
Amazon Linux	ec2-user
Bitnami	bitnami
CentOS	centos
Debian	admin
Fedora	fedora
FreeBSD	ec2-user
NanoStack	ubuntu
OmniOS	root
RHEL 6.3 and earlier	root
RHEL 6.4 and later	ec2-user
SUSE	root
TurnKey	root
Ubuntu	ubuntu

Connecting to a Linux EC2 instance from a Microsoft Windows system

The prerequisites for connecting to a Linux EC2 instance from a Microsoft Windows system are as follows:

1. Download PuTTY and PuTTYgen to the Microsoft Windows machine. You can get links to download PuTTY and PuTTYgen from <http://www.putty.org/>.
2. Get the public DNS or public/Elastic IP of the desired Linux instance to connect.
3. When IPv6 is assigned to an EC2 instance, connecting to it requires the source machine to also have an IPv6 address.
4. Keep the relevant private key file handy, which is downloaded while creating an instance.
5. Ensure that SSH port 22 is open in inbound rules of the security group assigned to the instance.
6. You need to convert the downloaded key file from .pem to a private key as .ppk.

Converting a PEM file to a private key (PPK)

The following steps describe how to convert a .pem file to a .ppk file:

1. Open PuTTYgen, and click on the **Load** button. Select all files from the drop-down menu and choose the appropriate .pem file that you need to convert to a .ppk file.
2. At the time of loading a file, default filtration is only executed on .ppk files. Change it to show **All Files (*.*)** to get a list of desired .pem files to load, as follows:



Figure 5.14: Loading .pem files

3. In the same PuTTYgen screen, make sure that the parameters are configured to store a public key as an RSA format and a bit size of 2048:



Figure 5.15: Parameters to save a private key without a password phrase

4. Click on the **Save private key** button.

5. When saving the key, a warning dialog box may appear; you can select Yes in the event of this warning:

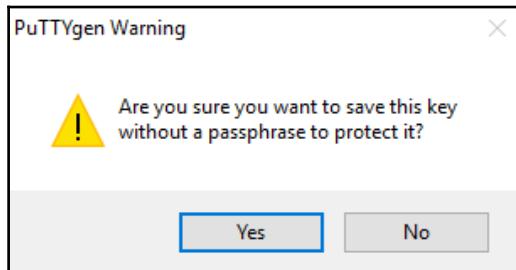


Figure 5.16: Warning dialog box

Finally, save the public key with the .ppk extension.

Connecting to an EC2 instance using a PuTTY session

Once you have a .ppk file, you are ready to connect to an AWS EC2 Linux instance. The following steps describe the process to initiate an SSH connection with an EC2 instance using PuTTY:

1. Run a PuTTY application on Microsoft Windows from where you need to connect to the EC2 instance.
2. In the **Category** pane, on the left-hand side, select **Session** and provide the following details:
 - The default port is 22 for Linux OS.
 - The connection type should be **SSH**.
 - The username should be the default SSH username based on the operating system type, as shown in the previous table, and public DNS or public/Elastic IP, as follows:

Consider the syntax, as follows:

<username>@<PublicIP>

For example, to connect to an RHEL/CentOS 7 EC2 instance, use the following command:

```
ec2-user@34.204.99.20:
```

The entire preceding configuration mentioned can be seen in the following screenshot:

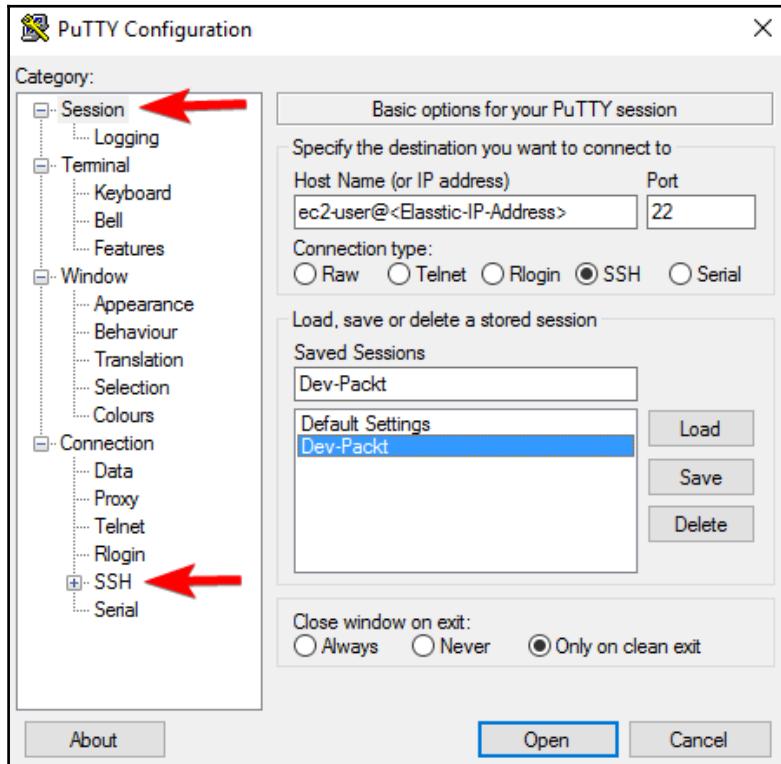


Figure 5.17: PuTTY session

3. On the left-hand side, in the **Category** pane, select **Connection** | **SSH** | **Auth** and click **Browse...** to provide a private key, as shown in the following screenshot:

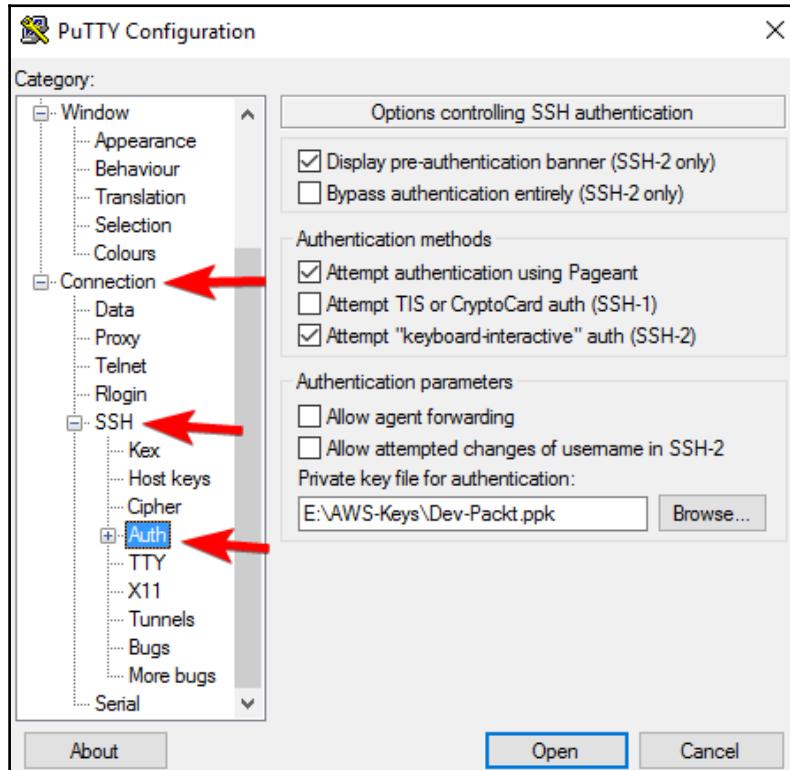


Figure 5.18: Providing a private key

4. Once you click **Open**, a security dialog box may appear confirming that you trust the host you are about to connect with. Choose **Yes** and the SSH connection will take place.

Troubleshooting SSH connection issues

While establishing an SSH connection with an EC2 instance, if, in spite of the fact that all the required details are properly provided, it fails to establish an SSH connection, check out the following points:

- Ensure that you are giving the correct IP address of the instance. Verify the username you have given, along with the IP address.
- Make sure an EC2 instance is up and running.
- Ensure that the security group has SSH port 22 open and is accessible. Check the operating system-level firewall and ensure it's not blocking the connection.
- If you are behind a network proxy, ensure that your network proxy is not blocking it.
- Ensure that you are using the correct .ppk file.
- After verifying all the preceding steps, if you are still not able to log in, you can try stopping and restarting the instance.
- You can also diagnose the issue by stopping the instance, detaching its root drive, and attaching and mounting it to another healthy EC2 instance as a secondary drive. Once the drive is attached to another EC2 instance, you can diagnose configuration issues.

EC2 instance metadata and user data

Metadata is data about an EC2 instance. EC2 instance details, such as AMI ID, hostname, instance ID, instance type, private IP address, and public IP address, are metadata of the instance. EC2 instance metadata can be retrieved by querying 169.254.269.254 on the same machine. From a Linux system, you can use the following command to retrieve the metadata from the EC2 instance:

```
$ curl http://169.254.169.254/
```

Issuing the `curl` command gives the following output, categorizing the metadata based on the date it was introduced by AWS:

```
1.0
2007-01-19
2007-03-01
2007-08-29
2007-10-10
2007-12-15
2008-02-01
```

```
2008-09-01
2009-04-04
2011-01-01
2011-05-01
2012-01-12
2014-02-25
2014-11-05
2015-10-20
2016-04-19
2016-06-30
2016-09-02
latest
```

Furthermore, the latest metadata is divided into three categories, as shown in the following output:

```
$ curl http://169.254.169.254/latest/dynamic
meta-data user-data
```

An EC2 instance's individual metadata properties can be retrieved by adding the `property` at the end of the following command:

```
$ curl http://169.254.169.254/latest/meta-data/
```

For example, you can retrieve `ami-id` by querying the following URL:

```
$ curl http://169.254.169.254/latest/meta-data/ami-id
```

Similarly, you can use the following list of properties with the `curl` command to retrieve its value:

```
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
```

On an Amazon Linux EC2 instance, the `ec2-metadata` command also gives metadata.

Placement groups

A **placement group** is a logical grouping of EC2 instances within a single AZ. A placement group provides the lowest possible network latency across all the EC2 instances that are part of the same placement group. Not all EC2 instances support high network throughput (that is, a placement group). Before launching an instance in a placement group, you need to ensure that the instance type supports a placement group. It is best practice to create all the EC2 instances required in a placement group, and ensure they are created in a single launch request and have the same instance type. If multiple instance types are mixed in a placement group, then the lowest bandwidth among the EC2 instances is considered to be the highest network throughput of the placement group. It is recommended that you choose an instance type that supports a 10 Gbps or 20 Gbps network throughput. There is no additional charge for creating an instance group.

A number of important points regarding placement groups include the following:

- A placement group can span peered VPCs.
- When network traffic is flowing to and from outside the placement group, network throughput is limited to 5 Gbps.
- An existing EC2 instance in the same AZ cannot be moved inside a placement group. You need to create an AMI of the existing instance and then launch a new EC2 instance inside the placement group.
- Even in the same account, placement groups cannot be merged.
- When you stop and start an instance inside a placement group, it remains in the same placement group.
- If you get a capacity error while launching an instance inside a placement group, you can stop and start all instances in the placement group. Stopping and starting instances automatically migrates the instances to other hardware that has the capacity to run them.

Introducing EBS

EBS is an AWS block storage service that provides block-level, persistent storage volumes for EC2 instances. EBS volumes are a highly available and reliable storage solution. An EBS volume can be attached only to the EC2 instances running in the same AZ. It provides persistent storage and is independent from the EC2 instance. This means that the data on the EBS volume remains intact even if the instance is restarted. AWS charges for the allocated EBS volume sizes, even if the volume is not attached to any instance.

Also, charges are based on the allocated volume size and not based on how much data is stored on the volume. EBS volumes can be formatted into the desired block size and filesystem. This is very suitable for read and write, such as database applications, or for the throughput of intensive workloads, such as big data. Once EBS volumes are attached to EC2 instances, they are used in the same way as a normal physical drive. Multiple EBS volumes can be attached to a single EC2 instance; however, one EBS volume can only be attached to a single EC2 instance.

AWS replicates EBS data at least three times within a single AZ. Unlike S3 data, it does not get replicated in multiple AZs within the same region. It is also important to understand that EBS volumes are not directly attached to the hosts (hypervisor), but that they are network-attached block storage.

Types of EBS

Currently, AWS provides the following types of EBS volumes. These EBS types have different performance and prices per GB:

- **Solid State Drive (SSD):**
 - General Purpose SSD (gp2)
 - Provisioned IOPS SSD (io1)
- **Hard Disk Drive (HDD):**
 - Throughput optimized HDD (st1)
 - Cold HDD (sc1)
- **Previous generation volume:**
 - Magnetic (standard)

General Purpose SSD (gp2)

gp2 volumes are one of the EBS volume types that provide persistent storage. gp2 volume types are ideal for a number of workloads. gp2 volumes are also very efficient and provide single-digit millisecond latencies. A gp2 volume is capable of bursting up to 3,000 IOPS for a significant amount of time. You can provision a minimum of 1 GiB size of gp2 volume and a maximum of up to 16 TiB of a gp2 volume. A gp2 volume provides 3 IOPS per GiB of volume size. However, if a volume size is 33.33 GiB or less, it provides a minimum of 100 IOPS. As you increase the volume size, the IOPS it provides also increases. However, a gp2 volume can provide a maximum of 16,000 IOPS. If you use multiple gp2 volumes in an instance, AWS imposes a limit of a maximum of 80,000 IOPS per instance.

The following list tells you where to use gp2 volumes:

- They are recommended for almost all workload types.
- They can be used as a root volume for an operating system.
- They can be attached to a virtual desktop.
- They can be used in interactive apps requiring low-latency storage.
- They can be used in development workloads.
- They can be used in testing environments.

Provisioned IOPS SSD (io1)

Provisioned IOPS SSD (io1) volumes are SSD volumes that are intended to address the needs of I/O intensive application workloads. io1 volumes are specifically used for database workloads that require high performance storage and consistent throughput. Unlike gp2 volumes, the io1 volume provides a consistent performance. You can specify a consistent IOPS rate while creating the volume. io1 volumes can provide maximum performance out of all other volume types. An io1 volume size can range from 4 GiB to 16 TiB. An io1 volume can have a minimum of 100 IOPS and a maximum of 64,000 IOPS. If you use multiple io1 volumes in an instance, AWS imposes a limit of a maximum of 80,000 IOPS per instance.

The following list tells you where to use io1 volumes:

- Mission-critical applications
- Business-critical applications requiring consistent performance
- Large database workloads, such as SQL Server and Oracle

Throughput optimized HDD (st1)

Throughput optimized HDD (st1) volumes are designed to provide a financially viable magnetic storage option. st1 volumes are architected to measure the performance in terms of throughput and not IOPS. The st1 volume type is recommended for a large and linear workload such as data warehouse, log processing, Amazon **Elastic MapReduce (EMR)**, and ETL workloads. It cannot be used as a bootable volume. An st1 volume size can range from 500 GiB to 16 TiB. An st1 volume can have a maximum of 500 IOPS per volume. If you use multiple st1 volumes in an instance, AWS imposes a limit of a maximum of 80,000 IOPS per instance.

The following list tells you where to use st1 volumes:

- Applications requiring consistent and fast throughput at a low cost
- Big data
- Data warehouse
- Log processing

Cold HDD (sc1)

Cold HDD (sc1) volumes are designed to provide a cost-effective magnetic storage option. sc1 volumes are designed to measure the performance in terms of throughput and not IOPS. The sc1 volume type provides a lower throughput limit compared to st1. It is recommended for large, linear, cold-data workloads. It's a good low-cost alternative to st1 if you require infrequent access to your data. sc1 volumes cannot be used as bootable root volume. An sc1 volume size can range from 500 GiB to 16 TiB. An sc1 volume can have a maximum of 250 IOPS per volume. If you use multiple sc1 volumes in an instance, AWS imposes a limit of a maximum of 80,000 IOPS per instance.

The following list tells you where to use sc1 volumes:

- In throughput-oriented storage
- For large volumes of data when you don't need to access it frequently
- In application requirements where there is a need to lower the storage cost

Encrypted EBS

Amazon provides a simple EBS encryption solution that does not require you to build, maintain, and secure your own key management infrastructure.

After creating an encrypted EBS volume, when you attach it to a supported instance, it encrypts the following types of data:

- All data at rest, stored inside the volume.
- All data that is moving between the volume and the EC2 instance.
- All snapshot backups taken from the volume.

- AWS encrypts the data on the servers that host EC2 instances and provide encryption of data-in-transit from EC2 instances and on to EBS storage.
- Amazon EBS encrypts the data using **AWS Key Management Service (KMS)** with a customer master key whenever you create an encrypted volume and, subsequently, any snapshots from them.

When an encryption-enabled EBS volume is attached to the supported EC2 instance type, encryption takes place at EC2 for data-in-transit from EC2 to EBS storage. All future snapshot and disk I/O are encrypted. An encryption master key from Amazon KMS is used to perform encryption and decryption. Two types of encryption master keys can be used – an **Amazon created key and custom key**, or a **customer-provided key**. When creating an encrypted EBS volume for the first time in any AWS region, AWS automatically creates a master key in that region. By default, this key can only be used to encrypt the EBS volume. In order to use a custom key to encrypt EBS volumes, you need to create a **Customer Master Key (CMK)** in AWS KMS. Creating a CMK gives more control over disabling and defining access control, and creating and rotating encryption keys. At present, AWS does not provide a way to encrypt the root volume attached to the AWS EC2 instance, although there are some third-party tools, such as BitLocker, that you can use to encrypt the root volume. Other than the root, all attached EBS volumes can be encrypted. AWS uses **Advanced Encryption Standard (AES)**–256 algorithms for encryption.

Monitoring EBS volumes with CloudWatch

Once the desired size and type of EBS volumes have been created, it is recommended that you monitor the performance of the volumes. Monitoring helps in identifying a performance bottleneck, if any, due to a particular issue. We can use performance logs in CloudWatch to determine whether any volume type needs an upgrade in terms of size, IOPS, or throughput. When an EBS volume is created, AWS automatically creates several CloudWatch metrics for each EBS volume. Monitoring data is categorized into basic and detailed monitoring. Basic monitoring details are free and include metrics such as read bandwidth (KiB/s), write bandwidth (KiB/s), read throughput (Ops/s), and write throughput (Ops/s).

Only Provisioned IOPS SSDs (io1) send monitoring data to CloudWatch at one-minute intervals. The remainder of the EBS volume types, such as General Purpose SSD (gp2), Throughput Optimized HDD (st1), Cold HDD (sc1), and Magnetic (standard), send data to the CloudWatch metrics at five-minute intervals.

CloudWatch does not monitor at what rate the disk is being filled, or at what percent the disk is utilized or empty. For such requirements, you need to create a custom CloudWatch matrix.

More details about monitoring EC2 instances and EBS volumes are given in Chapter 7, *Monitoring with CloudWatch*.

Snapshots

An EBS snapshot is an AWS service that provides a mechanism to back up EBS volumes. AWS provides a way to back up your EBS data on S3 by taking a point-in-time snapshot. Snapshots are incremental in nature. That means it only saves data blocks that have changed since the last snapshot backup taken from the volume. This incremental approach of backing up data saves the time and cost of storage. If there are multiple snapshots for an EBS volume, and you delete one of the snapshots, AWS deletes only the data relevant to that snapshot. Other snapshots created out of the same volume refer to the base data and the incremental change relevant to them.

However, AWS stores the snapshot on S3; snapshots are not directly visible to users on S3. AWS stores the snapshot on a separate area in S3, which is inaccessible to end users. Users can see their snapshots on a snapshot dashboard, given within the EC2 dashboard.

Whether EBS volumes are attached to any instance or not, an EBS snapshot can be taken. A snapshot not only provides an option to perform a point-in-time backup of EBS volumes, but it also acts as a baseline for new EBS volumes. A snapshot can be used to migrate existing EBS volumes along with its data from one AZ, region, or AWS account to another. A snapshot of an encrypted volume is also encrypted.

The internal mechanism for a snapshot is to write the copy of data from the EBS volume to the S3 where it is stored redundantly across multiple AZs in the same region. When we access S3 using a web console, CLI, or API, we can't see the snapshots in S3.

It is critical for an organization to draft a backup and retention policy that determines how frequently snapshots are taken for EBS volumes. It is advisable that the backup and retention policy also defines a retention period for each snapshot depending on organizational needs. Housekeeping activities should be automated using scripts or tools to take the snapshots, as well as delete unwanted snapshots from the account, so as to control unnecessary costs.

Snapshots are the incremental backup. For any EBS volume, when taking the first snapshot, all the written blocks are copied to S3 and, finally, the **Table of Contents (TOC)** for the snapshot is written to the S3. The TOC points to these blocks. When taking a consequent snapshot for the same EBS volume, it only copies modified blocks from the EBS volume to S3 and creates a relevant table of contents. The TOC points to the recent blocks copied from EBS to S3, as well as blocks copied during previous snapshots, which are not changed. The following diagram helps us to see this. In the same diagram, we can see that TOC 2 and TOC 3 are pointing to some of the new and some of the old blocks:

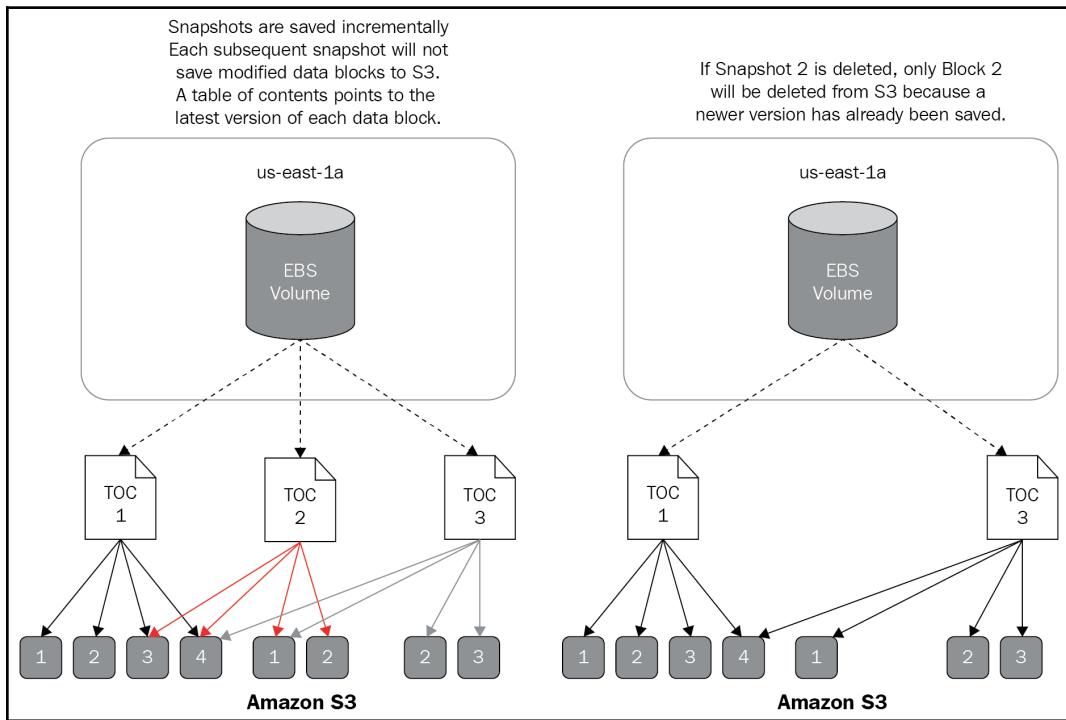


Figure 5.19: Snapshot creation and deletion

It also helps us to understand that, when any intermediate snapshot is deleted, only those blocks that are not referred to by any other snapshot TOCs are deleted.

EBS-optimized EC2 instances

In a normal EC2 instance, the usual network traffic and EBS traffic flows through the same network interface. If network traffic on application processes increases, it adversely affects the EBS performance. Similarly, activities on an EBS volume read and write can adversely affect other network activities on the instance. The following diagram indicates how network traffic from an EC2 instance and EBS volume flows through the same network link:

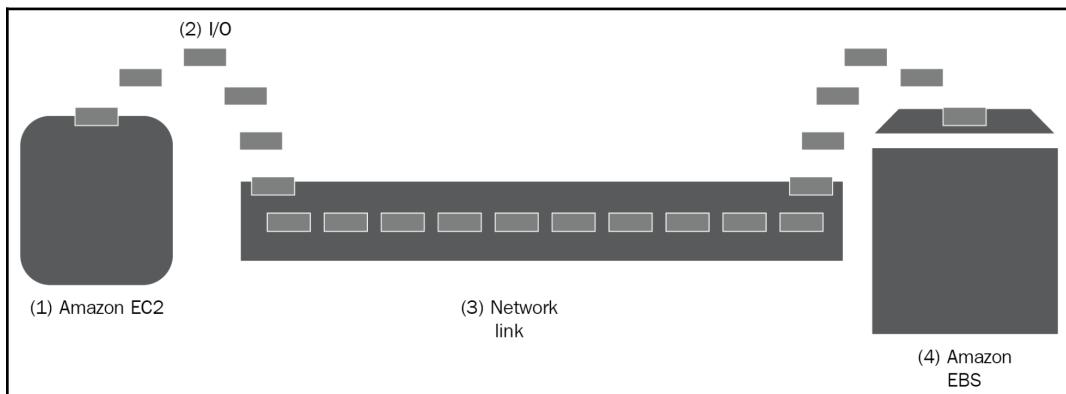


Figure 5.20: Network traffic on an EC2 instance and EBS volume

To handle such performance issues, AWS provides EBS-optimized instance types. An EBS-optimized instance provides dedicated throughput between the EBS volumes and the EC2 instance. Such instance types are essential for the application, where predictable and consistent disk performance is required. While using Provisioned IOPS SSD volumes, it is recommended that you use an EBS-optimized instance. This ensures the best performance out of Provisioned IOPS SSD volumes.

EC2 best practices

The following list summarizes EC2 best practices:

- Ensure that unused EC2 instances are stopped and, if not required, terminate them. This reduces unnecessary costs in the monthly AWS billing.
- Closely observe snapshots and AMIs, regularly perform housekeeping, and discard all the AMIs and snapshots that are not required. It is recommended that you automate and monitor this process.

- It is equally important to perform a periodical AMI and snapshots restore to make sure that the recovery process is performing within the stipulated time.
- Ensure that an instance type is set as per the requirements of the application hosted on the instance. It is recommended that you optimize the instance type according to application performance.
- To match a seasonal spike in compute requirement, plan for autoscaling and load balancing.
- Divide your application load into multiple instances rather than going for one big instance, where possible. Dividing the application workload over multiple instances, in different AZs, can avoid a single point of failure.
- Ensure that you use on-demand, spot, and reserved instances in the environment based on requirements. Balancing the instance types can significantly reduce costs, since reserved and spot instances provide a huge cost benefit.
- Always keep your key pairs safely. Once a key pair is lost, it cannot be recovered from AWS.
- Do not embed access and secret keys in the EC2 instance. Where possible, use EC2 roles for accessing AWS resources from an EC2 instance.
- Attach appropriate IAM roles and policies, at the time of creating an EC2 instance, to grant access to other AWS services.
- Periodically update security groups and maintain the least permissive rules. Periodically update and patch the operating system to overcome possible security and other vulnerabilities.
- According to the data storage requirement, such as persistent or temporary, select EBS or instance type-backed AMI to provision an instance.
- It is recommended that you use separate volumes for the operating system and data storage.
- Always tag AWS resources with appropriate and relevant tags. This provides a convenient way to identify the correct EC2 instance at the time of performing maintenance.
- Create golden AMIs and update them periodically.
- Periodically perform maintenance to delete obsolete and unwanted AMIs as well as snapshots to minimize monthly AWS billing.

- Monitor EC2 and EBS volumes to identify any performance bottleneck in the environment.
- By default, public IPs attached to an EC2 instance change with every stopping and starting of the instance. Considering this, it is recommended that you design and deploy an application on the instance that can handle dynamic IP addresses.

Any Elastic IP attached to an instance remains intact, even if the instance is restarted or stopped. Elastic IP can be released from one instance and associated with another instance.

Summary

- Amazon provides scalable computing capacity using EC2.
- No need to buy hardware—EC2 provides a pay-as-you-go model.
- EC2 instances are quick to provision and access.
- You can scale the number of instances dynamically up or down based on requirements.
- You can use AMIs to launch EC2 instances.
- AMI includes preconfigured operating systems and software for ease of use.
- AWS provides a number of instance types based on various combinations of CPU, memory, storage, and networking capacity.
- AWS uses key pairs, public keys and private keys, that are used to secure access to EC2 instances.
- A public key is stored in an EC2 instance, and a private key is provided to you that can be used to access the instances remotely.
- Instance store volume data is automatically deleted when you stop or terminate an instance.
- AWS EBS offers a persistent storage option, wherein data remains intact even if you stop or restart your instance.
- You can use AWS Region and AZs and launch your instances in multiple locations. Such an approach provides you with high availability on your application/workload.
- You have to associate a security group with your instances that acts as a firewall and restricts the protocols, ports, and source IP ranges that your instances can serve.

- An EC2 instance is assigned a private IP, as well as a public or Elastic IP.
- You can assign metadata to EC2 instances using tags.
- You can choose VPC and subnet for your instance while launching it.
- You can access EC2 instances using AWS console and CLI tools.
- There are different pricing models for EC2 instances, in other words, on-demand, reserved, scheduled-reserved, and spot pricing.
- Dedicated hosts can be used when an organization's regulatory requirement stipulates that the hardware resources used for running specific workloads should be isolated.
- An AMI can have either of two root device types:
 - Amazon EBS-backed AMI (which uses permanent block storage to store data)
 - Instance store-backed AMI (which uses ephemeral block storage to store data)
- EC2 supports two virtualization types—PV and HVM.
- You can use the SSH client to access Linux-based EC2 instances.
- You can use the RDP client to access Windows-based EC2 instances.
- AWS provides gp2 and io1 as SSD.
- AWS provides st1 and sc1 as HDD.
- AWS still supports previous generation Magnetic volumes.
- You can encrypt EBS volumes using KMS while launching the EC2 instances.
- AWS does not provide a mechanism to encrypt root volumes. You can use third-party encryption tools to encrypt root volumes if required.
- EBS snapshots are implicitly stored in S3, but are not directly visible to end users.
- Snapshots are an incremental backup of the volume.
- An EBS-optimized instance provides dedicated throughput between the EBS volumes and the EC2 instance.

6

Handling Application Traffic with ELB

Elastic Load Balancer (ELB) is a load balancing service that distributes incoming application traffic across multiple EC2 instances and increases fault tolerance of an application. This chapter describes how to create ELB, how it works, and what the critical aspects of the ELB service are.

The following topics will be covered in this chapter:

- Introducing ELB
- Types of ELB
- Features of ELB
- Working with ELB
- ELB best practices

Introducing ELB

ELB is an AWS service that automatically distributes incoming network or application traffic to a number of EC2 instances. It monitors the health of each of the EC2 instances associated with it and forwards traffic only to healthy instances. ELB provides a single point of contact for the EC2 instances behind ELB. Each of the EC2 instances are marked with a status, either `InService` if it is healthy, or `OutOfService` if it is unhealthy. Traffic is routed only to `InService` instances. ELB provides a single point of contact for the application traffic that is hosted on multiple EC2 instances. By routing traffic only to healthy instances, ELB provides fault tolerance to the application and ensures the high availability of the application. The following diagram shows how multiple EC2 instances are hosted behind ELB:

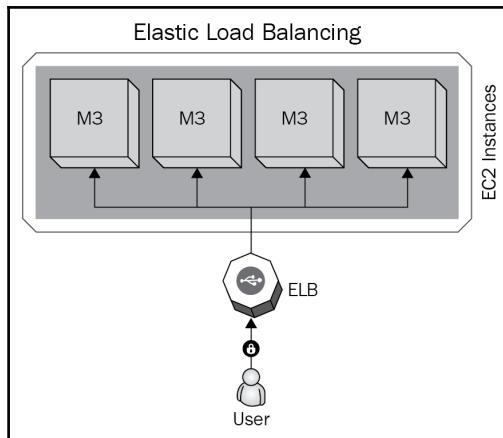


Figure 6.1: ELB

Benefits of using ELB

ELB provides high availability, fault tolerance, elasticity, and security to your environment. We will now look at the benefits of ELB in brief:

- **High availability:** An application hosted behind an ELB on a fleet of EC2 instances spread across multiple **Availability Zones (AZ)** provides high availability. Consider a scenario where an application is hosted in an EC2 instance without using ELB. If traffic to the application spikes, the EC2 instance may not be able to handle the traffic and application performance, as well as availability being affected. Consider the similar traffic scenario where an application is hosted across multiple EC2 instances behind ELB. ELB distributes the traffic in a round-robin fashion to all the instances, ensuring that no EC2 instance is flooded with traffic.
- **Fault tolerance:** ELB monitors the health of each of the instances associated with it. If an instance is unreachable, it marks the instance as `OutOfService`. Similarly, if an instance is reachable and healthy, it marks the instance as `InService`. The traffic is directed only to the `InService` instances. This way, even if an instance is down, the application traffic is not affected and it provides fault tolerance for the application.
- **Elasticity:** In spite of high availability and fault tolerance with a limited number of instances in ELB, traffic can spike beyond the capacity of the instances in ELB. If traffic is very low, however, the number of instances in ELB may be underutilized. Overutilization of instances may lead to application performance issues, while underutilization results in unnecessary costs. To handle both of these scenarios, ELB can be associated with an **Auto Scaling group**, which provides elasticity to ELB by automatically increasing the number of instances in ELB when the traffic is high, and automatically reducing the number instances when the traffic is low.
- **Security:** ELB, when used with AWS **Virtual Private Cloud (VPC)**, provides robust networking and security features. It provides the ability to create an internal-facing ELB, using private IP addresses to route traffic within the **Virtual Private Network (VPN)**. You can also create an internet-facing load balancer to route traffic from the internet to instances in a private subnet. In either case, instances are not directly accessible to the traffic. ELB acts as a frontend to the instances associated with it, and provides a security layer to protect them.

Types of ELB

There are three types of ELB – **Classic Load Balancer**, **Application Load Balancer**, and **Network Load Balancer**.

Classic Load Balancer

Classic Load Balancer is one of the initial offerings of AWS and handles the traffic depending upon application or network-level information. It is used for load balancing simple traffic across multiple EC2 instances where there is a need for a highly available, automatically scaled, and secured environment. It is not suitable for the applications that require advanced routing capabilities for handling the application traffic.

Creating a Classic Load Balancer

Before creating a load balancer, it is necessary to ensure that the desired configuration of VPC and EC2 instances are in place. If you create an ELB with EC2 instances spread across multiple AZs, ensure that the required instances are in place in each AZ to perform tests after creating the ELB. You should also verify that the security group attached to the respective EC2 instances allows incoming traffic on required ports and protocols. It is also recommended that the desired application or web server is configured properly on the target EC2 instances. With this background, let's follow these steps to create a Classic Load Balancer:

1. Log in to the AWS web console with sufficient credentials to create an ELB.
2. Go to the EC2 dashboard and select **Load Balancers** from the left-hand side pane, as shown in the following screenshot:

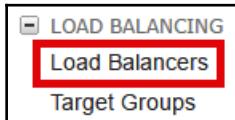


Figure 6.2: Load balancers option on EC2 dashboard

3. Click on the **Create Load Balancer** button, as shown in the following screenshot:

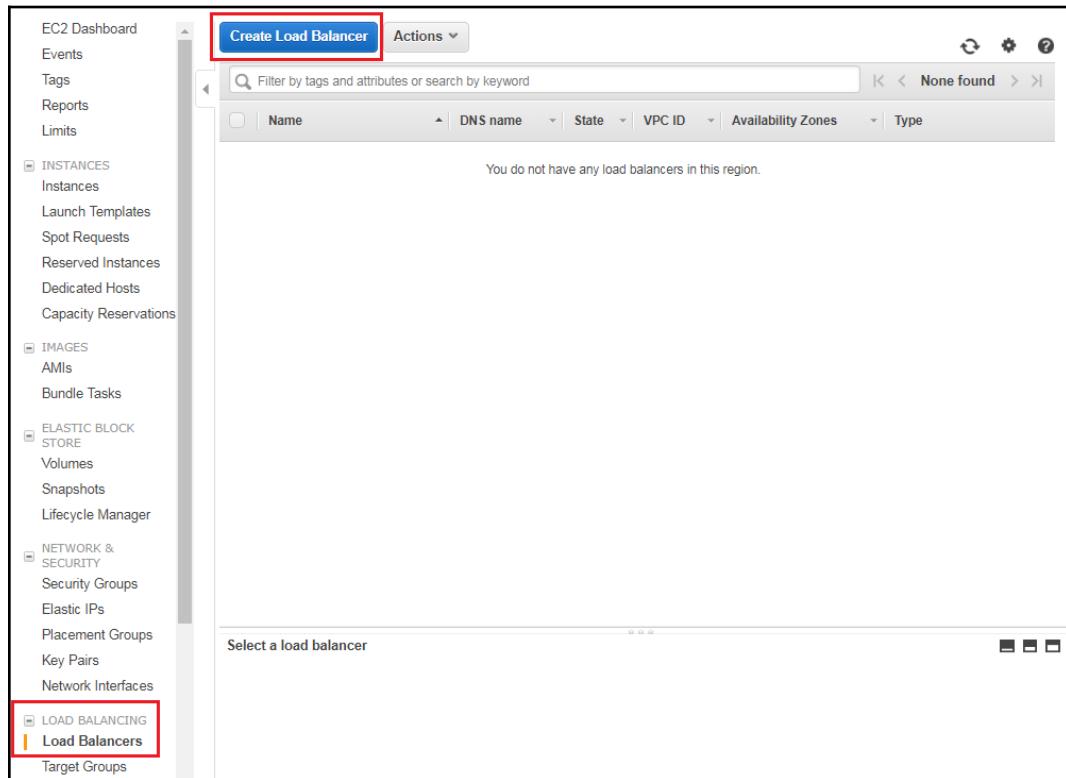


Figure 6.3: Creating a load balancer button

4. Select the **Classic Load Balancer** type, as shown in the following screenshot:

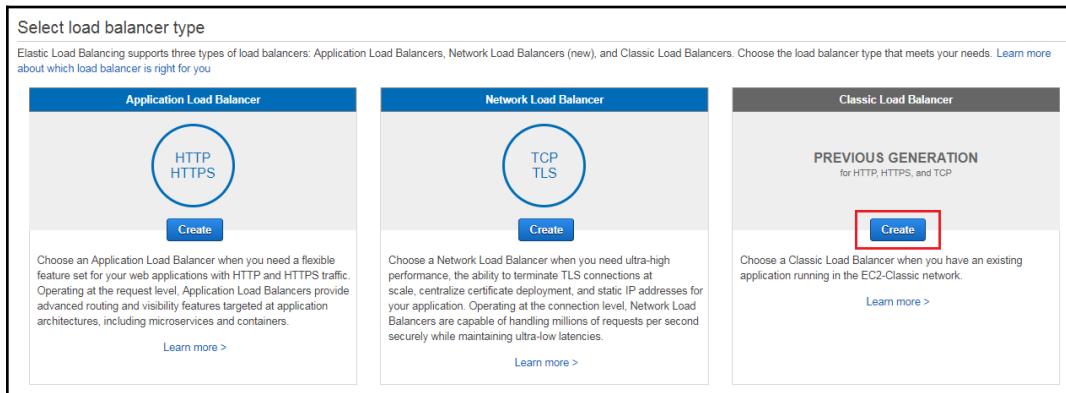


Figure 6.4: Selecting Classic Load Balancer

5. Next, we will define the Classic Load Balancer basic configuration:

This wizard will walk you through setting up a new load balancer. Begin by giving your new load balancer a unique name so that you can identify it from other load balancers you might create. You will also need to configure ports and protocols for your load balancer. Traffic from your clients can be routed from any load balancer port to any port on your EC2 instances. By default, we've configured your load balancer with a standard web server on port 80.

Load Balancer name: Only a-z, A-Z, 0-9 and hyphens are allowed
Create LB Inside: My Default VPC (172.31.0.0/16)

Create an internal load balancer: (what's this?)
Enable advanced VPC configuration:

Listener Configuration:

Load Balancer Protocol	Load Balancer Port	Instance Protocol	Instance Port
HTTP	80	HTTP	80

Add **Cancel** **Next: Assign Security Groups**

Figure 6.5: Classic Load Balancer – configuration

For this step, you need to understand the following ELB options:

- **Load Balancer name:** Provide a meaningful and relevant ELB name; it can be alphanumeric (A-Z, a-z, 0-9) and can contain a dash (-).

- **Create LB Inside:** Select the desired VPC where the EC2 instances reside. Only one VPC can be selected from the drop-down menu.
 - **Create an internal load balancer:** Only select this option when creating a load balancer to manage traffic from within an AWS environment. An internal load balancer cannot serve internet traffic and can be accessed from within the network or associated VPC.
 - **Enable advanced VPC configuration:** Select this option to perform a manual selection of available subnets within a selected VPC in the region. When you select this option, other relevant options become visible.
 - **Listener Configuration:** A listener defines the ports and protocols that ELB opens for end users to send requests and it defines target ports and protocols on the EC2 instances where ELB can forward the incoming traffic. When the listener port on ELB is selected as HTTPS, in the resulting consecutive steps, it asks you to upload an **Secure Sockets Layer (SSL)** certificate along with essential cipher configuration.
6. **Create a new security group or Select an existing security group**, as shown in the following screenshot. It is recommended that you create individual security groups for every ELB. Security groups should open only the minimum required ports and protocols:

Step 2: Assign Security Groups

You have selected the option of having your Elastic Load Balancer inside of a VPC, which allows you to assign security groups to your load balancer. Please select the security groups to assign to this load balancer. This can be changed at any time.

Assign a security group:

Create a new security group
 Select an existing security group

Security group name: quick-create-1

Description: quick-create-1 created on Thursday, May 16, 2019 at 1:07:59 PM UTC+

Type	Protocol	Port Range	Source
Custom TCP	TCP	80	Custom 0.0.0.0/0

Add Rule

Cancel Previous Next: Configure Security Settings

Figure 6.6: Assign security groups

7. Configure the SSL certificate. You can either **Choose an existing certificate from AWS Certificate Manager (ACM)**, **Choose an existing certificate from AWS Identity and Access Management (IAM)**, or **Upload a new SSL certificate to AWS Identity and Access Management (IAM)**, as shown in the following screenshot:

Step 3: Configure Security Settings

Upload a certificate to IAM

Certificate name: e.g. myServerCert

Private Key: (pem encoded)

Certificate body: (pem encoded)

Certificate chain: Optional (pem encoded)

Select a Cipher

Configure SSL negotiation settings for the HTTPS/SSL listeners of your load balancer. You may select one of the Security Policies listed below, or customize your own settings. [Learn more](#) about the Security Policies and configuring SSL negotiation settings.

Predefined Security Policy: ELBSecurityPolicy-2016-08

Custom Security Policy

SSL Protocols

- Protocol-TLSv1
- Protocol-SSLv3
- Protocol-TLSv1.1
- Protocol-TLSv1.2

SSL Options

- Server Order Preference

Cancel Previous Next: Configure Health Check

Figure 6.7: Configure Security Settings

When a listener is selected to listen on a HTTPS (443) port, it is essential to provide the certificate details in order to move on to the next step.

8. The next step is to **Configure Health Check**. You can use the health check options table as a reference to configure the health check options, as follows:

Step 4: Configure Health Check

Your load balancer will automatically perform health checks on your EC2 instances and only route traffic to instances that pass the health check. If an instance fails the health check, it is automatically removed from the load balancer. Customize the health check to meet your specific needs.

Ping Protocol: HTTP

Ping Port: 80

Ping Path: /index.html

Advanced Details:

- Response Timeout:** 5 seconds
- Interval:** 30 seconds
- Unhealthy threshold:** 2
- Healthy threshold:** 10

Buttons: Cancel, Previous, Next: Add EC2 Instances

Figure 6.8: The Configure Health Check dialog box

The various health check options are listed as follows:

Health check option	Description
Ping Protocol	This can be either the HTTP , HTTPS , TCP , or SSL target instance, which should allow the selected protocol for pinging it.
Ping Port	This is the port on which ELB can send the ping request to instances. This port should be the port on which EC2 hosts the application. The default port is 80, which can be changed based on where the application listens.
Ping Path	This points to a specific document or page on the EC2 instance. Ideally, this should be the path to the initial page loaded on the application. The ping is considered to be a success if it is able to reach the page on a given path. To minimize the time to complete each ping, it is recommended that you point it to the document root (/) rather than /index.html.
Healthy threshold	This is the threshold value in the range of 2 to 10. It defines the number of consecutive successful health checks before considering any EC2 instance as a healthy instance.
Unhealthy threshold	This is the threshold value in the range of 2 to 10. It defines the number of consecutive failed health checks before considering any EC2 instance as an unhealthy instance.
Response Timeout	This is an integer value in the range of 2 to 60 seconds. It defines the time interval in seconds. If an instance fails to respond during this time, ELB considers it a failed health check. The instance is marked as an unhealthy instance if it crosses the unhealthy threshold. The health check timeout must be smaller than Interval .
Interval	This can be given as an integer value in the range of 5 to 300 seconds. ELB waits for the number of seconds defined in the interval between two health checks.

9. Add EC2 instances from each of the AZs selected. Select **Enable Cross-Zone Load Balancing** and **Enable Connection Draining** as required. If connection draining is enabled, you can specify the number of seconds against it. ELB waits for the number of seconds defined for in-flight traffic to drain before forcing a session to close on deregistering an EC2 instance:

Step 5: Add EC2 Instances

The table below lists all your running EC2 Instances. Check the boxes in the Select column to add those instances to this load balancer.

VPC vpc-cfc50dab (172.31.0.0/16)

<input type="checkbox"/> Instance	Name	State	Security groups	Zone	Subnet ID	Subnet CIDR
No instances available.						

Availability Zone Distribution

Enable Cross-Zone Load Balancing (i)

Enable Connection Draining (i) 300 seconds

Cancel **Previous** **Next: Add Tags**

Figure 6.9: Add EC2 instances to an ELB

10. Now, add tags as shown in the following screenshot. It is recommended that you give meaningful and relevant tags on ELB as per any enterprise naming conventions:

Step 6: Add Tags

Apply tags to your resources to help organize and identify them.

A tag consists of a case-sensitive key-value pair. For example, you could define a tag with key = Name and value = Webserver. [Learn more](#) about tagging your Amazon EC2 resources.

Key	Value
Name	Learn-ELB

Create Tag

Cancel **Previous** **Review and Create**

Figure 6.10: Adding tags to the ELB

11. Finally, click on **Review and Create** and verify the configuration. If everything is as per the requirements, click on the **Create** button. Once the ELB creation is completed, it provides an ELB endpoint. An ELB endpoint can be configured in the CNAME record set in the DNS to forward end user requests to ELB.

Application Load Balancer

Application Load Balancer has recently been introduced in the AWS offerings. Unlike a Classic Load Balancer, it provides advanced application-level routing options. It provides us with the ability to route the traffic based on application content spread across multiple services, microservices, or container services with multi-tiered application architecture.

Network Load Balancer

Network Load Balancer has also been introduced to the AWS offerings recently. It handles TCP traffic at fourth layer of the OSI model. With the capacity to handle millions of requests per second, it can still maintain ultra-low latencies. It also automatically provides load balancing for the containerized application and provides better fault tolerance for applications hosted on EC2, as well as in containers.

Features of ELB

There are several features of ELB. Each type of load balancer, be it a Classic Load Balancer, an Application Load Balancer, or a Network Load Balancer, has its own set of features. The following table illustrates these features and compares the three types of ELB. This comparison can help you in choosing the correct load balancer type depending on your requirements:

Features	Application Load Balancer	Network Load Balancer	Classic Load Balancer
Protocols	HTTP, HTTPS	TCP	TCP, SSL/TLS, HTTP, HTTPS
Platforms	VPC	VPC	EC2-Classic, VPC
Health checks	✓	✓	✓
CloudWatch metrics	✓	✓	✓
Access logs	✓	✓	✓
Zonal fail-over	✓	✓	✓
Connection draining (deregistration delay)	✓	✓	✓
Route to multiple ports on a single instance	✓	✓	
Web sockets	✓	✓	

IP addresses as targets	✓	✓	
Load balancer deletion protection	✓	✓	
Path-based routing	✓		
Host-based routing	✓		
Native HTTP/2	✓		
Configurable idle connection timeout	✓		✓
Cross-zone load balancing	✓	✓	✓
SSL offloading	✓		✓
Server Name Indication (SNI)	✓		
Sticky sessions	✓		✓
Backend server encryption	✓		✓
Static IP		✓	
Elastic IP address		✓	
Preserve source IP address		✓	
Resource-based IAM permissions	✓	✓	✓
Tag-based IAM permissions	✓	✓	
Slow start	✓		
User authentication	✓		
Redirects (*)	✓		
Fixed response (*)	✓		

ELB feature comparison

Let's explore each of the features supported by their respective load balancer type:

- **Protocols:** As depicted in the table, a Classic Load Balancer supports transport layers as well as application layers. Transport layers consist of the TCP and SSL protocols, while application layers use HTTP and HTTPS protocols. An Application Load Balancer takes routing decisions at the application layer and supports only HTTP and HTTPS protocols. On the other hand, a Network Load Balancer takes routing decisions at the network layer and supports only TCP.
- **Platforms:** A Classic Load Balancer, being one of the initial offerings, supports EC2-Classic as well as EC2-VPC. Application Load Balancer and Network Load Balancer we introduced later in the AWS offerings for advance application- and network-level network flow routing options. Both the lateral entrant load balancers support EC2 instances hosted in a VPC only.

- **Health checks:** This is used to determine whether an instance is capable of handling traffic or not. ELB periodically sends pings, tries to establish a connection, or sends HTTP/HTTPS requests. These requests are used to determine the health status of an instance and are called health checks. All the healthy instances in ELB that serve the traffic have a status of `InService` instances. All the unhealthy instances that cannot serve the traffic are called `OutOfService` instances. The ELB routes only send requests to the healthy instances. ELB stops routing traffic requests to the `OutOfService` instances and resumes sending traffic to the instances as soon as the instance status becomes healthy and `InService`.
- **CloudWatch metrics:** AWS sends data points to CloudWatch for load balancers and all the instances associated with it. CloudWatch aggregates those data points and creates statistics in an ordered set of time series data. This time series data is called **CloudWatch metrics** for ELB. With CloudWatch metrics, you can verify the number of healthy EC2 instances in a load balancer during a specific time period. This helps to verify whether the system is consistently performing as expected or not. With CloudWatch, you can create an event trigger in case the metric is outside of the acceptable range. Such event triggers can send mail to stakeholders or take any specific action based on its association with either a lambda function or Auto Scaling group.
- **Access logs:** ELB generates access logs for each request sent to it. With each passing request, it captures information such as the time of request, the source IP address, latencies, the request path, and the server response. The access log can be used to analyze the traffic and to troubleshoot any issue. Enabling an access log is optional, and it is disabled by default. Once the access log is enabled for ELB, it captures the log and stores it in an Amazon S3 bucket. The S3 bucket name can be specified while enabling the access log on ELB. AWS does not charge any additional amount for access logs; however, you are charged for the storage you use on an S3 bucket for access logs.
- **High availability (zone fail-over):** ELBs are highly available. The incoming traffic from clients is distributed across the targets within the same AZ. At the same time, the health of the registered targets is monitored by the load balancer. The load balancer also ensures that the traffic is routed to healthy targets in service. The traffic is stopped and rerouted to other healthy targets in the same AZ when the load balancer comes across an unhealthy target. You will have set up targets in another AZ in case all the targets in a particular AZ are unhealthy. The load balancer will automatically fail-over to route traffic to your healthy targets in the other AZs.

- **Connection draining:** ELB stops sending requests to instances that are either unhealthy or are deregistering from ELB. This can lead to the abrupt closure of an ongoing session initiated by a user. Such abruptly closed sessions give an unpleasant experience to a user. To handle such situations, AWS supports connection draining in ELB. When connection draining is enabled, and if an instance becomes unhealthy or is being deregistered, ELB stops sending new requests to such instances; however, it also completes any in-flight requests made to such instances. The timeout value for connection draining can be specified between 1 second and 3,600 seconds. The default timeout value for connection draining is 300 seconds. The load balancer forces a connection to close and deregisters it if the time limit is reached.
- **Route to multiple ports on a single instance:** An Application Load Balancer supports routing traffic to multiple ports on an EC2 instance. An EC2 instance can run multiple applications on different ports on a single EC2 instance:

It can run the main web server on port 80
It can run the admin application on port 8080
It can run the reporting application on port 5000

In such a scenario, an Application Load Balancer can route all the traffic requests with a host header as `www.example.com` to port 80 on the instance, all traffic requests with a `www.example.com/admin` host header to port 8080 on the instance, and all the traffic requests with a `www.example.com/reporting` host header to port 5000 on the instance. Routing to multiple ports on a single instance is only supported in an Application Load Balancer and Network Load Balancer.

- **WebSockets:** WebSockets are an advanced technological development that enables the application to open an interactive communication session between the browser and an application server. If WebSockets are enabled, it allows you to send a message to an application server and receive event-driven responses from the server without polling the server for its response. It provides a persistent connection between a browser and the application server. The browser establishes the connection with the application server using a process called **WebSocket handshaking**. Once a session is established, the browser or the application can start sending the data unilaterally as and when required. The Application Load Balancer and Network Load Balancer support WebSockets, while a Classic Load Balancer does not support it.

- **IP addresses as targets:** A Network Load Balancer supports target types either as an **instance** or an **IP**. IP address routing can be used to build a hybrid architecture where a single load balancer accepts incoming traffic from clients and distributes traffic across running services on cloud VPC and the data center. This makes it possible to host multiple web sites on the same EC2 instance with individual IP addresses, but also uses common port number to support IP-based virtual hosting. It also makes it possible to host multiple instances of a service, such as containers, on the same instance while using multiple **Elastic Network Interface (ENI)** and security groups to implement fine-grained access control.



Information on using an Application Load Balancer through IP address to AWS and on-premises can be found at <https://aws.amazon.com/blogs/aws/new-application-load-balancing-via-ip-address-to-aws-on-premises-resources/>.

- **Load balancer deletion protection:** Application Load Balancers and Network Load Balancers support a configuration option that ensures that an ELB is not accidentally deleted by anybody. If ELB deletion protection is enabled, you cannot delete an ELB unless this option is disabled again.
- **Path-based routing:** An Application Load Balancer provides a mechanism to route traffic to a specific target group based on the URL path specified in the host header of the request. For example, requests to `www.example.com/production` can be sent to target group A; requests to `www.example.com/sandbox` can be sent to target group B; and requests to `www.example.com/admin` can be sent to target group C. Path-based routing is only supported in an Application Load Balancer.
- **Host-based routing:** Host-based routing refers to a mechanism of routing traffic to a specific target group based on the hostname specified in the host header of the request. For example, requests to `www.example.com` can be sent to target group A; requests to `mobile.example.com` can be sent to target group B; and requests to `api.example.com` can be sent to target group C. Host-based routing is only supported in an Application Load Balancer.

- **Native HTTP/2 support:** HTTP/2, also called HTTP/2.0, is a major revision of the HTTP network protocol used on the internet. Using HTTP/2 features, web applications can increase their speed. It also improves the way data is framed and transported. Websites can increase the efficiency and minimize the number of requests required to load an entire web page. An Application Load Balancer supports HTTP/2, the industry-standard protocol, and provides better visibility on the health of the EC2 instances and microservers or containers.
- **Configurable idle connection timeout:** Every time a user makes a request to ELB, it maintains two connections. One connection is created with the client and another connection is created with the target EC2 instance. For each of these connections, ELB maintains an idle timeout period. If there is no activity during the specified idle time between the client and the target EC2 instance, ELB closes this connection. In short, if there is no traffic flowing from client to EC2 or vice versa, ELB closes the connection. By default, idle time duration is 60 seconds in ELB. It is supported by the Application Load Balancer and Classic Load Balancer.
- **Cross-zone load balancing:** When a Classic Load Balancer is created with instances spread across multiple AZs, it distributes the traffic evenly between the associated AZs. If you have an ELB with 10 instances in **US-East-1a** and 2 instances in **US-East-1b**, the traffic is distributed evenly between 2 AZs. This means that 2 instances in US-East-1b serve the same amount of traffic as 10 instances in **US-East-1a**. This is the default behavior of Classic Load Balancing. If you enable cross-zone load balancing, ELB distributes traffic evenly between all the EC2 instances across multiple AZs. Cross-zone load balancing is configurable in a Classic Load Balancer and a Network Load Balancer; however, it is always enabled in an Application Load Balancer.
- **SSL offloading:** The Application Load Balancer and Classic Load Balancer supports HTTPS encryption and decryption. This enables traffic encryption between ELB and client-initiated HTTPS connections. It is also known as SSL termination or offloading. This helps to get rid of operational overhead and administrative complexity. Now, rather than deploying an X.509 certificate on each of the EC2 instances, it just needs to upload a certificate to ELB.

- **SNI:** Application Load Balancer supports SNI. This makes it possible to host multiple applications, each with its own SSL certificate behind one load balancer. This also makes it possible to deploy a multi-tenant **Software-as-a-Service (SaaS)** application behind one load balancer. It also supports smart certificate selection algorithms with SNI and helps the load balancer to select precise certificate matches with client requests based on multiple factors, including the capabilities of the client.
- **Sticky sessions:** Sticky sessions are a way to consistently route traffic requests from a particular user to the same target instance based on an HTTP session, IP address, or a cookie. A Classic Load Balancer supports application cookies, or, if an application does not have a cookie, you can create a session cookie by specifying stickiness duration in an ELB configuration. It creates a cookie named AWS ELB to map the session to an instance. An Application Load Balancer supports sticky sessions using load balancer-generated cookies. This is a mechanism to route traffic requests originated from a user to the same target group every time during a session. When a user sends a request to an application for the first time, ELB routes the request to one of the instances and generates a cookie. This cookie is included in the response back to the user. When the user sends the subsequent requests, the request contains the same cookie. Based on the cookie, this request is sent to the same target instance for as long as the session lasts. The name of the cookie generated in an Application Load Balancer is **AWSALB**.
- **Back-end server encryption:** During SSL offloading, the client's HTTPS requests terminate on ELB, and ELB communicates over HTTP to the target EC2 instances. In general, network traffic between source and target AWS resources can't be listened by third-party resources in the same AWS account. However, that may be not enough to fulfill compliance to standards such as PCI and HIPAA, so it may be essential to have end-to-end encrypted communication. Both Application Load Balancers and Classic Load Balancers support end-to-end encrypted communication with self-signed certificates. It enables backend authentication and ensures that the backend EC2 instance provides a public key when ELB is communicating with them over HTTPS/SSL.

- **Static IP:** The Application Load Balancer or Classic Load Balancer keep rotating underneath the IP address. It doesn't create any issues for the end user to be able to access an application using a URL, as it may resolve to the global DNS, based on CNAME. However, it may be difficult for the system or security administrator to whitelist a specific IP address for the firewall. On the other hand, the Network Load Balancer supports a static IP address for each AZ. So, it becomes easy to whitelist specific IP address in firewall. It is important to remember that the Network Load Balancer works on layer four and only supports TCP connection, so it doesn't support HTTPS connection off-loading like the Application Load Balancer does.
- **Elastic IP address:** The Network Load Balancer provides a static IP address for each AZ. Rather than having some random static IP, it also optionally gives a provision to attach an Elastic IP. In case of any situation when you are required to re-construct the Network Load Balancer, it will come up with some random static IP. As a result, clients who have whitelisted an earlier static IP may need to change the IP. But, when attached to the Elastic IP, you will have legacy to re-attach it, and, as a result, the customer isn't required to make any changes in the firewall.
- **Preserve source IP Address:** When an end user request hits an ELB, the ELB changes the source IP and other request headers and forwards it to one of the EC2 instances where the application is hosted. This is the default behavior of an Application Load Balancer and Classic Load Balancer, which bar an application from obtaining original client connection information. In some enterprise applications, it is required to have original source connection details to perform traffic analysis. This information helps the application to understand more about the end user's behavior. However, this ELB does not provide the original connection information to the application by default; it supports a proxy protocol, which can be used to obtain connection information from ELB. The proxy protocol is an IP that carries client connection information. You can enable or disable the proxy protocol on these ELBs with the help of AWS CLI. On the other hand, the Network Load Balancer, by default, preserves the client source IP address and provides to the backend servers.



If you specify targets using an instance ID, the source IP addresses of the clients are preserved and provided to your applications as seen at <https://docs.aws.amazon.com/elasticloadbalancing/latest/network/load-balancer-target-groups.html>.

- **Resource-based IAM permissions:** The Application Load Balancer, Network Load Balancer, and Classic Load Balancer support the control of resource-level permissions using an **Amazon Resource Name (ARN)**. It can be based on the load balancer name, ID, listener ID, or target group ID.
- **Tag-based IAM permissions:** All three load balancer types also support controlling privileges in IAM policies based on resource tags. This gives extra flexibility to manage privileges based on an organizational or compliance-based tags.
- **Slow start:** By default, as soon as any EC2 instance is registered to the target group and identified as healthy and `InService` it starts receiving its full share of requests. Slow start can be configured for a target group from 30 seconds to 15 minutes. Once it is enabled, even EC2 instances identified as `InService` will wait for the slow start time period to elapse. Once it has passed, the load balancer linearly increases the number of requests to the target. It is very helpful for the resources (such as previous generation EBS volumes) or applications that require a warm-up period to respond to requests at full performance.
- **User authentication:** Previously, to securely authenticate a user with an Application Load Balancer, developers were required to write a code to offload the authentication responsibility from the backend to the Application Load Balancer. But now, the Application Load Balancer supports federated user identification or web user identifications without writing the same authentication code again and again.
- **Redirects and fixed response:** The Application Load Balancer allows us to offload request redirection or fixed response for content-based routing. Redirection makes possible for ALB to redirect from one URL request to another URL, for example HTTP to HTTPS. It could achieve a better SEO to get a high score for SSL/TLS for compliance. On the other hand, with a fixed response, you can choose which requests should be served by ALB and which requests should give a fixed response. A fixed response makes it possible to handle HTTP error code.

How ELB works

Since all the ELB terminologies and features are explained, let's look at how exactly ELB works.

The working of the Classic Load Balancer

As per the OSI model, a Classic Load Balancer runs at layer four, which is the transport layer. Layer four operates at the network protocol level. It does not check the content of actual network packets. A transport layer-level load balancer does not check the specific details of HTTP and HTTPS requests. In other words, it distributes the load without necessarily knowing much detail about the incoming traffic requests. A user creates the Classic Load Balancer with instances spread across one or more AZs.

The load balancer configuration includes the following:

- The security group, which defines the security of ELB including the source of traffic that is allowed on the target group, ports, and protocols open on ELB for incoming traffic.
- Listener ports and protocols on which ELB listens for incoming traffic and target ports and protocols on the EC2 instances where ELB directs the traffic. The user includes SSL/TLS certification in the configuration.
- The user defines the health check for the associated instances. The health check enables ELB to monitor the instances and determines whether an EC2 instance is `InService` or `OutOfService`. ELB routes traffic only to healthy instances with an `InService` status.
- The user can enable cross-zone load balancing to balance the traffic across all the instances in multiple AZs as required. If cross-zone load balancing is not enabled, traffic is routed on round-robin fashion between AZs.
- The user can enable connection draining. If connection draining is enabled before deregistering an instance from ELB, it allows in-transit sessions to complete for a defined span of seconds.
- Once ELB is configured and associated with an application, AWS provides an ELB endpoint. An ELB endpoint can be used to define a CNAME in Route 53 or any other DNS.
- As described in the following diagram, a user can type in a site URL, `www.example.com`, in the browser.
- The user's request for the website hits a DNS server, which resolves to an ELB endpoint registered with the DNS, and the request is forwarded to the ELB endpoint.

- The ELB endpoint resolves in a public IP address if ELB is defined as an internet-facing ELB, or it resolves in a private IP address if ELB is defined as an internal-facing ELB:

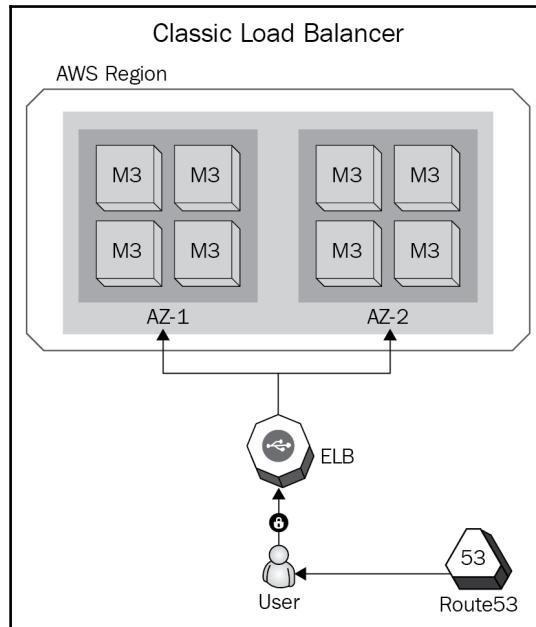


Figure 6.11: Classic Load Balancer

ELB receives the request forwarded by the user and checks the source IP, host header, and cookie, to see whether sticky sessions is enabled. If the source of the request is authorized to access ELB, it forwards the traffic in a round-robin fashion, to the AZs associated with ELB. If cross-zone load balancing is enabled, the traffic is distributed between all the instances in a round-robin fashion instead of distributing it at AZ level. If sticky sessions is enabled, ELB establishes a persistent session between the client browser and the EC2 instance. All subsequent requests from this user are forwarded to the same EC2 instance until the user session ends.

The target EC2 instance forwards the response, which is captured by ELB and forwarded back to the user.

The working of the Application Load Balancer

As per the OSI model, an Application Load Balancer runs at layer seven, called the application layer. An Application Load Balancer is more powerful than a Classic Load Balancer. It checks the traffic packets for more detail. It also has access to the HTTP and HTTPS headers of the requests and fetches more detail from the packets, which empowers it to do a more intelligent job in distributing the load to target instances.

An Application Load Balancer supports content-based routing. You can enable host-based and path-based routing on an Application Load Balancer. It also supports routing to multiple ports on a single instance. It provides support for HTTP/2, which enables a website to increase efficiency and minimize the number of requests required to load a web page.

With the support of WebSockets, an Application Load Balancer can enable the application to open an interactive communication session between the browser and an application server. A user creates an Application Load Balancer with instances spread across one or more target groups in multiple AZs.

The load balancer configuration includes the following:

- Security group, which defines the security of ELB, including the source of traffic that is allowed on the target group, ports, and protocols open on ELB for incoming traffic.
- Listener ports and protocols on which ELB listens for incoming traffic and target ports and protocols on EC2 instances where ELB directs the traffic. The user can also define content-based routing. Content-based routing includes host-based routing or path-based routing. Depending upon the configuration, the traffic requests are forwarded to specific target groups.
- The user includes SSL/TLS certification in the configuration.
- The user defines the health check for the associated instances. The health check enables ELB to monitor the instances and determines whether an EC2 instance is `InService` or `OutOfService`. The ELB routes traffic only to healthy instances with an `InService` status.
- Cross-zone load balancing is automatically enabled in the Application Load Balancer and the traffic is routed on a round-robin fashion across all the instances.
- The user can enable connection draining. If connection draining is enabled before deregistering an instance from the ELB, it allows an in-transit session to complete for a defined span of seconds.

- Once an ELB is configured and associated with an application, AWS provides an ELB endpoint. The ELB endpoint can be used to define a CNAME in Route 53 or any other DNS.
- A user can type in a site URL, `www.example.com`, in the browser.
- The user request for the website hits a DNS server, which resolves to an ELB endpoint registered with the DNS and the request is forwarded to the ELB endpoint.
- The ELB endpoint resolves in a public IP address if ELB is defined as an internet-facing ELB, or it resolves in a private IP address if ELB is defined as an internal-facing ELB.

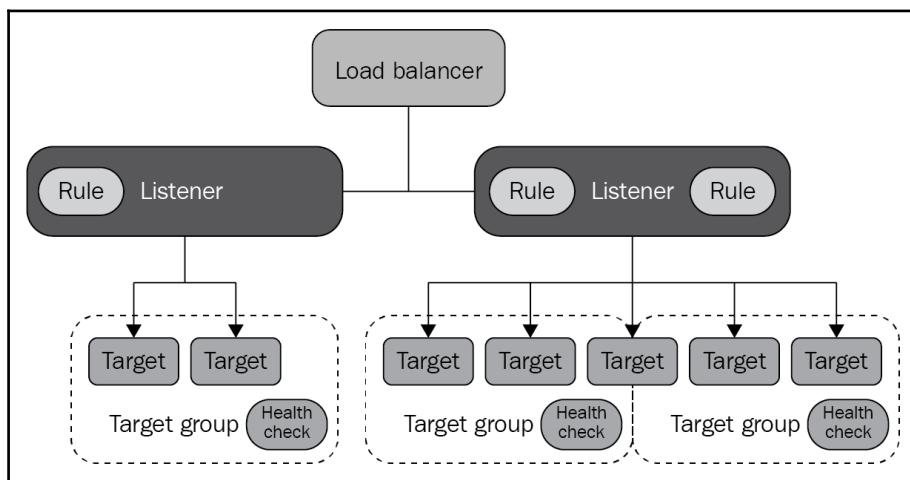


Figure 6.12: Application Load Balancer

ELB receives the request forwarded by the user, and then checks the source IP, host header, and cookie, to see whether sticky sessions is enabled.

If the source of the request is authorized to access ELB, it forwards the traffic in a round-robin fashion to the target group associated with the ELB. The traffic is distributed between all the instances in round-robin fashion. If sticky sessions is enabled, the ELB establishes a persistent session between the client browser and the EC2 instance. All subsequent requests from this user are forwarded to the same target until a session lasts.

The target EC2 instance forwards the response, which is captured by ELB and forwarded back to the user.

ELB best practices

ELB best practices are as follows:

- While defining a load balancer, it is recommended you identify target AZs and target groups.
- Use multiple AZs in ELB, as it provides high availability and fault tolerance.
- It is highly recommended that a security group for ELB opens only the required ports and protocols.
- Always configure health checks for ELB on appropriate ports and protocols. If ELB is created for a web server, use HTTP/HTTPS in health checks, instead of TCP.
- Do not create internet-facing ELBs for internal needs.
- Use SSL security certificates to encrypt and decrypt HTTPS connections where possible.
- If a heavy traffic spike is expected on a given schedule, contact AWS support and ask them to pre-warm ELB.
- Use ELB deletion protection to prevent accidental deletion.
- Use cross-zone load balancing in a Classic Load Balancer for evenly distributing the load across all EC2 instances in associated AZs.
- Carefully enable connection draining on ELBs associated with critical user applications.

Summary

- ELB is an AWS service that automatically distributes incoming network or application traffic to a number of EC2 instances.
- It monitors the health of each of the EC2 instances associated with it and forwards traffic only to healthy instances.
- ELB provides a single point of contact for the EC2 instances behind ELB.
- Each of the EC2 instances is marked with a status: either `InService` if it is healthy, or `OutOfService` if it is unhealthy.
- ELB only routes traffic to `InService` instances.

- ELB helps to provide high availability, scalability, elasticity, fault tolerance, and security to your application workloads running on EC2 instances.
- There are three types of load balancers supported by AWS ELB—Classic Load Balancer, Network Load Balancer, and Application Load Balancer.
- ELB stops sending requests to instances that are either unhealthy or are deregistering from ELB when connection draining is enabled.
- You can prevent ELB from being accidentally deleted by enabling deletion protection option.
- When a Classic Load Balancer is created with instances spread across multiple AZs, it's called cross-zone load balancing.
- The Application Load Balancer and Classic Load Balancer support HTTPS encryption and decryption.
- Sticky sessions are a way to consistently route traffic requests from a particular user to the same target instance based on an HTTP session, IP address, or cookie.
- The Network Load Balancer supports a static IP addresses for each AZ.

7

Monitoring with CloudWatch

CloudWatch is an AWS service that can be used for monitoring various infrastructure and application resources running on your AWS cloud. CloudWatch can be used to collect a number of metrics from the AWS resources. This allows you to track these metrics and initiate actions based on the thresholds you set. CloudWatch can also collect log files, generate metrics out of them, and help to monitor log files. You can set alarms on specific events and trigger an action whenever an event occurs. For example, if CPU utilization for a specific instance crosses a threshold of 80%, you can initiate an action to spin up a new instance.

The following topics will be covered in this chapter:

- Introducing CloudWatch
- How Amazon CloudWatch works
- Elements of CloudWatch
- Creating a CloudWatch alarm
- Billing alerts
- CloudWatch dashboards
- Monitoring types – normal and detailed
- CloudWatch best practices

Introducing CloudWatch

CloudWatch supports the monitoring of many AWS services, such as EC2 instances, DynamoDB, and **Relational Database Service (RDS)**. You can also generate custom metrics and log files using your own applications and associate them with CloudWatch. Amazon services, such as Auto Scaling, use CloudWatch alarms to automatically scale an environment up or down based on the traffic in an environment.

CloudWatch provides a number of graphs and statistics. It gives your system broad insights into how resources are utilized and how to monitor application performance, and tracks the overall operational health of respective applications in an environment. All this infrastructure and application telemetry data can be used to ensure the smooth functioning of your environment.

How Amazon CloudWatch works

CloudWatch acts as a repository of metrics by collating raw data from various AWS services or applications and converting it into metrics, statistics, and graphs, and facilitates certain actions based on specific data points in metrics. The following diagram shows the high-level architecture of CloudWatch:

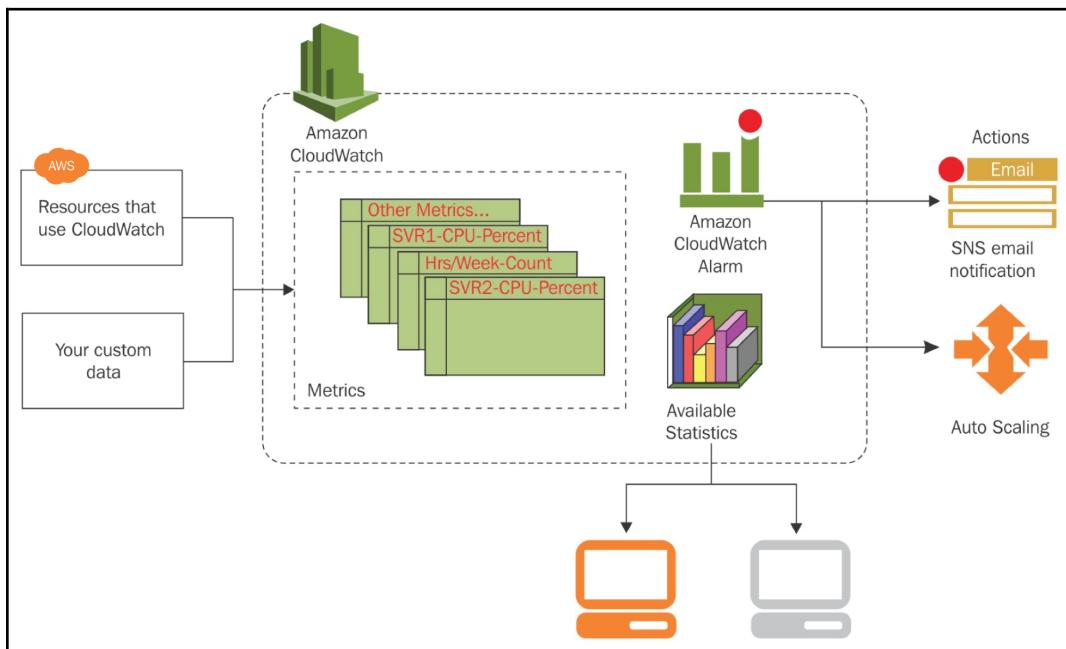


Figure 7.1: High-level architecture of CloudWatch

As shown in the preceding diagram, various AWS services and your custom metrics data are stored in CloudWatch. CloudWatch generates various statistical and graphical visualizations out of these metrics, which can be consumed directly from the AWS Management Console or by various other means, including, but not limited to, AWS CLI, API, and custom-built applications. CloudWatch enables the user to set alarms that can trigger certain actions based on the metrics threshold or events. It can send email notifications or automatically scale an environment up or down using Auto Scaling groups associated with the alarms.

Elements of Amazon CloudWatch

To understand and work with AWS-generated and custom metrics, it is important to understand a few basic concepts and terminologies used with Amazon CloudWatch, which are as follows:

- Namespaces
- Metrics
- Dimensions
- Statistics
- Percentile
- Alarms

Namespaces

CloudWatch namespaces are containers in which metrics for different applications are stored. It is a mechanism to isolate the metrics of different applications from each other. Namespaces ensure that an application's metrics, as well as respective statistical data, are not accidentally mixed up with any other application's metrics. All the AWS services that use CloudWatch to register their metrics use a unique namespace. A namespace name begins with AWS/ and is generally followed by the application name. If you create a custom application and need to store metrics in CloudWatch, you must specify a unique namespace as a container to store custom metrics. The namespace can be defined at the time of creating the metrics. The namespace name can be a string of up to 256 characters, including alphanumeric (A-Z, a-z, 0-9), hyphens (-), underscores (_), periods (.), slashes (/), hashes (#), and colons (:).



For more details on namespaces, you can refer to <http://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/aws-namespaces.html>.

Metrics

Metrics are sets of data collected over a period of time with a specific time interval for quantitative assessment, measurement, and comparison of performance data that is generated by a specific application or a service. For example, CPU utilization data for an EC2 instance is stored in a relevant CloudWatch metric at a time interval of 1 minute. Each AWS service stores several metrics in CloudWatch. For an instance, EC2 stores metrics such as `CPUUtilization`, `NetworkIn`, `NetworkOut`, `DiskReadOps`, and `DiskWriteOps`.

CloudWatch, by default, stores metrics data at an interval of 5 minutes. If you enable advanced monitoring, it can store a metrics data point at an interval of 1 minute. Since July 26, 2017, CloudWatch has enabled high-resolution custom metrics, which makes it possible to create custom metrics even with a one-second resolution.

Each data point in a metric must be marked with the timestamp. Timestamps can be two weeks in the past or up to two hours into the future. In any case, when a timestamp is not provided, CloudWatch automatically creates a timestamp based on the time when a data point is received. Timestamps are `DateTime` objects with a complete date, such as `2018-10-26 T22:39:36Z`. It is recommended that you store timestamps in **Coordinated Universal Time (UTC)**, as the retrieval of statistics from CloudWatch is in UTC.

CloudWatch retains all metrics for 15 months before discarding them; however, a lower-level granularity of data is deleted to keep the overall volume of data at a reasonable size:

Granularity of data	Data retention period
< 60 seconds (1 minute)	3 hours
60 seconds (1 minute)	15 days
300 seconds (5 minutes)	63 days
3600 seconds (1 hour)	455 days (15 months)

CloudWatch allows you to create custom metrics with a custom name and the time interval required to store a data point. CloudWatch metrics are referred to as either **standard-resolution metrics** or **high-resolution metrics**, depending on the granularity of the data it contains. Standard-resolution metrics store data with one-minute granularity. High-resolution metrics store data with one-second granularity.

Metrics are region-based. AWS-generated default metrics or custom metrics can be found only in the region where they are created. Metrics cannot be deleted manually. They automatically expire after 15 months from the time of the last published data point into the metrics. Metrics are unique in nature, defined by the combination of a namespace, metric name, and one or more dimensions. Each data point has a timestamp, a data value, and, optionally, a unit of measurement.

Dimensions

A **dimension** in a CloudWatch metric is a mechanism to uniquely identify metrics. It is a name/value pair that is associated with metrics.

For example, CloudWatch stores metrics for EC2 instances in a namespace called AWS/EC2. All EC2 metrics are stored in the specific namespace. If you need to retrieve metrics for a specific `instanceID` value, you search the metrics with its `instanceID` value, such as `i-09x7xxx4x0x688x43`.

In the preceding example, `instanceID` is its name and `i-09x7xxx4x0x688x43` is its value, which represents a dimension. When you search the metrics with the `instanceID` value, you are using a dimension. A dimension is a unique identifier of metrics. When you search with a different value of `instanceID`, CloudWatch provides you with a different metric that is related to another instance. Similarly, you can search metrics with a different dimension name. For example, you can search the EC2 metrics with an `imageID`. Metrics can have a maximum of 10 dimensions. Here's the list of dimensions in EC2:

- `instanceId`
- `imageId`
- `instanceType`
- `AutoScalingGroupName`

Just like EC2, other services have their own dimensions. You can also create a metric with custom dimensions.

Statistics

Statistics are collections of aggregated metrics data for a specific period of time. Metrics data is aggregated by using namespaces, metric names, dimensions, and several data points in a given time period. CloudWatch provides the following statistics on the metrics data:

Statistic	Description
Minimum	The least, smallest, or lowest value recorded in the metrics for a specific period of time. For example, the lowest CPU utilization recorded during the day on an EC2 instance.
Maximum	The largest, biggest, or highest value recorded in the metrics for a specific period of time. For example, the highest CPU utilization recorded during the day on an EC2 instance.
Sum	The total value resulting from the addition of all the matching metrics for a specific period of time. It is useful to study the total volume of activities. For example, the total data transferred on an EC2 instance in the past 24 hours.
Average	The average value resulting from the addition of all the matching metrics for a specific period of time and dividing it by the total number of sample count. For example, the average CPU utilization on an EC2 instance for the last hour.
SampleCount	This counts the number of data points used for the statistical calculation.
pNN.NN	The value represented in percentile. It uses up to two decimal places to specify a number. It helps to study statistics in terms of percentile, such as p80.43.

At any given point in time, a data point may contain more than one value, as shown in the following table. This describes some statistics with sample data:

Hour	Raw data	Sum	Minimum	Maximum	SampleCount
1	80, 85, 75, 70, 77	387	70	85	5
2	60, 80, 70, 75, 65	350	60	80	5

Percentile

A **percentile** helps in finding the comparative standing of a value in a set of data. Let's take an example of a dataset that contains the CPU utilization of an EC2 instance. The example data is arranged in ascending order for clarity:

12	17	25	55	58	61	63	70	83	97
----	----	----	----	----	----	----	----	----	----

Let's consider the percentile for 83 from the preceding dataset. In this example, 83 stands at the 90th percentile. That means 90% of the data is less than or equal to 83 and the remaining 10% is above 83.

A percentile gives a better insight into the data with a better understanding of how the data is distributed. The following AWS services support percentiles:

- EC2
- **Application Load Balancer (ALB) and Classic Load Balancer (CLB)**
- RDS
- Kinesis
- API Gateway

CloudWatch provides options to monitor systems and applications using various statistics, such as maximum, minimum, average, sum, or percentile. If you choose percentile, CloudWatch starts displaying the statistics according to percentile value.

Alarms

CloudWatch alarms help to define a threshold value that is constantly monitored, and an action is triggered when the threshold condition is breached. For example, you can define a threshold of 80% CPU utilization on an EC2 instance and trigger an action whenever the CPU utilization is $\geq 80\%$ for three consecutive periods.

You can set the action as one or more SNS notifications, Auto Scaling actions, or EC2 actions. An SNS notification can be used to send an alert via email and/or SMS, and it can trigger a Lambda function. Auto Scaling actions are used to scale an environment up by adding more instances or scale an environment down by reducing the number of instances. EC2 actions can be used to reboot, stop, terminate, or initiate a recovery on the instance.

An alarm can have three possible states:

- The alarm status displays **OK** when the metric is within the defined threshold.
- The alarm status displays **ALARM** when the metric is beyond the defined threshold.
- The alarm status displays **INSUFFICIENT_DATA** when the alarm has only just been configured, the metric is unavailable, or we do not have sufficient data for the metric to determine the alarm state.

Creating a CloudWatch alarm

The following steps describe the process of creating a CloudWatch alarm:

1. Open the CloudWatch console by navigating to <https://console.aws.amazon.com/cloudwatch/> on your browser. It brings you to the CloudWatch dashboard.
2. Click on **Alarms**:

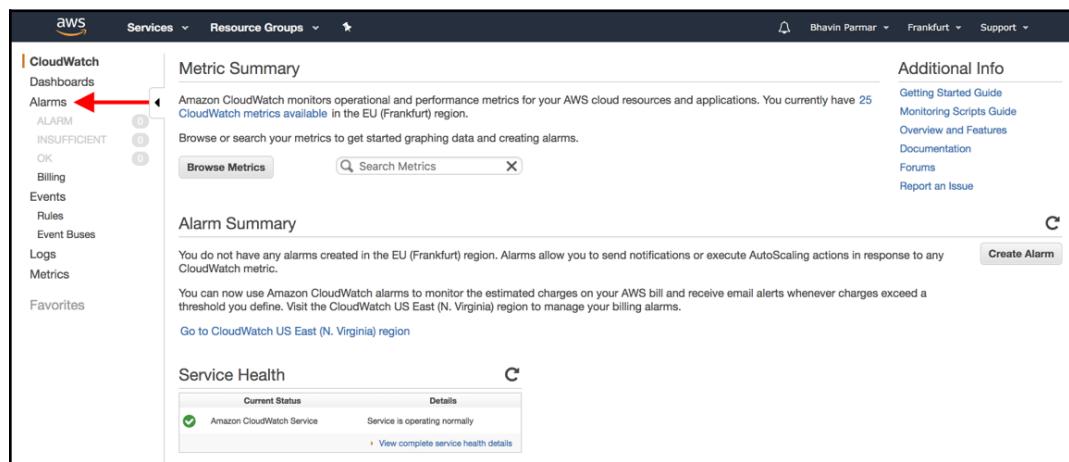


Figure 7.2: CloudWatch dashboard

3. Click on the **Create Alarm** button:

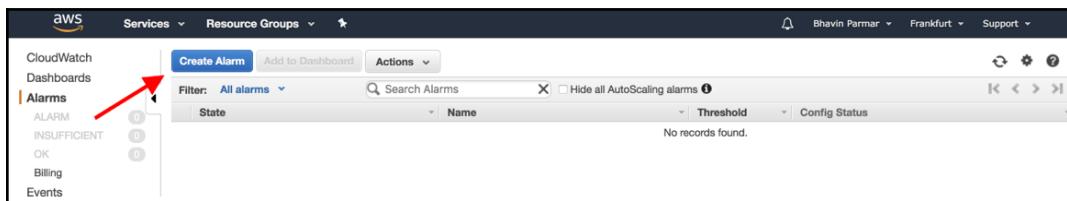


Figure 7.3: The Create Alarm button

4. Click on **Select metric**:

The screenshot shows the 'Create new alarm' wizard. The first section is 'Metric', with the sub-instruction 'Select a metric to alarm on.' and a large input field containing a placeholder 'Select metric'. A red arrow points to this input field. The next section is 'Alarm details', with fields for 'Name:' and 'Description:', both currently empty. Below that is a configuration section for 'Whenever:' with a condition 'is: >= 0' and a note 'for: 1 ⚡ out of 1 datapoints ⓘ'. The final section is 'Additional settings' with a dropdown for 'Treat missing data as:' set to 'missing'. Each section has a horizontal line separating it from the next.

Figure 7.4: Selecting a metric

5. This window shows multiple categories of metrics, depending upon the metrics you have in the account. For example, if you have EC2 instances in the account, it shows **EC2** metrics; if you have ELB resources in the account, it shows **ELB** metrics; and, similarly, it shows different categories of metrics for which you have resources in the account. In the following screenshot, we can see metrics categories for **EBS** and **EC2**. Depending upon the requirement, you can select any of the category metrics as needed; for our example, we will select **EC2**:

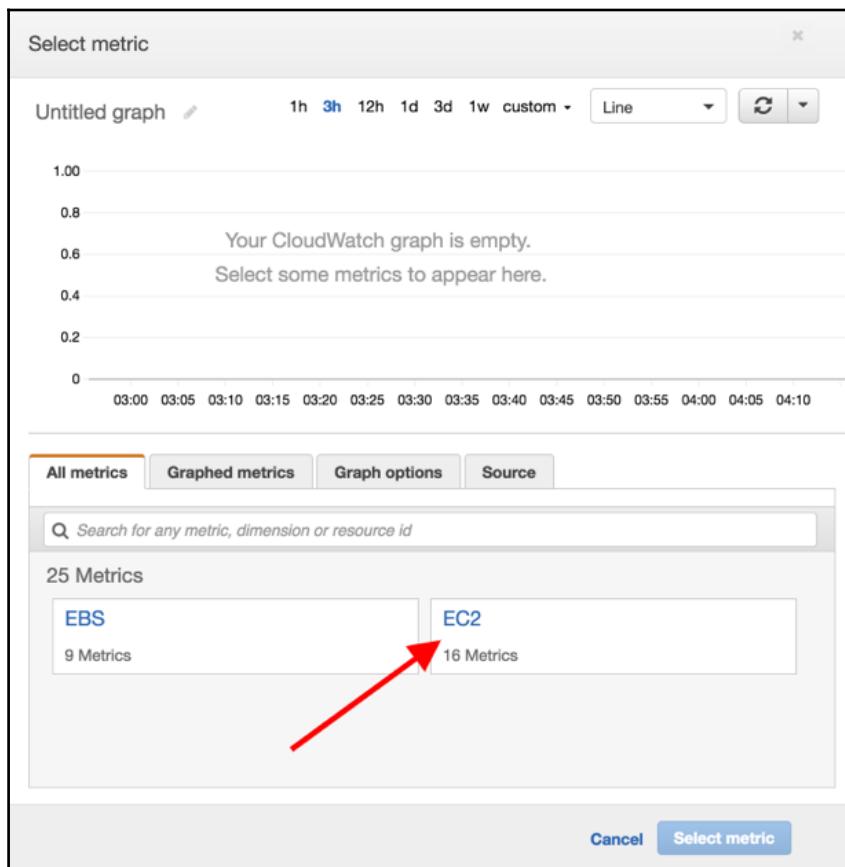


Figure 7.5: CloudWatch Metrics by category

6. Select Per-Instance Metrics:

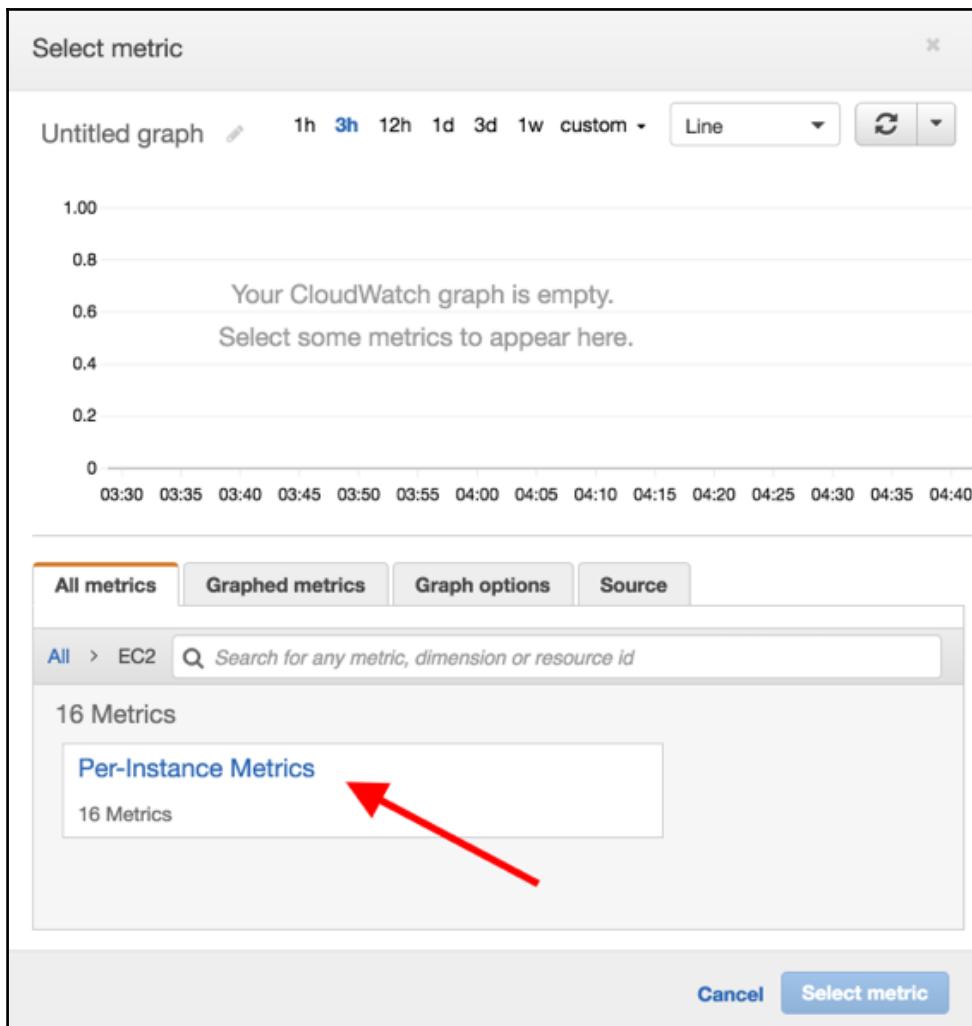


Figure 7.6: Per-instance metrics

7. There are a number of metrics, such as **CPUUtilization**, **NetworkIn**, and **NetworkOut**. You can select any of these metrics as needed. For this example, select a metric against an EC2 instance for **StatusCheckFailed_Instance**, and click on the **Select metric** button:

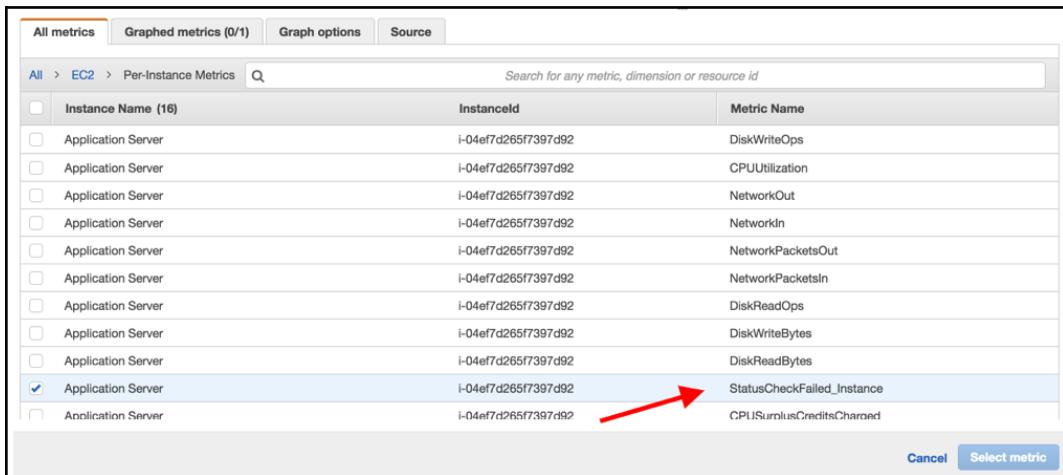


Figure 7.7: Select a metric

8. In the subsequent window, provide the alarm **Name**, **Description**, and threshold values:

Create new alarm

Metric [Edit](#)

This alarm will trigger when the blue line goes up to or above the red line for 1 datapoints within 5 minutes

1.00

0.8

0.6

0.4

0.2

0

<div style="position: absolute; top: 210px; left: 5875px; width: 10px; height: 20px; background-color: #E67E2

9. In the **Actions** section, you can set one or more actions from an SNS notification, an Auto Scaling action, or an EC2 action. An SNS notification can be used to send an alert over email or an SMS over mobile. An Auto Scaling action is used to scale an environment up by adding more instances or scale an environment down by reducing the number of instances. An EC2 action can be used to reboot, stop, terminate, or initiate a recovery on the instance. For this example, in the same window, select the **+ EC2 Action** button:

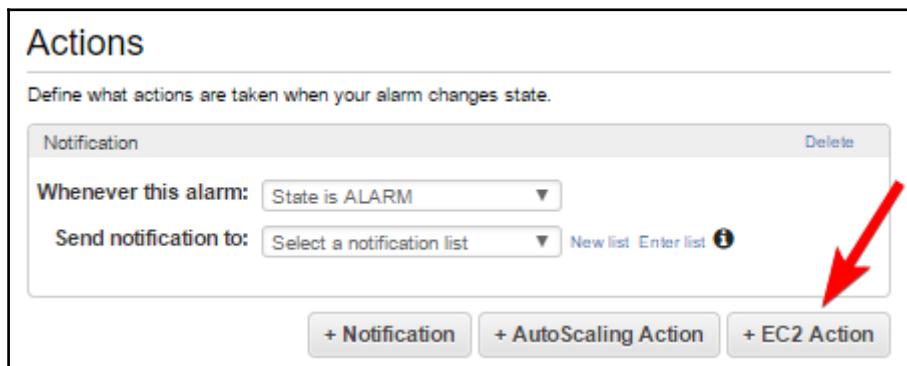


Figure 7.9: Actions

10. When you click **+ EC2 Action**, it displays the following screen. From the given screen, select **State is ALARM** against the **Whenever this alarm** line item and select **Recover this instance** from the action list:

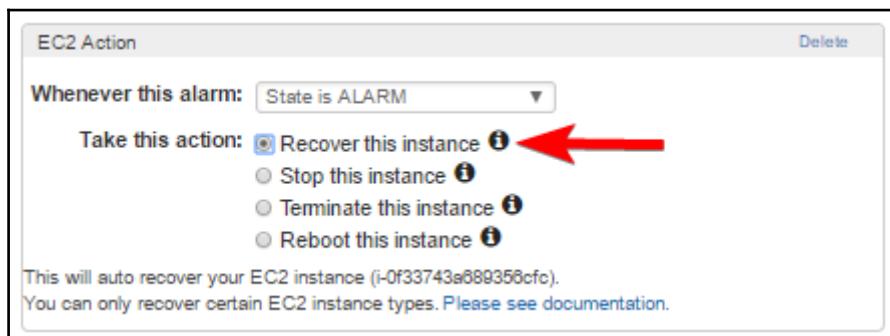


Figure 7.10: EC2 Action



The **Recover this instance** option is supported only with the C3, C4, C5, M3, M4, M5, R3, R4, R5, T2, T3, X1, and X1e instance types. Also, make sure the instance-store EC2 instances are not configured with this option, as instance-store data gets lost on restart.

11. If the instance is configured for basic monitoring, you have to select a minimum of five minutes. An instance can be configured for basic monitoring or advanced monitoring. With basic monitoring, the instance is monitored at an interval of five minutes. With advanced monitoring, the instance is monitored at an interval of one minute, based on the consecutive alarm period selected in the **Alarm Threshold** section and the interval period selected for checking the alarm.
12. You can click on the **Create Alarm** button after selecting all the required options in the **Create Alarm** window; this creates the alarm. Once the alarm is created, it gets triggered based on the alarm status. In this example, alarm actions are triggered when the instance status is failed for three consecutive periods of five minutes each. An instance status failure alarm occurs when there is a hardware issue or any issue with the virtualization host where the EC2 instance is hosted on AWS. If you set this alarm, AWS automatically tries to recover the instance.

You can follow similar steps to create different types of alarms based on a metric of your choice. For example, if you choose the CPU utilization metrics, you can either send an alert to an administrator when the CPU utilization breaches the threshold, or you can perform Auto Scaling actions. Auto Scaling actions can add or remove instances in an Auto Scaling group to optimize the resources required for serving the traffic.

Billing alerts

Just as CloudWatch monitors other AWS resources, it can also monitor the monthly billing charges for an AWS account. You can set the threshold for billing. As soon as the billing amount reaches the threshold or shoots above the specified threshold, it notifies the specified administrators. You need to enable billing alerts before you can configure alerts on billing data. You can enable billing alerts from the AWS account settings. Remember, only the root user can enable the billing alerts; AWS IAM users cannot do that.

The process of enabling billing alerts is as follows:

1. Log into the AWS account with the root user credentials.
2. Click on the account name and **My Billing Dashboard**:

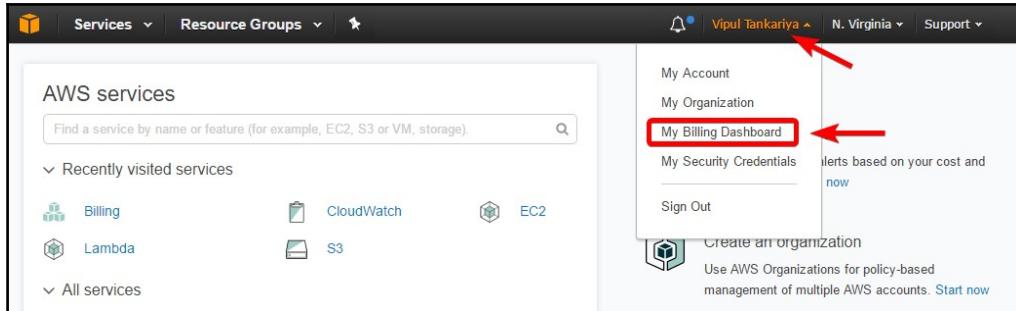


Figure 7.11: Opening the My Billing dashboard

3. From the left-hand pane, click on **Preferences** and check **Receive Billing Alerts**. Optionally, you can enable **Receive PDF invoice By Email** and **Receive Billing Reports**:

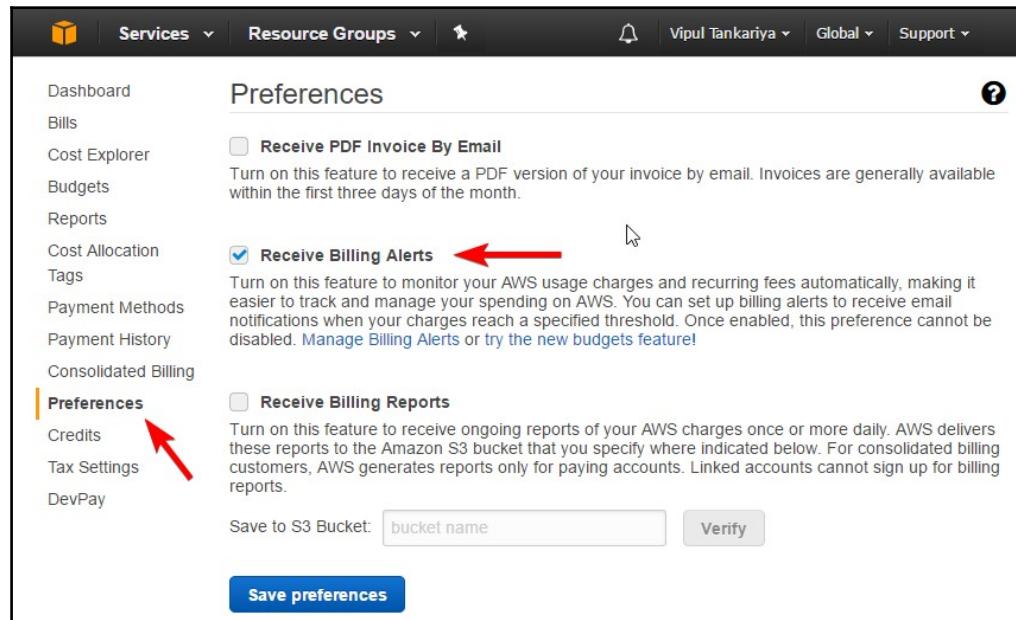


Figure 7.12: Billing dashboard preferences



All billing CloudWatch alerts can be configured only from the **N. Virginia** region, as all related metrics are created there by default. Once billing alerts has been enabled from **Preferences**, before creating billing alerts, make sure that you have switched to the **N. Virginia** region.

CloudWatch dashboards

Amazon CloudWatch provides a customizable dashboard inside a web console. It can display a set of critical metrics together. You can create multiple dashboards, where each dashboard can focus on providing a distinct view of your environment. You can also create a custom dashboard to view and monitor the selected AWS resources from the same or different regions. It provides a way to get a single view of critical resource metrics and alarms for observing the performance and health of the environment. It gives you the freedom to add, remove, move, resize, and rename the graphs, as well as change the refresh interval of selected graphs in a dashboard. The following figure shows a screenshot of the default dashboard view:

The screenshot shows the CloudWatch Metrics service dashboard. On the left, a sidebar menu includes options like CloudWatch Metrics, Dashboards (which is selected and highlighted in orange), Alarms, ALARM (0), INSUFFICIENT (0), OK (0), Billing, Events, Rules, Event Buses, Logs, Insights, Metrics, Favorites, and a link to Add a dashboard. The main content area is titled "Dashboards" and contains a "Create dashboard" button. Below it is a table listing four existing dashboards:

Name	Favorite	Last updated (UTC)
Packt-CRM-Dashboard	☆	2019-05-14 06:26
Packt-DR-Infrastructure-Dashboard	☆	2019-05-14 06:27
Packt-Management-Dashboard	☆	2019-05-14 06:27
Packt-Subscription-Application-Dashboard	☆	2019-05-14 06:28

To the right of the dashboard list is a "Additional Information" sidebar with links to Getting Started Guide, Documentation, Forums, and Report an Issue. At the bottom of the page, there are links for Feedback, English (US) language selection, and standard footer links for Privacy Policy and Terms of Use.

Figure 7.13: CloudWatch Dashboard

Monitoring types – basic and detailed

Amazon CloudWatch monitoring can be broadly categorized into two categories, basic monitoring and detailed monitoring, which are described here:

- **Basic monitoring:** Basic monitoring is free and collects data at 5-minute intervals. By default, when you provision AWS resources, all AWS resources except ELB and RDS start with a basic monitoring mode. ELB and RDS monitor the resources at 1-minute intervals. For other resources, you can switch the monitoring mode to detailed monitoring.
- **Detailed monitoring:** Detailed monitoring is chargeable and makes data available at 1-minute intervals. Currently, AWS charges \$0.015 per hour, per instance. Detailed monitoring does not change the monitoring on ELB and RDS, which, by default, collects data at 1-minute intervals. Similarly, detailed monitoring does not change the EBS volumes, which are monitored at 5-minute intervals.

You can enable detailed monitoring while launching an instance or after provisioning the instances. While launching an EC2 instance, it provides an option to enable detailed monitoring in the third step, as shown in the following screenshot. Refer to Chapter 5, *Getting Started with Elastic Compute Cloud (Ec2)*, which provides more details on how to launch an EC2 instance:

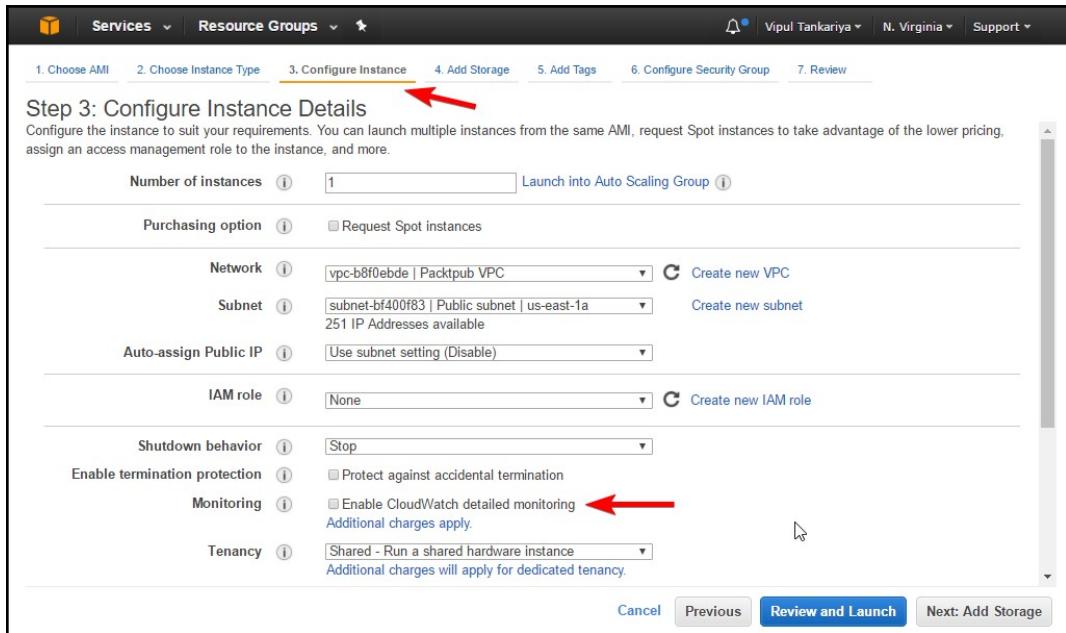


Figure 7.14: Enabling detailed monitoring while launching an EC2 instance

To enable detailed monitoring on an existing EC2 instance, follow these steps:

1. Navigate to <https://console.aws.amazon.com/ec2/> and open an EC2 console from your browser.
2. From the left-hand navigation pane, select **Instances**.
3. Once the instance list is launched, select an instance for which you want to change the monitoring type.
4. Select **Actions** | **CloudWatch Monitoring** | **Enable Detailed Monitoring**.
5. In the **Enable Detailed Monitoring** dialog box, choose **Yes, Enable**.
6. Click on **Close**.

CloudWatch best practices

Here are the best practices that we can follow in CloudWatch:

- Monitor AWS resources and hosted applications using CloudWatch. This helps to identify performance bottlenecks and to optimize resource costs.
- Enable billing alerts for an AWS account. This helps you to monitor the monthly costs and keep tabs on them.
- For a better understanding of the CloudWatch visualization, toggle the metrics between UTC and local time.
- By default, CloudWatch provides basic monitoring for resources and records metrics at 5-minute intervals. It is recommended that you use detailed monitoring for critical resources, which records metrics at 1-minute intervals.
- Enable custom metrics where required. For example, you can enable memory monitoring on EC2 instances, which is not part of the default EC2 metrics. Create custom metrics for monitoring application behavior and link them to CloudWatch. These provide better insight on the application. For example, custom metrics for monitoring JVM can be created for Java-based applications.
- Enable Auto Recovery and Auto Restart Alarms for your critical EC2 instances. It can automatically recover the instances from hardware-related or virtualization host-related issues on AWS. Also, Auto Restart Alarm can recover the instance from operating-system-level issues.
- Automate monitoring tasks as much as possible. Upload your critical custom logs to CloudWatch for quick statistical analysis. When creating custom metrics, verify log files on your EC2 instances with CloudWatch metrics to ensure that the right data synchronizes with CloudWatch.
- Enable SNS notifications for a critical metrics threshold breach.
- AWS provides 10 alarms per month per customer for free. If you intend to operate in a free tier, ensure that you do not exceed this limit.
- AWS supports a maximum of up to 5,000 alarms for a customer in a region. If you are a heavy alarms user, plan your alarms in such a way that all critical alarms are created before reaching the limit.

- CloudWatch does not validate actions while configuring them. Ensure that configured actions exist and are validated properly. For example, ensure your SNS group has valid email IDs associated with it. Similarly, ensure that the Auto Scaling group has a valid launch configuration associated with it.
- When testing an application associated with an alarm, you can temporarily disable alarms rather than getting flooded with alerts or unwanted actions being triggered.
- In certain conditions, AWS resources may not send metric data to CloudWatch. For example, an EBS volume does not send data to CloudWatch if it is not attached to an instance. When troubleshooting any metrics availability issue, ensure that the required conditions for monitoring the metrics for the resource are met.

Summary

- CloudWatch is an AWS service that can be used for monitoring various infrastructures and application resources running on your AWS cloud.
- CloudWatch acts as a repository of metrics by collating raw data from various AWS services or applications and converting it into metrics, statistics, and graphs, and facilitates certain actions based on specific data points in metrics.
- CloudWatch namespaces are containers in which metrics for different applications are stored. It is a mechanism to isolate metrics of different applications from each other.
- Metrics are sets of data collected over a period of time with a specific time interval for quantitative assessment, measurement, and comparison of performance data that is generated by a specific application or a service.
- A dimension in a CloudWatch metric is a mechanism to uniquely identify metrics. It is a name/value pair that is associated with metrics.
- Statistics are collections of aggregated metrics data for a specific period of time. Metrics data is aggregated using namespaces, metric names, dimensions, and several data points in a given time period.

- A percentile helps in finding the comparative standing of a value in a set of data.
- CloudWatch alarms help to define a threshold value that is constantly monitored, and an action is triggered when the threshold condition is breached.
- CloudWatch can also monitor the monthly billing charges for an AWS account and can generate billing alerts.
- Amazon CloudWatch provides a customizable dashboard inside a web console. It can display a set of critical metrics together.
- CloudWatch basic monitoring is free and it collects data at 5-minute intervals.
- CloudWatch detailed monitoring is chargeable and it makes data available at 1-minute intervals.

8

Simple Storage Service, Glacier, and CloudFront

Before we describe what Amazon S3 is, let's look at some basic concepts of storage. Storage services are usually categorized based on how they work and how they are used. Specifically, there are three broad types of storage services—block storage, file storage, and object storage:

- **Block storage:** In simple terms, block storage is a type of storage that is not physically attached to a server, but is accessed as a local storage device, just like a hard disk drive. At the backend, the storage service provider creates a cluster of disks, divided into a number of storage blocks. Each block is virtually connected to a server and treated as local storage. The server OS manages the block of storage assigned to it. For example, AWS EBS is a block storage type. When you provision a 100 GB EBS volume, a block of 100 GB is assigned from the cluster of disks to that volume. The EBS volume is associated with an EC2 instance. The volume is subsequently formatted and a filesystem is created on it. This volume is managed by the respective operating system installed on the EC2 instance for storing and retrieving data on EBS.

As each block of storage is treated as a local disk, block storage works well for creating filesystems and installing operating systems and databases. Even though the block storage architecture is comparatively complex, it provides high performance.

- **File storage:** File storage is also known as file-based storage. It is a highly available, centralized place for storing your files and folders. You can access file-level storage using file-level protocols such as **Network File System (NFS)**, **Server Message Block (SMB)**, and **Common Internet File System (CIFS)**. You can use file storage to store and retrieve files and folders. Just like in block storage, you can edit files stored in file storage. Unlike in block storage, you do not have access to format file storage, create a filesystem, and install an operating system on it. It is centrally managed by the service provider.
- **Object storage:** Object storage is a type of storage architecture where the data is stored as objects. Each object consists of the data, metadata, and a globally unique identifier. Metadata is data about data, and it provides basic information about the data stored in an object. Unlike in block storage and file storage, you cannot edit the data stored in object storage. If you need to edit a file, you have to create a new file and delete the old file (or keep multiple versions of the same file).

The following table provides a comparison between block storage, file storage, and object storage:

Features	Block storage	File storage	Object storage
Unit of transaction	Blocks.	Files.	Objects, files with metadata.
How you can update	You can directly update the file.	You can directly update the file.	You cannot update the object directly. You create a new version of the object and replace the existing one, or keep multiple versions of the same object.
Protocols	SCSI, Fiber Channel (FC) , SATA.	SMB, CIFS, NFS.	REST/SOAP over HTTP/HTTPS.
Support for metadata	No metadata support; it only stores filesystem attributes.	No metadata support; it only stores filesystem attributes.	Supports custom metadata.
Usage	For creating filesystems, installing OSes, and storing transactional data.	Used as centralized and shared file storage.	Used as a cloud storage for storing static files and data.

Strength	High performance.	Simple to access, highly available, and easy for sharing files.	Scalable, available over the internet and distributed access.
Weakness	Restricted to data center capacity.	Restricted to data center capacity.	Not suitable for in-place editing of data.
Can you format it?	Yes; you get complete access to format it and manage filesystems on it.	No, you cannot format it. It's a shared service wherein you only have access to manage data on it.	No, you cannot format it. It's a shared service wherein you only have access to manage objects on it.
Can you install an OS on it?	Yes; block storage can be used as a root volume and you can have an OS on it.	No, you cannot install an OS on it.	No, you cannot install an OS on it.
Pricing	You are charged based on allocated volume size, irrespective of how much data you store on it.	You are charged based on the amount of data you store on it.	You are charged based on the amount of data you store on it.
Example of specific storage service	EBS.	Amazon Elastic File System (EFS) .	S3.

We will be covering the following topics in this chapter:

- Introducing Amazon S3
- Creating a bucket
- Understanding objects
- S3 storage classes
- Life cycle management
- Hosting a static website on S3
- Cross-origin resource sharing
- Cross-region replication

Introducing Amazon S3

S3 is a cloud-based object storage service from Amazon. It is highly scalable and makes it easy to access storage over the internet. You can use S3 for storing and retrieving virtually unlimited amounts of data, at any time, from anywhere. It provides you with access to a highly scalable, reliable, efficient, and low-cost storage infrastructure that is used by Amazon to run its own global network of websites.

S3 is recommended for storing static content such as graphics files, documents, log files, audio, video, and compressed files. Virtually any type of data in any file format can be stored on S3. Currently, the permissible object size in S3 is 0 bytes to 5 TB. Objects in S3 are stored in a bucket. A **bucket** is a logical unit in S3 that is just like a folder. Buckets are created at the root level in S3 with a globally unique name. You can store objects, and also folders, inside a bucket. Any number of objects can be stored in each bucket. There is a soft limit of 100 buckets per account in S3.

S3 can be used for content storage and distribution, static website hosting, big data object stores, backup and archival, storing application data, and disaster recovery. Using JavaScript SDK and DynamoDB, you can also host dynamic applications on S3.

The following section describes the concepts and terminologies used in S3:

- **Buckets:** A bucket is a logical unit in S3, just like a folder. It is a container in which you can store objects, and also folders. Buckets are created at the root level in S3 with a globally unique name. Any number of objects can be stored in each bucket. For example, if you store an object named `books/acda.pdf` inside the `packtpub` bucket, then it is accessible using the `https://packtpub.s3.amazonaws.com/books/acda.pdf` URL. Buckets are generally used for organizing objects in S3. They are associated with an AWS account that is responsible for storing and retrieving data on the bucket. The account that owns the bucket is charged for data transfer. Buckets play a vital role in access control and pave the way for creating usage reports on S3. Buckets can be created in a specific region. You can enable version control on a bucket. If version control is enabled on a bucket, it maintains a unique version ID against each object stored in it.

- **Objects:** Objects are the basic entities stored in S3. Each object consists of the data, metadata, and a globally unique identifier. Metadata is stored in a set of name/value pairs; for example, **Date Last Modified**, **Content Type**, and **Content-Length**. There can be two types of metadata associated with an object—system-defined metadata and user-defined metadata. An object is identified with a unique key (name) within the bucket and a version ID, if versioning is enabled on the bucket.
- **Keys:** A key is a name that is assigned to an object. It is a unique identifier or name for an object within a bucket. Every object in a bucket has only one key associated with it. The combination of a bucket, a key, and its respective version ID uniquely identifies an object within a bucket. Every object within a bucket has a unique address for accessing it through a web service endpoint. The address URL consists of the bucket name, the key, and a version number, if versioning is enabled on the bucket.
For example,
consider <https://packtpub.s3.amazonaws.com/books/acda.pdf>. In this example, packtpub is the name of the bucket and books/acda.pdf is the key:



Figure 8.1: Object URL in S3

- **Region:** A region is a geographical region where Amazon S3 stores a bucket, based on user preferences. Users can choose a region while creating a bucket, based on the requirement. Ideally, a bucket should be created in the closest geographical region where the bucket needs to be accessed. Choosing the closest region while creating a bucket optimizes latency, while accessing the bucket reduces costs and complies with any regulatory requirements an organization may have.
Currently, Amazon has the following regions:

Region	Location of S3 servers
US East (N. Virginia)	Northern Virginia
US East (Ohio)	Columbus, Ohio
US West (N. California)	Northern California
US West (Oregon)	Oregon

Canada (Central)	Canada
Asia Pacific (Mumbai)	Mumbai
Asia Pacific (Seoul)	Seoul
Asia Pacific (Singapore)	Singapore
Asia Pacific (Sydney)	Sydney
Asia Pacific (Tokyo)	Tokyo
China (Beijing)	Beijing
EU (Frankfurt)	Frankfurt
EU (Ireland)	Ireland
EU (London)	London
EU (Paris)	Paris
South America (Sao Paulo)	Sao Paulo

S3-supported regions

When you create an object in a region, it is stored in the same region, unless you explicitly copy it over to any other region.

S3 data consistency model: Amazon provides two types of consistency model for S3 data when you perform various input/output operations with it: **read-after-write consistency** and **eventual consistency**.

The following table describes both of these data consistency models in S3:

Input/output operation	Data consistency model	Exception, if any
PUTS of new object	Read-after-write consistency	S3 gives eventual consistency for HEAD or GET requests while retrieving the key name before creating an object
Overwrite PUTS and Deletes	Eventual consistency	No exception

Data consistency model

Amazon provides read-after-write consistency for the PUTS of a new object. This means that if you create a new object in S3, you can immediately read it. Amazon provides eventual consistency for overwrite PUTS and Deletes operations. This means that it takes a few seconds before the changes are reflected in S3 when you overwrite or delete an existing object in S3.

Amazon replicates data across the region in multiple servers located inside Amazon data centers. This replication process provides high availability for data. When you create a new object in S3, the data is saved in S3; however, this change must replicate across the Amazon S3 regions. Replication may take some time, and you may observe the following behavior:

- After you create a new object in S3, Amazon immediately lists the object keys within the bucket, and the new object keys may not appear in the list.
- When you replace an existing object and immediately try to read the object, Amazon may return old data until the data is fully replicated.
- When you delete an object and immediately try to read it, Amazon may read the data for the deleted object until the deletion is fully replicated.
- When you delete an object in a bucket and immediately list the contents of the bucket, you may still see the deleted object in the contents of the bucket until the deletion is fully replicated.

Creating a bucket

The following steps describe the process of creating a bucket using the AWS Management Console:

1. Sign in to your AWS account and go to the S3 console, or visit <https://console.aws.amazon.com/s3/>. If you already have buckets in the account, this will display a list of the buckets; otherwise, you will see the following screenshot, stating that you do not have any buckets and how you can get started using S3:

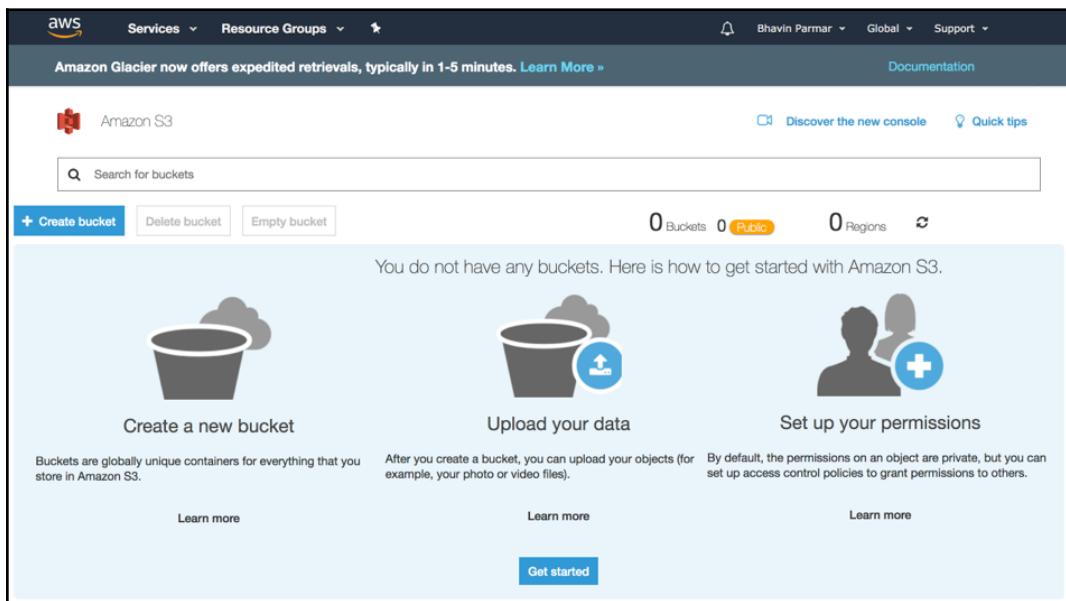


Figure 8.2: S3 console

2. Click on the + Create bucket icon, displayed in the following screenshot:



Figure 8.3: Create bucket

3. Clicking on the **+ Create bucket** button will display a popup, as shown in the following screenshot. Enter a DNS-compliant bucket name. The bucket name field must be unique across all existing bucket names in S3. Since S3 is a shared service, it is likely that you will not always get the bucket name you want, as it might have been taken by someone already. Select the appropriate region where you want to create the bucket from the drop-down menu, as indicated in the following screenshot. If you already have some buckets, you can **Copy settings from an existing bucket**. You can also click on the **Create** button if you do not want to follow the remaining steps. You will need to set the bucket properties and permissions later on if you immediately click on the **Create** button. To understand these steps, you can click on the **Next** button:

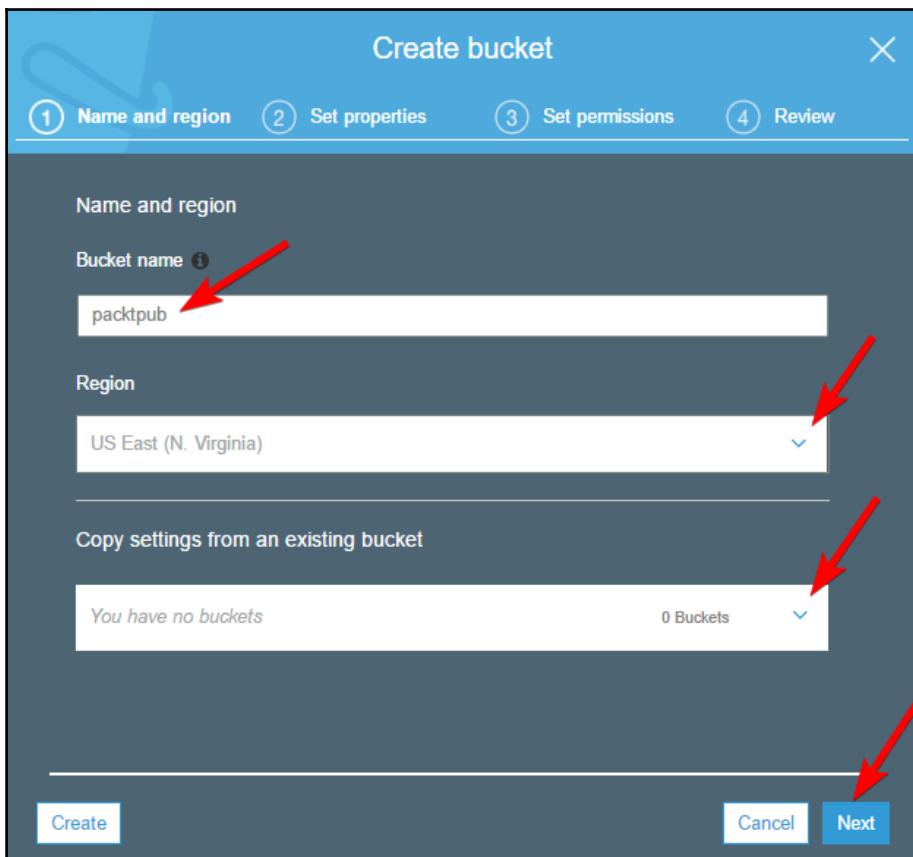


Figure 8.4: Create bucket screen

4. On the subsequent screen, as shown in the following screenshot, you can set the required properties. In the following screenshot you can see that by default, versioning is disabled, logging is disabled, encryption is disabled, and there are no tags. You can click on versioning and logging as required, or you can add tags as needed. When you click on these items, it will display the respective popups, as shown in the following screenshot. You can set the required properties as needed:

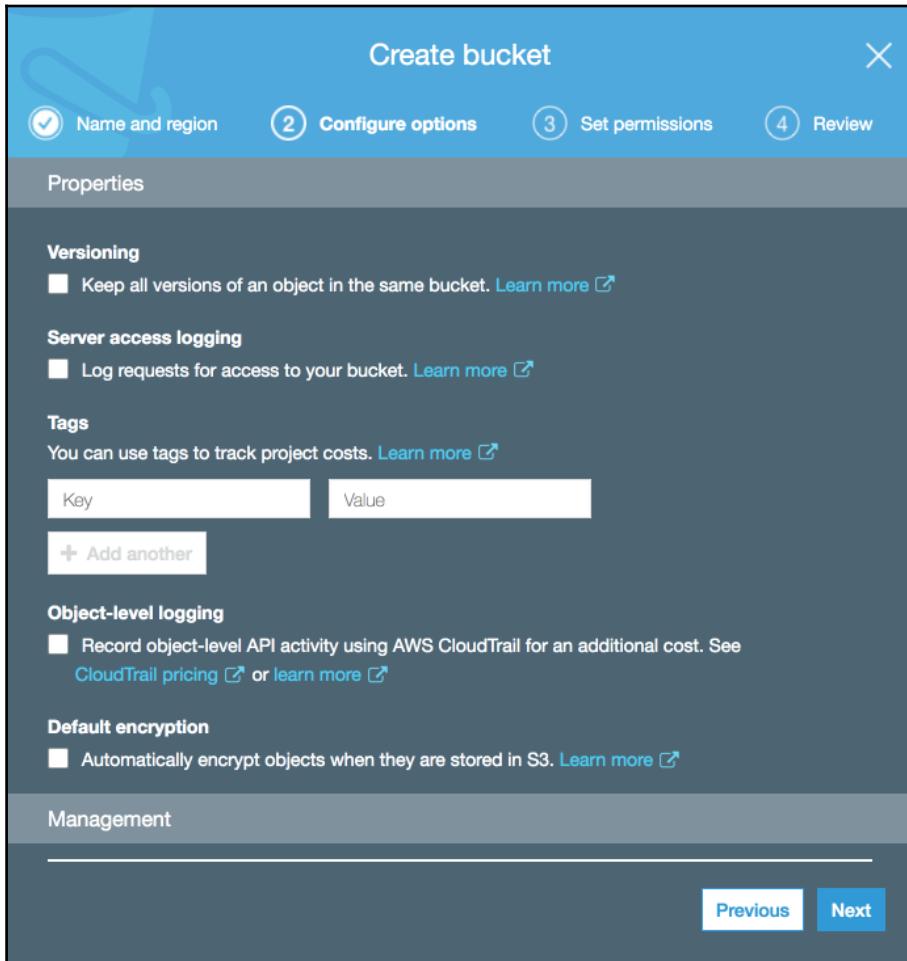


Figure 8.5: Bucket properties in the Create bucket wizard

5. On the subsequent screen, as shown in the following screenshot, you can set folder permissions. You can set individual user permissions, manage public permissions, and manage system permissions:

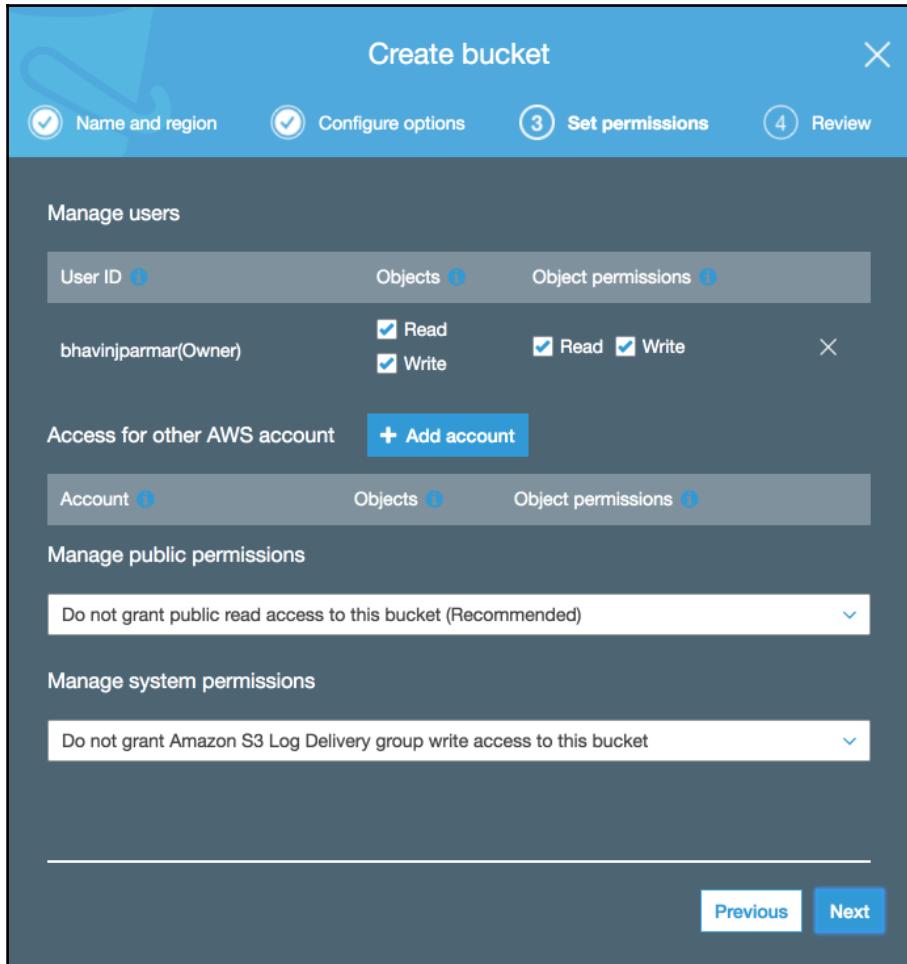


Figure 8.6: Managing bucket permissions in the Create bucket wizard

6. On the subsequent screen, as shown in the following screenshot, you can review your selections. If required, you can edit your selections under individual categories. After reviewing everything, click on the **Create bucket** button. It creates a bucket as per the input you've provided:

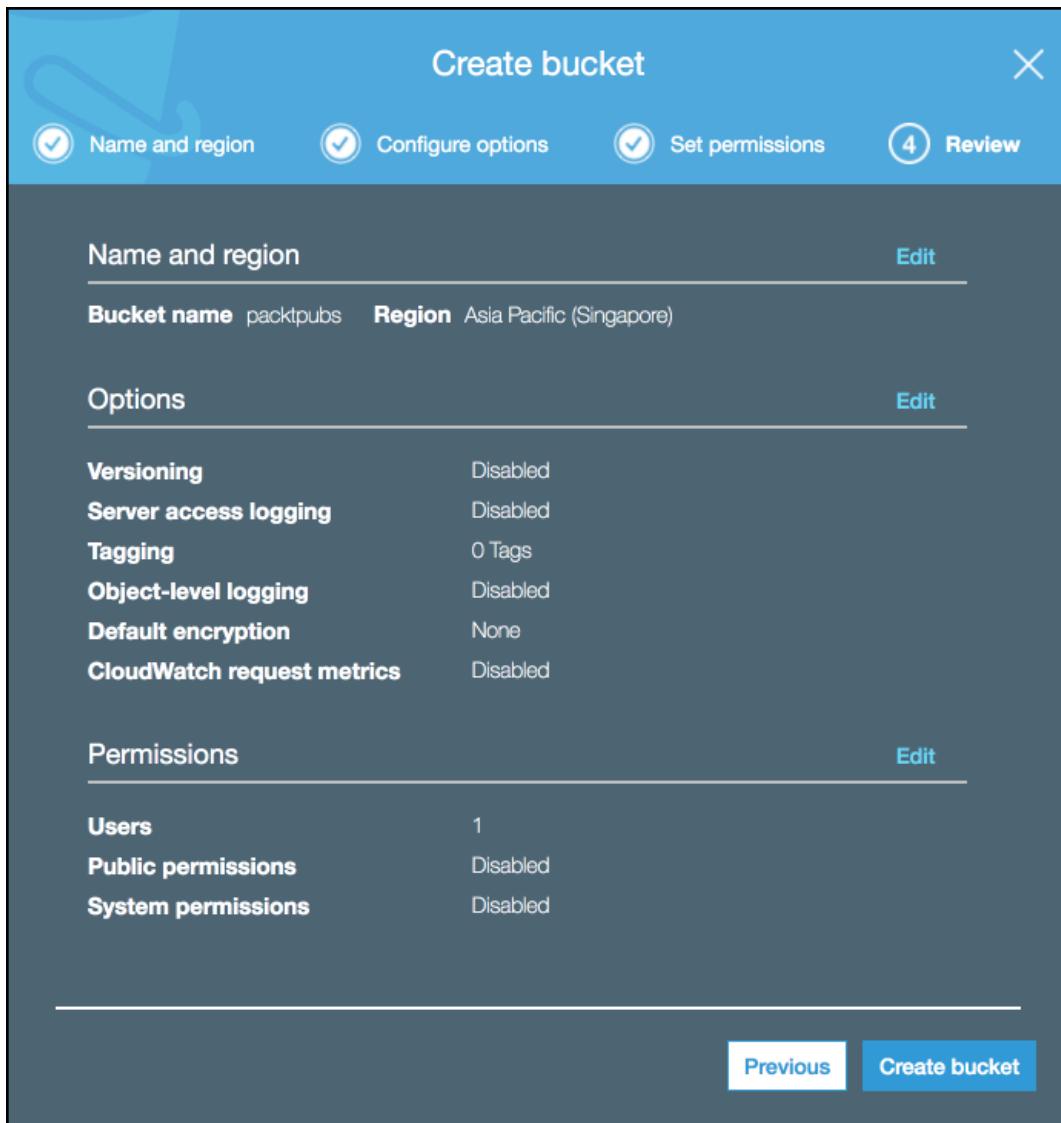


Figure 8.7: Review your steps in the Create bucket wizard

Bucket restrictions and limitations

Bucket restrictions and limitations are listed as follows:

- You can create a bucket using the S3 console, APIs, or the CLI.
- Amazon imposes a soft limit of 100 buckets on an AWS account. You can increase this soft limit by raising a support request with Amazon.
- When you create a bucket, it is associated with the AWS account and the user you have used to create it. Bucket ownership cannot be transferred to another AWS account or another user within the same AWS account.
- There is no limit to the number of objects that can be created in a bucket.
- A bucket is created at the root level in S3; you cannot create a bucket inside a bucket.
- If you use an application that automatically creates a bucket, ensure that the application chooses a different bucket name if the bucket name generated by the application already exists.

Bucket names should comply with DNS naming conventions, as follows:

- Bucket names can range from 3 to 63 characters.
- Bucket names must be in lowercase letters. They can contain numbers and hyphens.
- Bucket names must start and end with a lowercase letter or a number.
- Bucket names must not be given as an IP address; for example, 192.168.1.10.
- It is recommended that you avoid using periods (.) in bucket names.

Bucket access control

Each bucket in S3 is associated with an access control policy, which governs how objects are created, deleted, and enumerated within the bucket.

When you create an S3 resource, all S3 resources, including buckets, objects, the life cycle policy, and static website configuration, are private by default. Only the resource owner that created the resource can access that resource. After creating the resource, the resource owner can optionally grant permissions to other users using an access control policy.

There are two types of access control policy:

- Resource-based policies
- User policies

Access policies that you associate with buckets and objects are called **resource-based policies**. Bucket policies and **access-control lists (ACLs)** are examples of resource-based policies. Access policies that you associate with users are called **user policies**. You can use a resource-based policy or a user policy (and at times, a combination of both) to control access to your S3 resources.

Bucket policy

A bucket policy generally comprises the following elements:

- **Resource:** This indicates Amazon S3 resources, such as buckets and objects. While creating a policy, you can specify the ARN to allow or deny permissions on that resource.
- **Action:** This indicates one or more actions that are either allowed or denied. For example, `s3:GetObject` specifies the permission to read the object data. Similarly, `s3>ListBucket` specifies the permission to list objects in the bucket.
- **Effect:** This specifies the action type: either `Allow` or `Deny` access. If permission is not explicitly granted on a resource, by default, access is denied. When you explicitly `Deny` access, it ensures that the user cannot access the resource, even if another policy grants access.
- **Principal:** This indicates the account/user who is allowed or denied access to the resources mentioned in the policy statement. In a user policy, you may not need to specify a principal. A user policy implicitly applies to the associated user.
- **Sid:** This is an optional identifier known as the **statement ID**, which is specified for the policy statement. Sid values must be unique within the policy statement. The following is an example of a bucket policy. The example policy allows the user `Heramb` the following three permissions on the bucket named `packtpubs`:
 - `s3:GetBucketLocation`
 - `s3>ListBucket` `s3:GetObject`

In the following policy statement, Account-ID should be replaced with the AWS account number:

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "Statement1",  
      "Effect": "Allow", "Principal": {  
        "AWS": "arn:aws:iam::Account-ID:user/Heramb"  
      },  
      "Action": [ "s3:GetBucketLocation", "s3>ListBucket", "s3:GetObject"  
      ],  
      "Resource": [ "arn:aws:s3:::packtpubs"  
      ]  
    }  
  ]  
}
```

In the same policy, if you change the effect from Allow to Deny, it explicitly denies the user Heramb access to the packtpubs bucket to perform the specific set of actions mentioned in the policy statement.

User policies

Access policies are associated with users or groups. Unlike with a bucket policy, you don't need to specify the principal in a user policy. A policy is implicitly applied to the user with whom it is associated.

An example of a user policy is as follows:

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

```
],
"Resource": "arn:aws:s3:::packtpubs"
},
{
"Effect": "Allow", "Action": [
"s3:PutObject", "s3:GetObject", "s3:DeleteObject"
],
"Resource": "arn:aws:s3:::packtpubs/*"
}
]
```

There are three parts to the preceding user policy example:

- The first part describes permission to list all the buckets using a `ListAllMyBuckets` action against `arn:aws:s3:::*`, which signifies all the resources in S3 for the account.
- The second part describes the `ListBucket` and `GetBucketLocation` permissions on the `packtpubs` bucket.
- The third part describes the permissions to create, read, and delete objects in the `packtpubs` bucket.

Once a user policy is created, it can be attached to a user or a group to grant them the respective access specified in the policy.

Transfer Acceleration

When you need to transfer a very large amount of data between your on-premises environment and S3, time, efficiency, and the security of the data play a very vital role. Under such requirements, S3 **Transfer Acceleration** can be very handy. It provides a fast, easy, and secure way to transfer files between S3 and any other source or target of such data transfers. For Transfer Acceleration, Amazon uses CloudFront edge locations. CloudFront edge locations are spread across the globe, which facilitates the Transfer Acceleration process.

The scenarios in which you should use Transfer Acceleration are as follows:

- You have a centralized bucket that your end customers use from across the globe for uploading data.
- You regularly transfer GBs and TBs of data across continents.
- The available bandwidth is underutilized while you transfer data to S3.

Enabling Transfer Acceleration

The steps for enabling Transfer Acceleration are as follows:

1. Log in to the AWS Management Console and go to the S3 console, or browse to <https://console.aws.amazon.com/s3>.
2. Open the bucket in which you need to enable Transfer Acceleration.
3. Click on the **Properties** tab, as shown in the following screenshot:

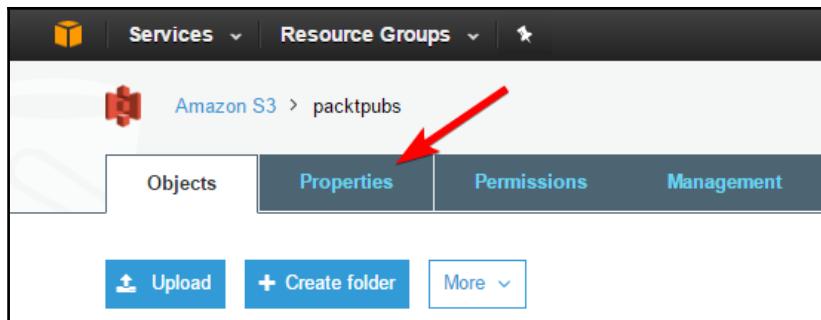


Figure 8.8: Selecting bucket properties

4. In the **Properties** tab, under **Advanced settings**, click on **Transfer acceleration**. It brings up a popup to enable or suspend Transfer Acceleration, as shown in the following screenshot. You can select **Enabled** in this popup to enable Transfer Acceleration on the selected bucket. Click on the **Save** button after the selection is done:

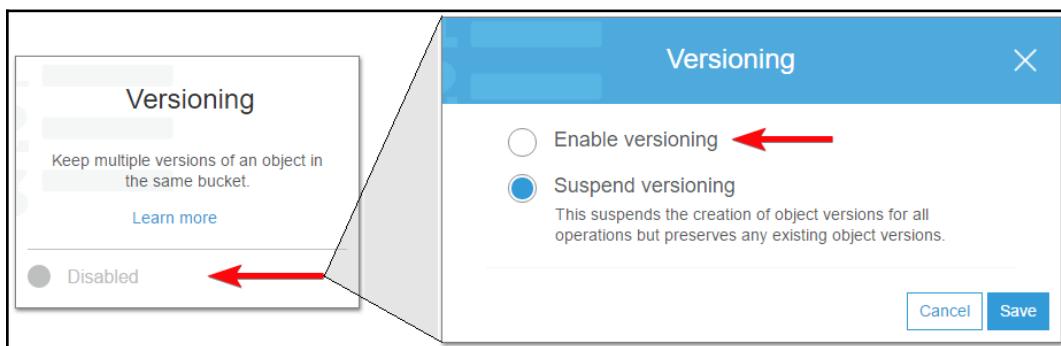


Figure 8.9: Enabling Transfer Acceleration on a bucket

Requester Pays model

Generally, when you create a bucket in S3, you pay for data storage and data transfer. Based on your usage, charges are added to the AWS account associated with the bucket. Amazon provides an option in which you can configure your bucket as a **Requester Pays** bucket. When you configure a bucket as a Requester Pays bucket, the requester pays for the requests they initiate to download or upload data in the bucket. You just pay for the cost of the data you store in S3:

- You can enable Requester Pays on a bucket when you want to share the data but do not want to get charged for the requests received, data downloads, or upload operations.
- When you enable Requester Pays, AWS does not allow you to enable anonymous access on the bucket.
- All requests to Requester Pays buckets must be authenticated; when you enable authentication, S3 can identify requesters and charge them for their respective usage of the bucket.
- If a system or application makes requests by assuming an IAM role, AWS charges the account where the assumed role belongs.
- If you make calls to the bucket using an application, the request must include `x-amz-request-payer` in the header section if you make POST, GET, and HEAD requests.
- If you make a REST request, you need to include `x-amz-request-payer` as a parameter in the request.
- Requester Pays buckets do not support anonymous requests, BitTorrent, and SOAP requests.
- Amazon does not allow you to enable end user logging on a Requester Pays bucket, and similarly, you cannot enable Requester Pays on a bucket where end user logging is enabled.

Enabling Requester Pays on a bucket

You can enable Requester Pays on a bucket in steps that are very similar to those you followed in Transfer Acceleration:

1. Log in to the AWS Management Console and go to the S3 console, or browse to <https://console.aws.amazon.com/s3>.
2. Open the bucket on which you need to enable Requester Pays.

3. Click on the **Properties** tab, as shown in *Figure 8.8* in the previous section.
4. Click on **Requester Pays** to enable it.

Understanding objects

Objects are the basic entities stored in S3. Amazon has designed S3 as a simple key value store. You can store a virtually unlimited number of objects in S3. You can segregate objects by storing them in one or more buckets.

Objects consist of a number of elements, that is, a key, a version ID, a value, metadata, subresources, and access control information. Let's look at these object elements:

- **Key:** The key is the name that is assigned to an object. It's just like a filename and can be used to access or retrieve the object.
- **Version ID:** If you enable versioning on a bucket, S3 associates a version ID with each object. The bucket may have one or more objects with the same key, but a different version ID. The version ID helps in uniquely identifying an object when there are multiple objects with the same key.
- **Value:** The value refers to the content or data that is stored on the object. It is generally a sequence of bytes. The minimum size of an object can be zero, and the maximum is 5 TB.
- **Metadata:** S3 stores reference information related to an object in its metadata, in the form of name/value pairs. There are two types of metadata: system metadata and user-defined metadata. System metadata is used for managing objects, and user-defined metadata is used for managing information related to objects.
- **Subresources:** An object can have subresources associated with it. Subresources are defined and associated with objects by S3. There can be two types of subresources associated with an object, and they are **ACL** and **torrent**:
 - ACL contains a list of users and respective permissions granted to them. When you create an object, the ACL entry only contains an owner. Optionally, you can add more users, with the required permissions for each user.

- A torrent is another sub-resource of an object. AWS supports the BitTorrent protocol. It is very simple to access S3 objects using a BitTorrent client. If you assign an anonymous permission on an object, that object can be accessed by a BitTorrent client by referring to the object URL with ?torrent at the end. Once an object URL is accessed with ?torrent at the end of it, AWS automatically creates a .torrent file for that object. Subsequently, you can distribute the .torrent file to end users to access the object using a BitTorrent client.
- **Access control information:** Amazon S3 enables you to control access on the objects you create using ACLs, bucket policies, and user policies. Access control information is nothing but the information containing permissions in the form of an ACL, a bucket policy, or user access policies.

Object keys

When you create an object in S3, you need to give a unique key name to the object in the bucket. A key name uniquely identifies an object in the bucket. When you browse a bucket in the S3 console, it displays a list of objects within the bucket. The list of names within the bucket are object keys.

An object key is a sequence of Unicode characters in UTF-8 encoding. A key name can be a maximum of 1,024 bytes long.

Object key naming guide

Each application applies its own mechanism to parse special characters. It is recommended that you follow best practices while naming an object key. The following best practices provide maximum compliance with DNS, web-safe characters, XML parsers, and various other APIs:

- An object key name consists of alphanumeric characters (0–9, a–z, and A–Z) and special characters, such as !, –, _, ., *, ', (, and).
- S3 can store buckets and objects. It does not have any hierarchical structure; however, prefixes and delimiters used in an object key name allow S3 to use folders.

- Key name examples of how S3 supports folders are as follows:
 - projects/acda-guide.xlsx
 - books/aws-networking.pdf
 - outlines/vpc.xlsx
 - help.txt
- In the previously mentioned examples, S3 uses key name prefixes such as projects/, books/, and outlines/. These key name prefixes, with / as the delimiter, enable S3 to represent a folder structure. The following screenshot shows the folder structure in S3:

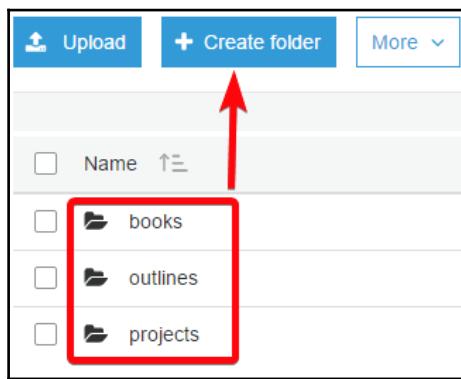


Figure 8.10: Folder structure in S3

When you open a folder, it displays objects inside the folder. The S3 console displays the bucket and folder in the breadcrumb, as shown in the following screenshot:

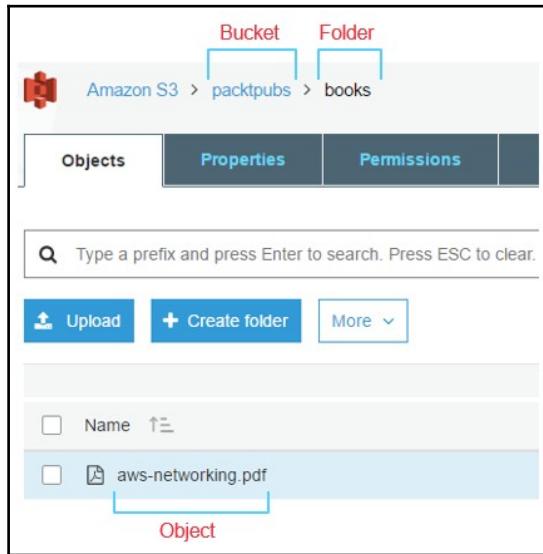


Figure 8.11: Objects inside a folder

The following is a list of special characters that require special handling if you use them in an object key name. Some characters may not be properly rendered by a browser or application. If you plan to include the following characters in S3 object key names, it is recommended that you build the appropriate logic to handle them in your application:

Ampersand (&)	Dollar (\$)	ASCII character ranges 00-1F hex (0-31 decimal) and 7F (127 decimal)
At (@)	Equals (=)	Semicolon (;)
Colon (:)	Plus (+)	Space-significant sequences of spaces may be lost in some uses (especially multiple spaces)
Comma (,)	Question mark (?)	

AWS recommends avoiding the following characters in object key names:

Backslash (\)	Left curly brace ({})	Non-printable ASCII characters (128-255 decimal characters)
Caret (^)	Right curly brace (})	Percent character (%)
Grave accent/back tick (`)	Right square bracket ([])	Quotation marks
Greater than symbol (>)	Left square bracket ([])	Tilde (~)
Less than symbol (<)	Pound character (#)	Vertical bar/pipe ()

Object metadata

S3 stores reference information related to an object in its metadata, in the form of name-value pairs. There are two types of metadata:

- System-defined metadata
- User-defined metadata

System-defined metadata

Amazon stores a set of system-defined metadata with every object in S3. For example, S3 stores the object creation date as well as the size of the object in object metadata. There are two types of system metadata: one in which only S3 can change the value of metadata, such as the object creation date and size, and other types of system metadata, such as storage class and server-side encryption, that can be controlled by users, based on selections.

The following table displays a list of system-defined metadata:

Name	Description	Can user modify the value?
Content-Length	Indicates object size, in bytes.	No
Content-MD5	Indicates a Base64-encoded 128-bit MD5 digest of the object.	No
Date	Indicates the current date and time.	No

Last-Modified	Indicates the date of the last modification on the object. It can be the creation date if the object is not modified after the initial creation.	No
x-amz-delete-marker	This is displayed against objects in a bucket where versioning is enabled. It indicates whether an object is marked for deletion.	No
x-amz-server-side-encryption	This metadata indicates whether server-side encryption is enabled on an object.	Yes
x-amz-server-side-encryption-aws-kms-key-id	When x-amz-server-side-encryption is enabled on an object and it includes aws:kms, it indicates the ID of the KMS master encryption key that is used for the object.	Yes
x-amz-server-side-encryption-customer-algorithm	This indicates whether server-side encryption is enabled with customer-provided encryption keys (SSE-C).	Yes
x-amz-storage-class	This indicates what storage class is used for storing the object.	Yes
x-amz-version-id	When versioning is enabled on a bucket, this metadata indicates the version of the object.	No

x-amz-website-redirect-location	When website hosting is enabled on an S3 bucket, this metadata indicates the redirection URL if the request redirection is configured.	Yes
---------------------------------	----------------------------------------------------------------------------------------------------------------------------------------	-----

User-defined metadata

Amazon S3 allows users to assign user-defined metadata to an object. When you create an object in S3, you can provide optional metadata as a name-value pair.

User-defined metadata is generally used for associating additional information with an object. Such metadata can help in identifying objects. It can also be used for automating data management tasks using scripts. For example, a script may traverse through all the objects in a bucket and check for specific metadata on an object. If a desired key/value pair of a metadata is assigned to an object, the script may further process the data in the object. User-defined metadata must begin with `x-amz-meta-`.

Here is how you can assign metadata to an object using the S3 console:

1. Log in to the AWS console and go to the S3 console.
2. Open the required bucket.
3. Click on the object on which you want to define metadata.
4. Click on the **Actions** drop-down menu.
5. Click on **Metadata**.
6. Click on **Add Metadata**.
7. Select `x-amz-meta-book-type` from the drop-down menu and type the remaining value in **Key**, as well as **Value**, and then click on **Save**.

Versioning

S3 allows you to keep multiple versions of an object in a bucket. Versioning can be enabled at the bucket level. Once versioning is enabled, it protects you from accidental updates and deletions on an object. When you overwrite or delete an object, it keeps multiple copies with version numbers.

For example, when you enable versioning on a bucket called `packtpub`, for each action on an existing object in the bucket, S3 creates a new version and associates a version ID with it, as shown in the following table:

Object	Last activity	Version ID
developer-guide.pdf	Jun 12, 2017 9:42:02 AM	VAgAtLChtLoMkKF4ZVoq.NAGRRBA1hSp
developer-guide.pdf	Jun 11, 2017 8:41:23 AM	3mWAzx.125VRt3.V.1ExutyOAEG1npX3
developer-guide.pdf	Jun 10, 2017 5:39:58 PM	hV_2iz3GgRvOTT1NoiL8KXg3FpLJkFI7

When you delete an object in a version-enabled bucket, S3 does not actually delete the object, but adds a delete marker to it.

Enabling versioning on a bucket

The steps for enabling versioning on a bucket are as follows:

1. Sign in to your AWS Management Console and go to the S3 console, at <https://console.aws.amazon.com/s3/>.
2. Click on the bucket in which you want to enable versioning, as shown in the following screenshot:

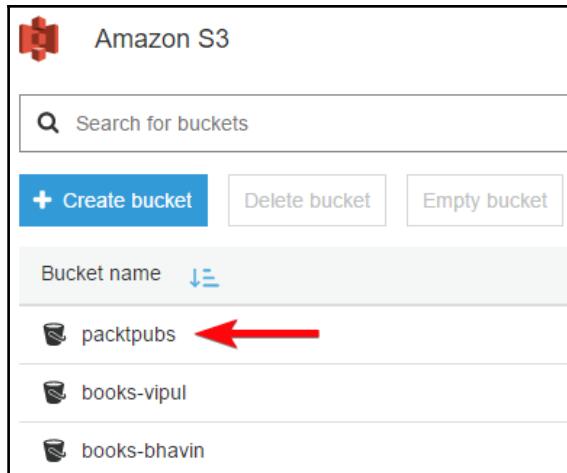


Figure 8.12: Select a bucket to enable versioning

3. Click on the **Properties** tab:



Figure 8.13: Select bucket properties

4. Click on **Versioning | Enable versioning**, and save the changes:

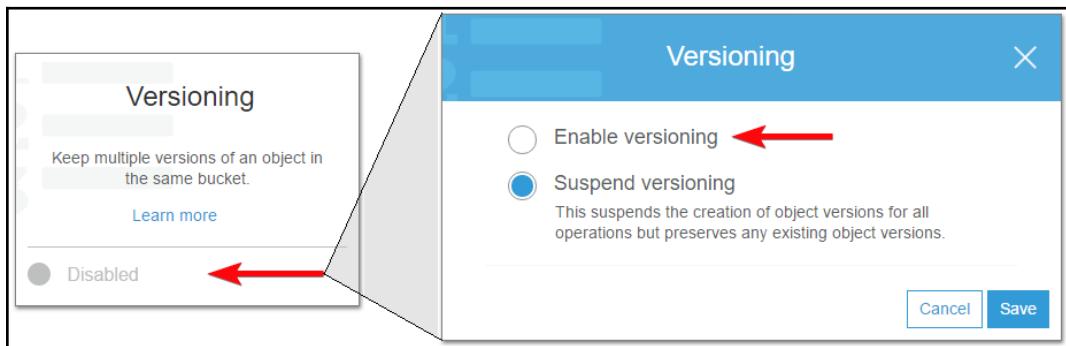


Figure 8.14: Enabling versioning

Object tagging

S3 allows you to add tags to your objects. Tagging an object helps to categorize the object. Each tag is a key and value pair.

The following is an example of tags on an object. Let's consider a scenario where an application processes data stored in an S3 bucket. While traversing through the objects in a bucket, it checks for a tag before processing the data in the object. In such scenarios, you may add the following tag to the objects:

```
Processed=True
```

Alternatively, you can use this tag:

```
Processed=False
```

The application may check for the tag in an object before processing the data in it. If the tag indicates `Processed=False`, then the application should process the data stored in the object and change the tag to `Processed=True`.

You can add tags to an object from the object properties in the S3 console. The syntax for adding tags to an object using the AWS CLI is as follows:

```
aws s3api put-bucket-tagging --bucket <Bucket> --tagging  
'TagSet=[{Key=<key>,Value=<value>}]'
```

An example of this is as follows:

```
aws s3api put-bucket-tagging --bucket packtpubs --tagging  
'TagSet=[{Key=Processed,Value=True}]'
```

Now that you understand what objects are, let's get into S3 storage classes.

S3 storage classes

Amazon S3 provides a number of storage classes for different storage needs. Storage classes are divided into the following five main types, based on how they are used:

- S3 Standard storage
- S3 **Infrequently Accessed (IA)**
- S3 One Zone-IA storage
- S3 **Reduced Redundancy Storage (RRS)**
- S3 Intelligent-Tiering
- Glacier

S3 Standard storage

S3 Standard storage is used as general-purpose storage for frequently accessed data. It provides high availability, durability, and high-performance storage for frequently accessed data. S3 Standard storage can be used in content distribution, cloud applications, big data analytics, mobile or gaming applications, and dynamic websites.

The key features of S3 Standard storage are as follows:

- Provides low-latency and high-throughput performance
- Ensures 99.99999999% durability for objects
- Provides 99.99% availability in a year, backed by the **Amazon S3 Service Level Agreement (SLA)**
- Enables SSL encryption of data in transit
- Supports AES-256 encryption of data at rest
- Supports data life cycle management for automatically migrating data from one class of storage to another

S3-IA storage

S3-IA storage is meant for data that is less frequently used, but needs to be available immediately when needed. It provides low-latency, high throughput, and durable data storage. It incurs relatively low GB storage and retrieval costs. Being a low-cost and high-performance storage option, S3-IA is best suited for backup, disaster recovery, and any long-term storage needs. You can keep S3-IA class objects within the same bucket with other class objects. It also supports object life cycle policies for automatically transitioning objects to other storage classes without requiring any modification of applications using objects. The key features of S3-IA are as follows:

- Suitable for long-term data storage, backup, and disaster recovery
- Provides low-latency and high-throughput performance, same as S3 Standard
- Ensures 99.99999999% durability for objects
- Provides 99.99% availability in a year, backed by the Amazon S3 SLA
- Enables SSL encryption of data in transit
- Supports AES-256 encryption of data at rest
- Supports data life cycle management for automatically migrating data from one class of storage to another

S3 One Zone-IA

In April of 2018, one more storage class was introduced. This is approximately 20% cheaper than the Amazon S3 Standard IA storage class, as it has a slightly lower availability and it is 99.5% over a year. Amazon S3 Standard IA replicates data in three **Availability Zones (AZs)**, while Amazon S3 One Zone-IA stores data only in one AZ. It is best suited to store infrequently accessed and reproducible data.

S3 RRS

As the name suggests, S3 RRS provides reduced levels of redundancy, as opposed to standard S3 storage. It is suitable for storing non-critical and reproducible data. It is a highly available storage solution for content distribution as a secondary storage for data that is available elsewhere, as well. It is ideal for storing thumbnails, transcoded media, or any other processed data that can be reproduced.

S3 stores RRS objects across multiple facilities and provides 400 times the durability of a local disk drive; however, RRS objects are replicated relatively infrequently, as compared to S3 Standard objects.

The following are some of the important characteristics of RRS:

- Provides a reduced level of redundancy.
- Comparatively cheaper than S3 Standard storage.
- It is backed by the Amazon S3 SLA.
- Provides 99.99% durability in a given year.
- Provides 99.99% availability in a given year.
- Designed to absorb data loss in a single facility.
- Enables SSL encryption of data in transit.
- Supports AES-256 encryption of data at rest.

S3 Intelligent-Tiering

In November, 2018, during its annual conference called **re:Invent**, Amazon announced a new S3 storage class called **S3 Intelligent-Tiering**. The S3 Intelligent-Tiering storage class can automatically optimize your storage cost by segregating your data to the most cost-effective S3 storage class, based on its usage pattern. S3 segregates your data into two access tiers, wherein one tier stores data that are frequently used, and another tier stores the data that is accessed infrequently.

When you enable S3 Intelligent-Tiering, S3 monitors your data and changes the storage class of objects to the infrequently accessed tier for objects that are not accessed for 30 consecutive days. Amazon charges a small monthly fee for monitoring and automating object tier selection.

Amazon does not charge retrieval fees for objects. You can directly store your objects in S3-Intelligent-Tiering, or you can configure S3 life cycle policies to automatically change the object class.

The following are some of the important characteristics of S3 Intelligent-Tiering:

- Automatically changes the object storage class based on the object usage pattern.
- Provides a durability of 99.999999999%.
- Provides an SLA of 99.99% availability in a year.
- Provides low-latency and high-throughput performance.
- Behind the scenes, data is stored in multiple availability zones, which provides resilience even in the case of an entire availability zone being down.
- Provides SSL support for data in transit.
- Data at rest can be encrypted using KMS.
- Supports S3 life cycle management for automatic storage class management on an object.
- Amazon charges a monthly fee for monitoring and auto-tiering the objects.

Glacier

Glacier is a very low-cost, secure, and durable data archival storage. You can store virtually unlimited amounts of long-term data on Glacier at a much cheaper rate. Glacier is ideal for storing long-term data, backups, archives, and data for disaster recovery. Unlike in S3, data stored on Glacier is not immediately available for access. You need to initiate a data retrieval request to access data on Glacier. To keep the costs low and still make it suitable for different retrieval requirements, Glacier provides the following three options for data retrieval:

Data retrieval option	Minimum time for retrieval	Comparative costs
Expedited retrieval	1 to 5 minutes	\$\$\$
Standard retrieval	3 to 5 hours	\$\$
Bulk retrieval	5 to 12 hours	\$

Comparison of S3 storage classes and Glacier

The following table compares the three storage classes of S3 with Glacier:

Description	Standard	Standard-IA	One Zone-IA	RRS	Glacier
Availability SLA	99.9%	99%	99.5%	N/A	N/A
Concurrent facility fault tolerance	2	2	0	1	N/A
Availability	99.99%	99.9%	????	99.99%	N/A
Durability	99.99999999%	99.99999999%	99.99999999%	99.99%	99.99999999%
First-byte latency	Milliseconds	Milliseconds	Milliseconds	Milliseconds	Select minutes or hours
Life cycle transitions	Yes	Yes	????	Yes	Yes
Minimum object size	N/A	128 KB	128 KB	N/A	N/A
Maximum object size	5 TB	5 TB	5 TB	5 TB	40 TB
Minimum storage duration	N/A	30 days	30 days	N/A	90 days
Retrieval fee	N/A	Per GB retrieved	Per GB retrieved	N/A	Per GB retrieved
SSL support	Yes	Yes	Yes	Yes	Yes
Storage class	Object level	Object level	Object level	Object level	Object level
Supported encryption at rest	AES-256	AES-256	AES-256	AES-256	AES-256
Data retrieval time	Immediately	Immediately	Immediately	Immediately	Minimum 3 to 5 hours
Recommended multipart upload size	100 MB	100 MB	100 MB	100 MB	100 MB

Life cycle management

Life cycle management is a mechanism in S3 that enables you to either automatically transition an object from one storage class to another storage class, or automatically delete an object, based on configuration. Life cycle rules can be applied to a group of objects based on filter criteria set in the rule.

S3 allows you to configure one or more life cycle rules, in which each rule defines a specific action. There are two types of actions that you can define in life cycle rules:

- **Transition actions:** This defines when an object storage class changes from an existing storage class to a target storage class. For example, you can define a rule for all object keys starting with `data/` in a bucket to transition from Standard storage to **STANDARD_IA** after 15 days. Similarly, you can define a rule to transition for all object keys starting with `data/` from **STANDARD_IA** to Glacier storage. Let's suppose that you have a bucket named `packtpubs`, and inside the bucket, you have a folder named `data`. Within the `data` folder you have `.csv` files. In such scenarios, this transition rule applies to all the files present in the `data` folder.
- **Expiration actions:** This defines when objects expire. When objects expire, Amazon S3 automatically deletes them for you. For example, you can set a rule for the object keys starting with `backup/` in a bucket to expire after 30 days. In such scenarios, all the files from the `backup` folder in a specific bucket expire after 30 days and are automatically deleted from S3.

Life cycle configuration use cases

It is advisable to configure life cycle rules on objects where there is absolute clarity on the life cycle of the objects. The following are the scenarios where you can consider defining life cycle rules:

- You have an application that generates and upload logs to an S3 bucket. The application does not need these logs after a week or a month. In such scenarios, you may want to delete these logs.
- You have a bucket in which users and applications are uploading objects. These objects are frequently accessed for a few days. After a few days of uploading, these objects are accessed less frequently.

- You are archiving data to S3 and you only need to keep this data for regulatory compliance purposes. You need this data in an archive for a specific period of time to cater for regulatory needs, and subsequently, this data can be deleted.
- You are making a backup of your databases on S3 and your organization has a predefined retention policy for this data. Based on the retention policy, you may want to keep the backup for a specific period, and then delete it.

Defining a life cycle policy for a bucket

Object life cycle rules can be configured using the Amazon S3 console, using the AWS SDK, or using the REST API. The following are the steps for configuring life cycle rules using the Amazon S3 console:

1. Sign in to your AWS account and go to the S3 console, at <https://console.aws.amazon.com/s3>.
2. Click on the bucket for which you want to create the life cycle policy.
3. Click on the **Management** tab, and then click on **+ Add lifecycle rule**, as shown in the following screenshot:

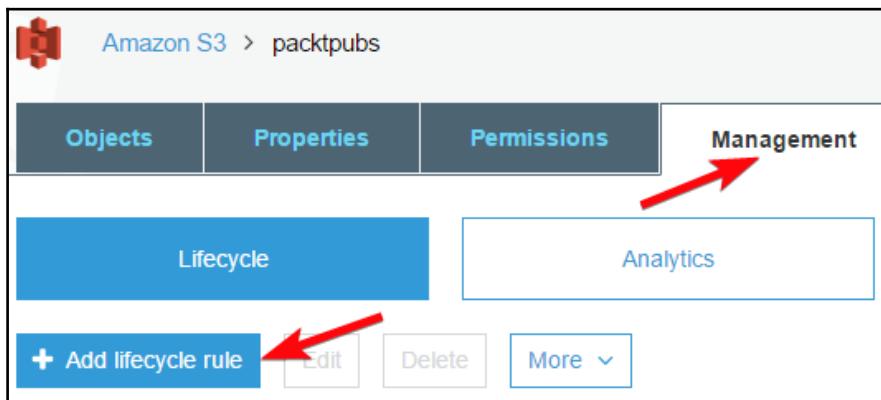


Figure 8.15: Adding a life cycle rule

4. In the subsequent window, enter the name for your rule, as shown in the following screenshot. The rule name must be unique in the bucket. You cannot create more than one rule with the same name on a bucket.

5. Specify the filter criteria for filtering the objects in a bucket. The filter criteria can be a string expression; for example, backup/. You can also specify one or more object tags and limit the scope of the rule accordingly, for example, backup/ | processed. You can select a prefix and tags while entering the filter value, as shown in the following screenshot. You can initially enter the backup/ prefix value, and then click on the tag, as shown in *Figure 8.17*, to enter the tag value. S3 uses the pipe (|) delimiter for separating the prefix and tag in the rule:

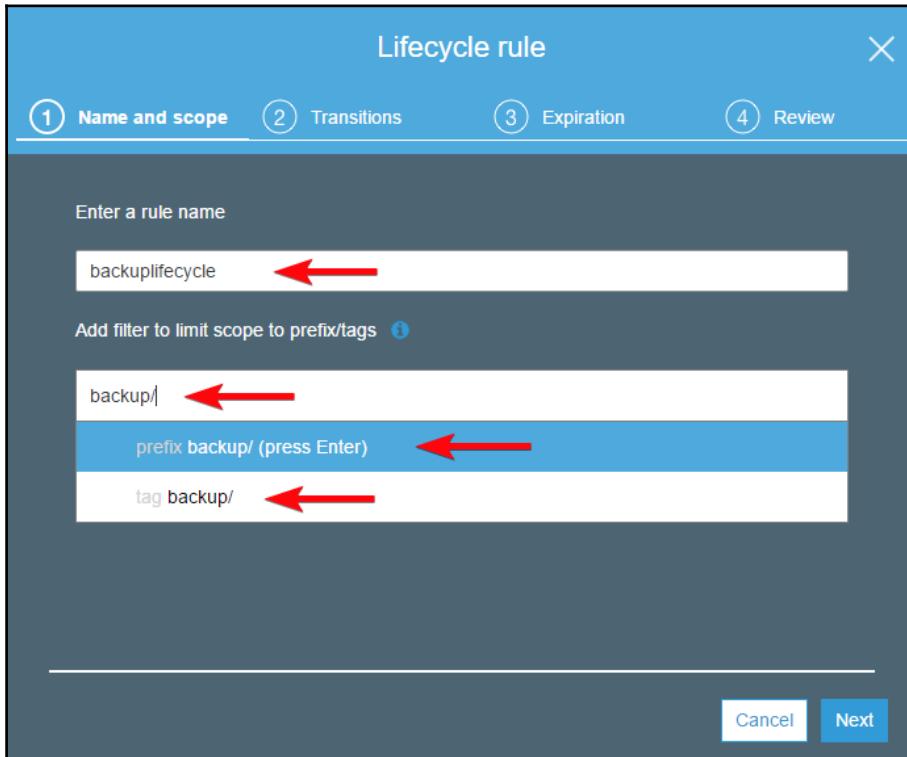


Figure 8.16: Enter the life cycle rule filter

6. On the next screen, you can define whether the rule you create applies to the current and latest version of the object, or the previous version. If versioning is enabled on the bucket, there may be more than one version of an object. Based on your preferences, you can select **Current version** or **Previous versions**, or both, as required:

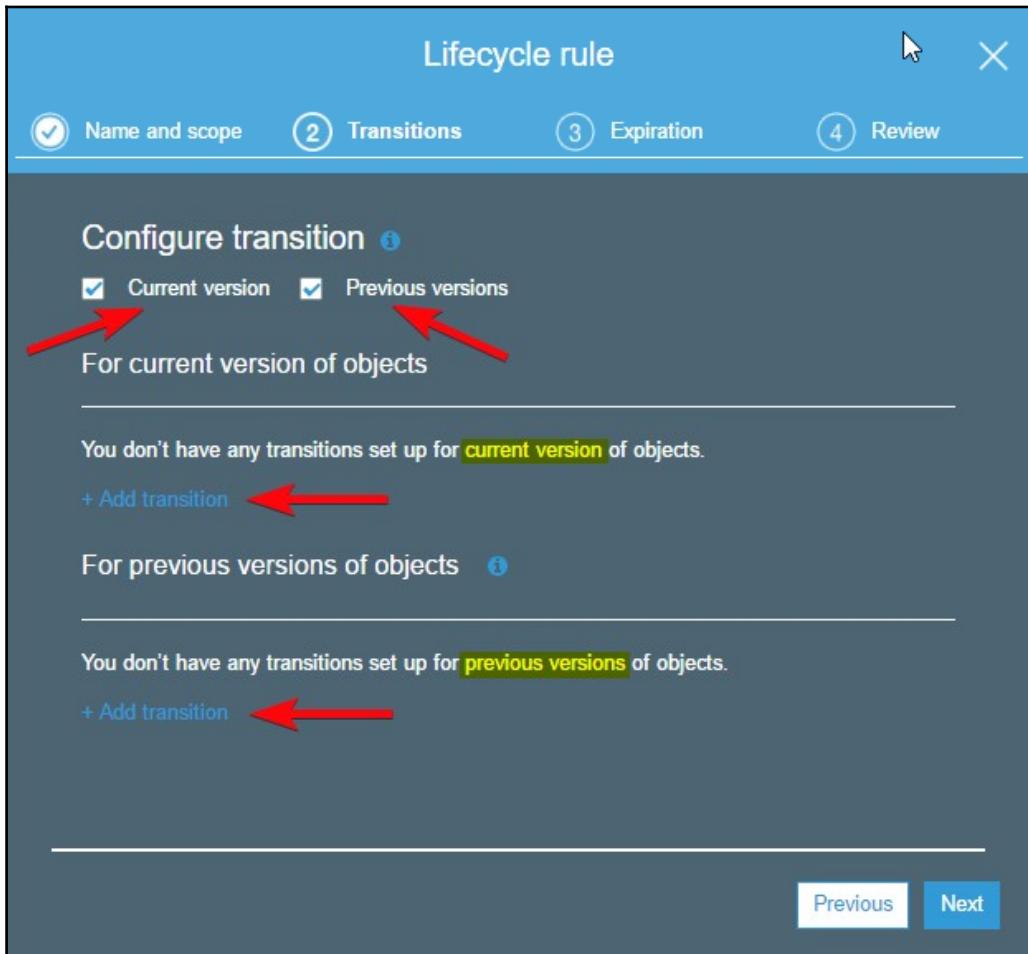


Figure 8.17: Select current or previous versions for applying a life cycle policy

7. Click on **+Add transition**, as shown in the preceding screenshot. It expands the options for selecting transition options, as shown in the following screenshot. Select the transition action from the combo box, either **Transition to Standard-IA after**, **Transition to One Zone-IA After**, or **Transition to Amazon Glacier after**. Also, enter the number of days after object creation when an object should transition, as shown in the following screenshot. You can also specify similar transition criteria for **Previous versions** of objects:

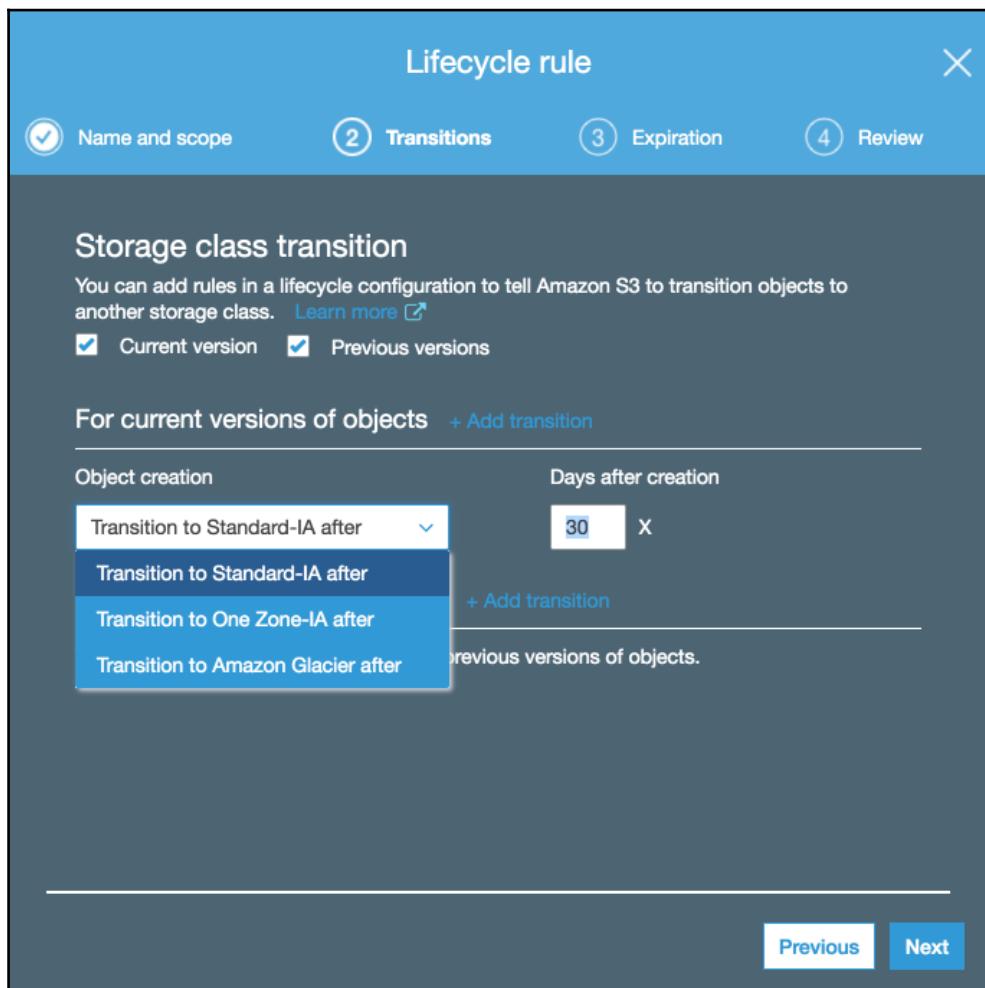


Figure 8.18: Object transition options for a life cycle policy

8. On the subsequent screen, configure the expiration options. Similar to the previous steps, you can select either **Current version** or **Previous versions**, or both of them, as required. You can additionally select to clean up expired object delete markers and clean up incomplete multipart uploads:

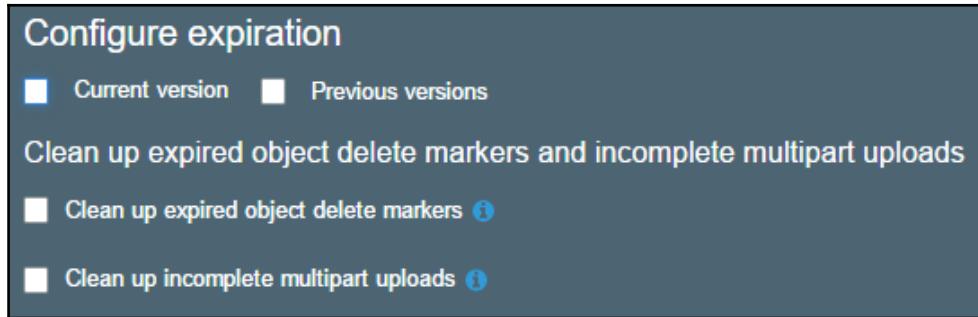


Figure 8.19: Configuring expiration options

Based on the version that's selected, you will get options for further selection. As shown in the following screenshot, you can choose to expire the current version after a specific number of days. You can also choose to **Clean up expired object delete markers**. Delete markers are not created for expired objects. If you choose to expire objects, you cannot select the option to clean up delete markers. Optionally, you can opt to **Clean up incomplete multipart uploads** after a specific number of days. This would be useful in a situation in which you uploaded a large object to S3, and the upload process was abruptly closed. S3 can automatically clean up such incomplete multipart uploads based on the selection here:

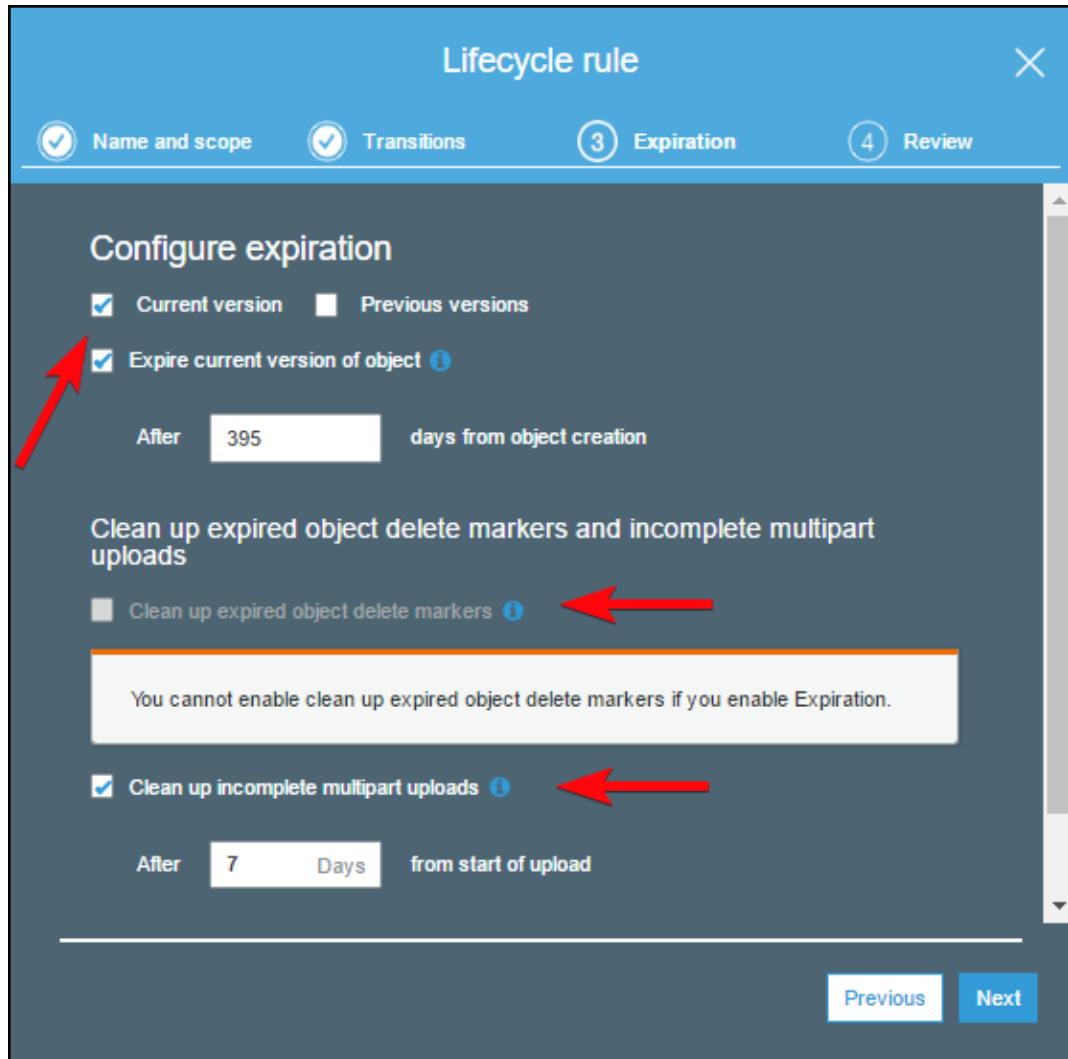


Figure 8.20: Providing additional data for expiration options

9. On the subsequent screen, review the **Lifecycle rule** and click on the **Save** button.

Hosting a static website on S3

Amazon S3 allows you to host a static website. A static website can contain web pages with static content, as well as client-side scripts. S3 does not support server-side scripting, and because of this, you cannot host a site with any server-side scripting, such as PHP, JSP, and ASP.NET. You can host HTML pages, CSS, client-side scripts such as JavaScript, and so on.

Here's a step-by-step process to enable static website hosting on an S3 bucket:

1. Sign in to your AWS console and go to the S3 console at <https://console.aws.amazon.com/s3>.
2. Click on the bucket in which you want to enable static website hosting.
3. Click on the **Properties** tab, as shown in the following screenshot:



Figure 8.21: Bucket properties tab

4. Click on **Static website hosting**, as shown in the following screenshot:

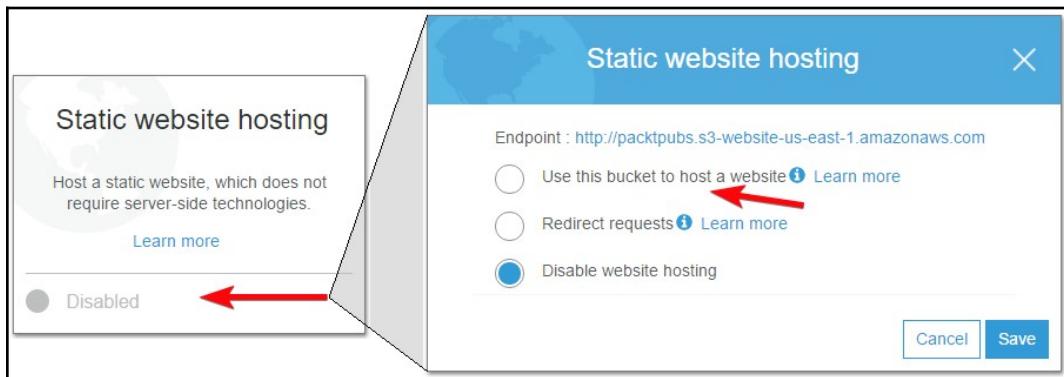


Figure 8.22: Enabling static website hosting

5. Specify index and error documents for your website, as shown in the following screenshot, and click on **Save**. You can also configure **Redirect requests** as needed, and you can optionally specify redirection rules. After configuring the options, you can browse your site from the endpoint URL of the bucket, as shown here:

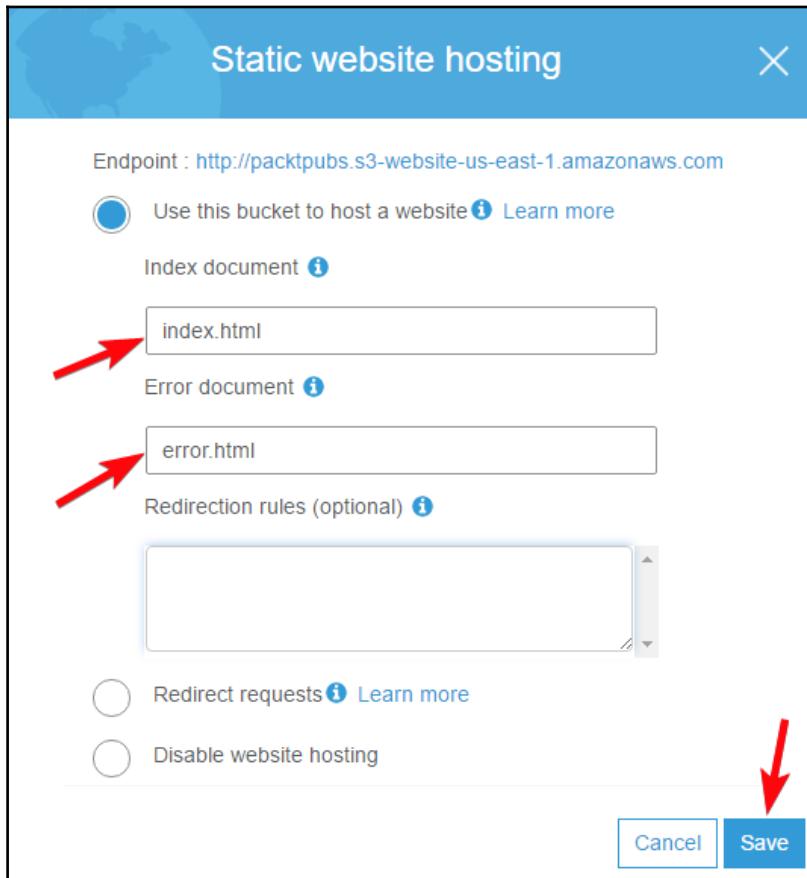


Figure 8.23: Specify index and error documents for static website

Using the preceding steps, you can host a static web site on S3. Sometimes, when you host your site data in more than one bucket, it may lead into some issues related to **cross-origin resource sharing (CORS)**. The next section will take you through this important topic.

Cross-origin resource sharing (CORS)

Before we look at CORS, let's look at the significance of the same origin policy. The cross-origin policy is a critical aspect of a web application security model. In a web application security model, by default, a web browser does not allow a script file associated with a web page to access data associated on a page in a different hostname, domain, or port number. The purpose of a cross-origin policy is to prevent any malicious script embedded on one page to access sensitive data on another web page.

For example, a script hosted in a `books.html` page on `www.packtpub.com` can access the **Document Object Model (DOM)** of any page within the same domain, that is, `www.packtpub.com`. If it tries to access the DOM of a page hosted on another domain, the access is denied. Even if a page is hosted on a subdomain, such as `books.packtpub.com`, when it tries to access the DOM of another page on `projects.packtpub.com`, it is denied the access. This is a way to maintain the security of the page based on the cross-origin web application security model.

CORS, as the name suggests, is the exact opposite of a cross-origin policy. It is a mechanism for client web applications hosted on one domain to use resources hosted on another domain. You can host rich client-side web applications using CORS support on S3. You can selectively enable CORS support on S3 by using the S3 console, the S3 REST API, and AWS SDKs.

Using CORS in different scenarios

The following are the use case scenarios wherein CORS can be used:

- **Use case 1:** Suppose that you are hosting a website on an S3 bucket named `packtpubs`. End users can access this site using `https://packtpubs.s3-website-us-east-1.amazonaws.com`. Amazon's S3 API endpoint for the bucket is assigned as `packtpubs.s3.amazonaws.com`. If you try to make authenticated GET and PUT JavaScript requests on the pages hosted in the bucket using the S3 API endpoint, these requests are blocked by a browser. You can allow such requests using CORS by explicitly allowing requests from `packtpubs.s3-website-use-east-1.amazonaws.com`.

- **Use case 2:** Consider a scenario in which you host a web site on a bucket and need to load fonts from a different bucket. In such a scenario, the browser denies access to the fonts bucket, as it refers to a different origin. CORS can help in such a scenario. You can explicitly allow cross-origin requests from the font bucket.

Configuring CORS on a bucket

For configuring CORS on a bucket, you need to create an XML document that defines the rule to allow cross-origin access on your bucket. You can either open full access to all domains, or open access for specific origin domains or URLs. To maintain the security of your site, it is recommended that you open access for specific domains. To further strengthen the security of the site, you can allow specific HTTP methods, such as GET, POST, PUT, and DELETE.

The following XML describes an example CORS configuration:

```
<!-- Sample policy -->
<CORSConfiguration>
  <CORSRule>
    <AllowedOrigin>*</AllowedOrigin>
    <AllowedMethod>GET</AllowedMethod>
    <MaxAgeSeconds>3000</MaxAgeSeconds>
    <AllowedHeader>Authorization</AllowedHeader>
  </CORSRule>
</CORSConfiguration>
```

The preceding policy is the default policy that you see when you enable CORS on a bucket. It allows GET requests from all origins. MaxAgeSeconds is the number of seconds a browser can cache a response from S3. AllowedHeader, by default, allows authorization requests. If you want to allow all headers, you can specify * in AllowedHeader. It is recommended that you exercise caution while configuring CORS and create one or more rules to allow specific domain and HTTP actions. The following example is more specific:

```
<!-- Sample policy -->
<CORSConfiguration>
  <CORSRule>
    <AllowedOrigin>http://www.packtpub.com</AllowedOrigin>
    <AllowedMethod>PUT</AllowedMethod>
    <AllowedMethod>POST</AllowedMethod>
    <AllowedMethod>DELETE</AllowedMethod>
    <AllowedHeader>*</AllowedHeader>
  </CORSRule>
```

```
</CORSRule>
<CORSRule>
<AllowedOrigin>*</AllowedOrigin>
<AllowedMethod>GET</AllowedMethod>
</CORSRule>

</CORSConfiguration>
```

There are two rules in the preceding example; the first rule allows PUT, POST, and DELETE actions from `http://www.packtpub.com`. The second rule allows GET requests from all origins with AllowedOrigin as *.

Enabling CORS on a bucket

The following steps describe the process of enabling CORS on a bucket:

1. Sign in to your AWS console and go to the S3 console, at `https://console.aws.amazon.com/s3`.
2. Click on the bucket in which you want to enable CORS.
3. Click on **Permissions | CORS configuration**, as shown in the following screenshot:

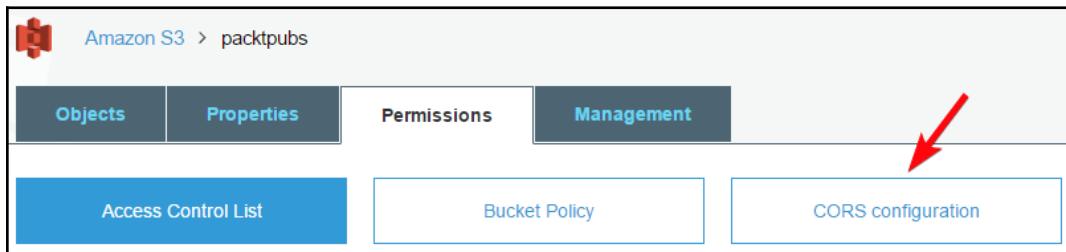


Figure 8.24: Click on CORS configuration

4. Edit the configuration XML as required, and click on **Save**, as shown in the following screenshot:

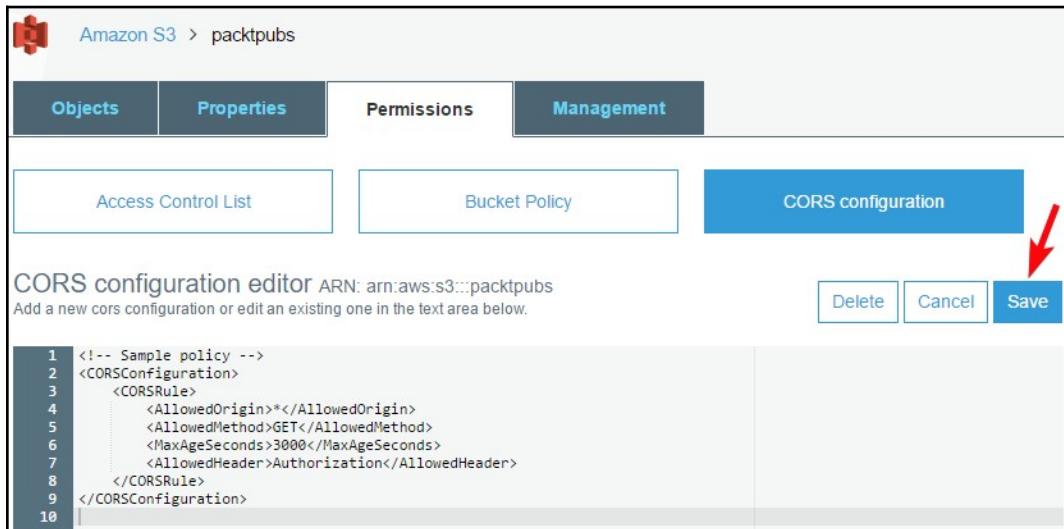


Figure 8.25: Edit the CORS configuration and save

Cross-region replication

Amazon S3 enables you to automatically and asynchronously copy objects from a bucket in one AWS region to another AWS region. This is a bucket-level feature, which can be configured on the source bucket. In the replication configuration, you can specify the destination bucket where you want your source bucket objects to be replicated. In the configuration, you can specify a key-name prefix. S3 replicates all the objects, starting with the specific key prefixes to destination bucket. Cross-region replication is generally used for compliance requirements, for minimizing latency in accessing objects, and for any operations in which compute resources in multiple regions need to access data from a region-specific bucket.

The following are some requirements for enabling cross-region replication:

- Both sources, as well as the destination bucket, must have versioning enabled on them.
- The source and destination buckets must be in different regions.
- S3 allows you to replicate objects from a source bucket to only one destination. You must provide permission to Amazon S3 to replicate objects from the source to destination bucket.
- If the source and destination bucket owners are different, the source bucket owner must have permission for `s3:GetObjectVersion` and `s3:GetObjectVersionACL` actions.
- If the source and destination buckets are in different AWS accounts, the source bucket owner must have access to replicate objects in the destination bucket.

Enabling cross-region replication

The following are the steps for enabling cross-region replication:

1. Sign in to your AWS console and go to the S3 console at <https://console.aws.amazon.com/s3>.
2. Click on the bucket in which you want to enable cross-region replication.
3. Click on the **Management | Replication | + Add rule** tab, as shown in the following screenshot:

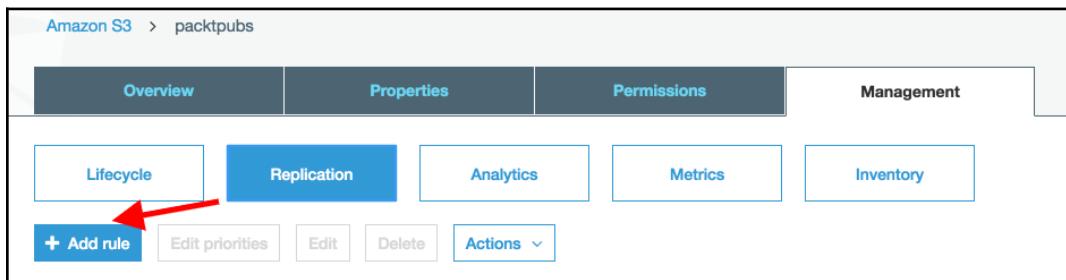


Figure 8.26: Enabling cross-region replication

4. As shown in the following screenshot, you can select a whole bucket or a specific key-name prefix in the bucket for source objects. In addition, you can enable encryption on the target where S3 objects are being replicated:

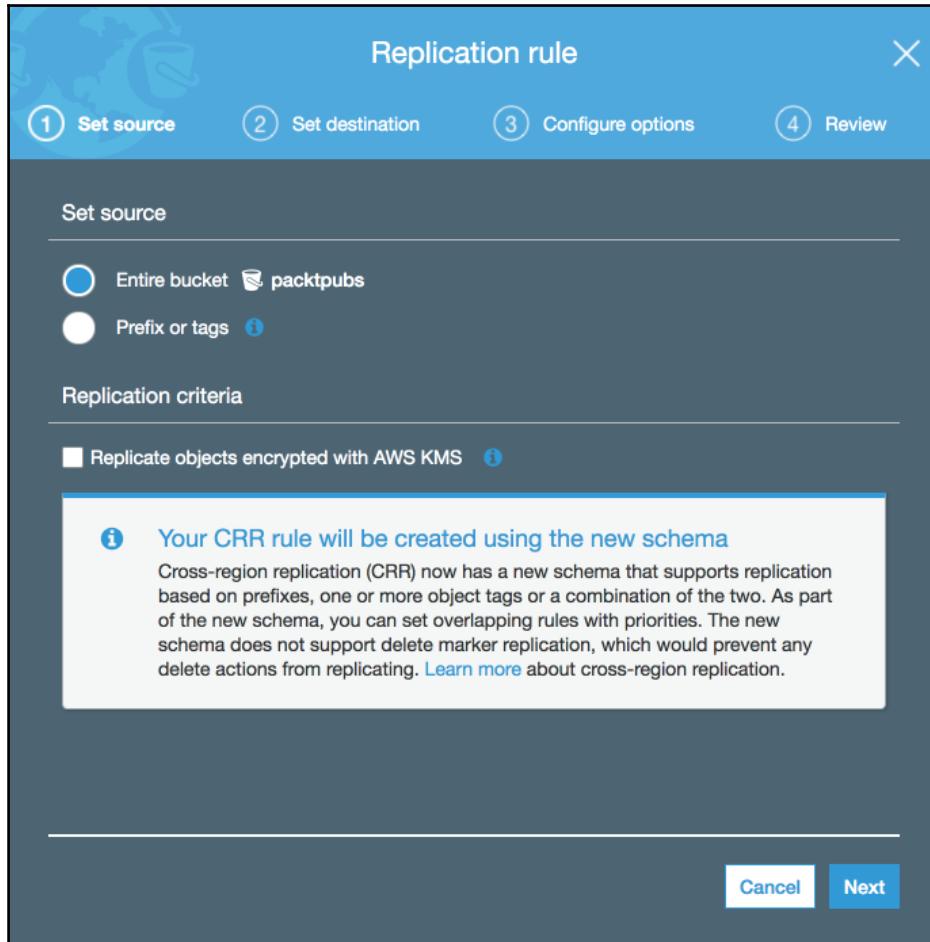


Figure 8.27: Configuring cross-region replication

5. Next, you can set the destination, as shown in the following screenshot:

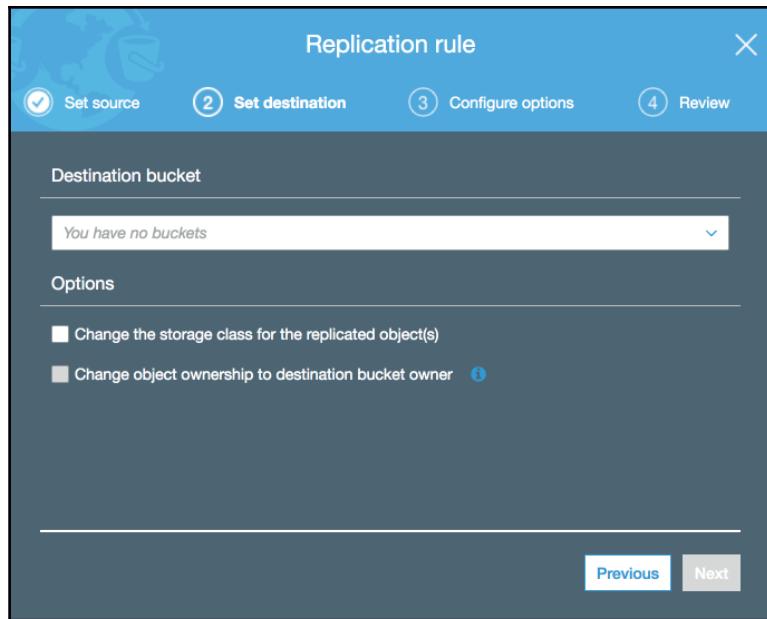


Figure 8.28: Selecting destination

When you select **Destination bucket**, a drop-down menu will appear, as shown in the following screenshot. At this stage, it also allows you to configure other options, such as changing the ownership and the storage class at the destination:

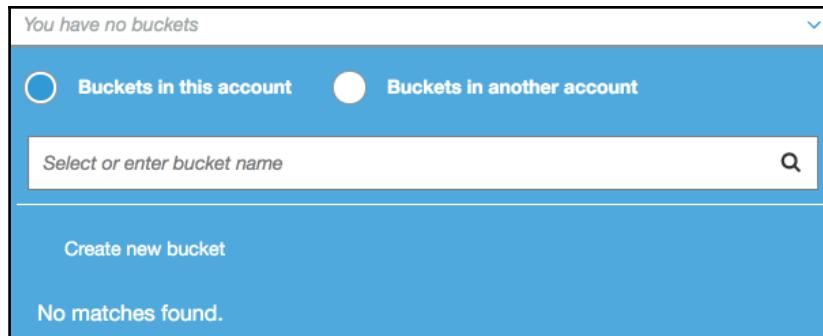


Figure 8.29: Destination bucket options

6. On the next screen, select the IAM service role, as AWS is performing a PUT action on our behalf and requires privileges to perform such an activity. Selecting the IAM role provides a drop-down menu. Selecting an existing IAM role will also provide you with an option to create a new IAM role. This will also provide you with an appropriate cross-region replication rule name:

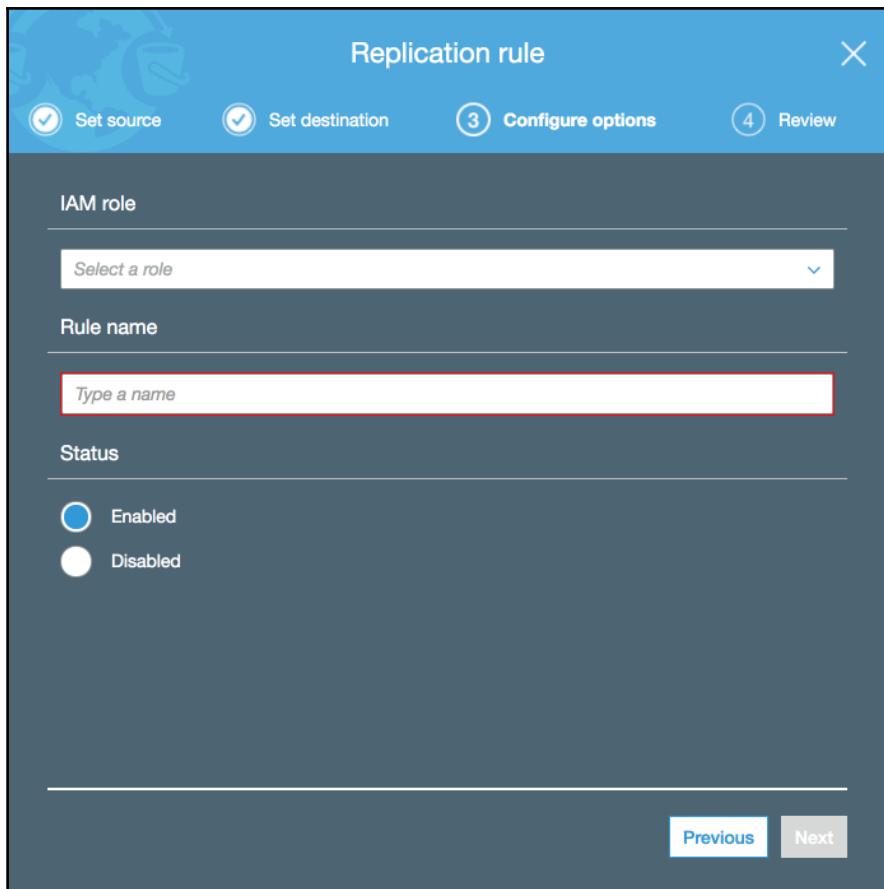


Figure 8.30: Configure options

7. Finally, review the settings, as shown in the following screenshot:

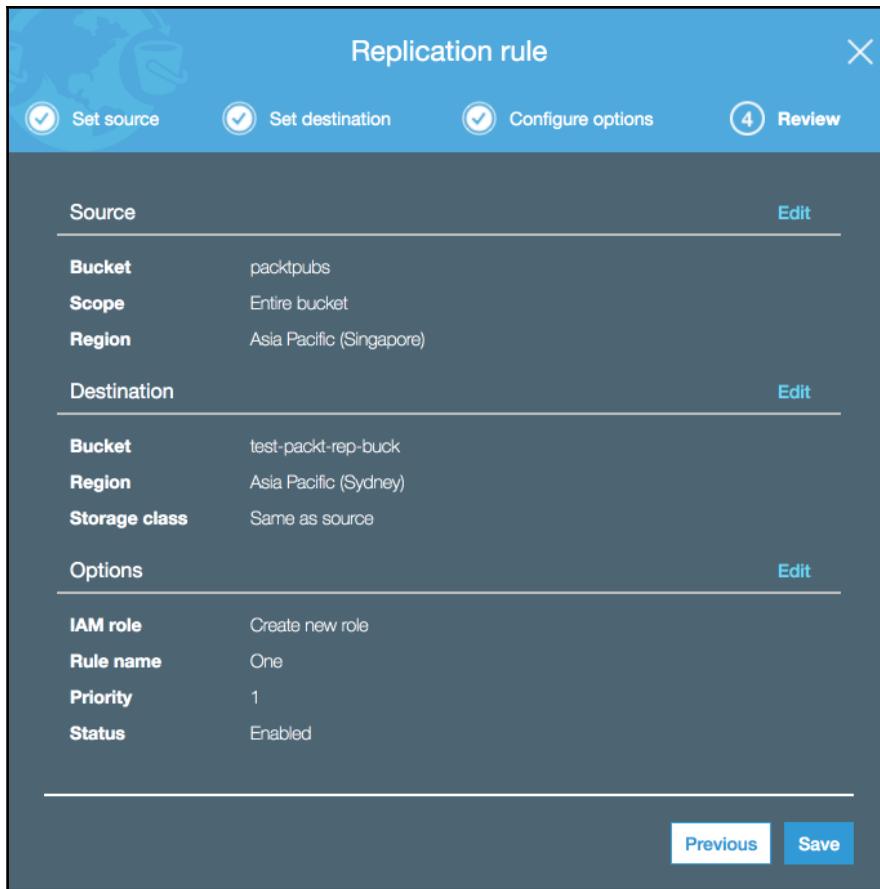


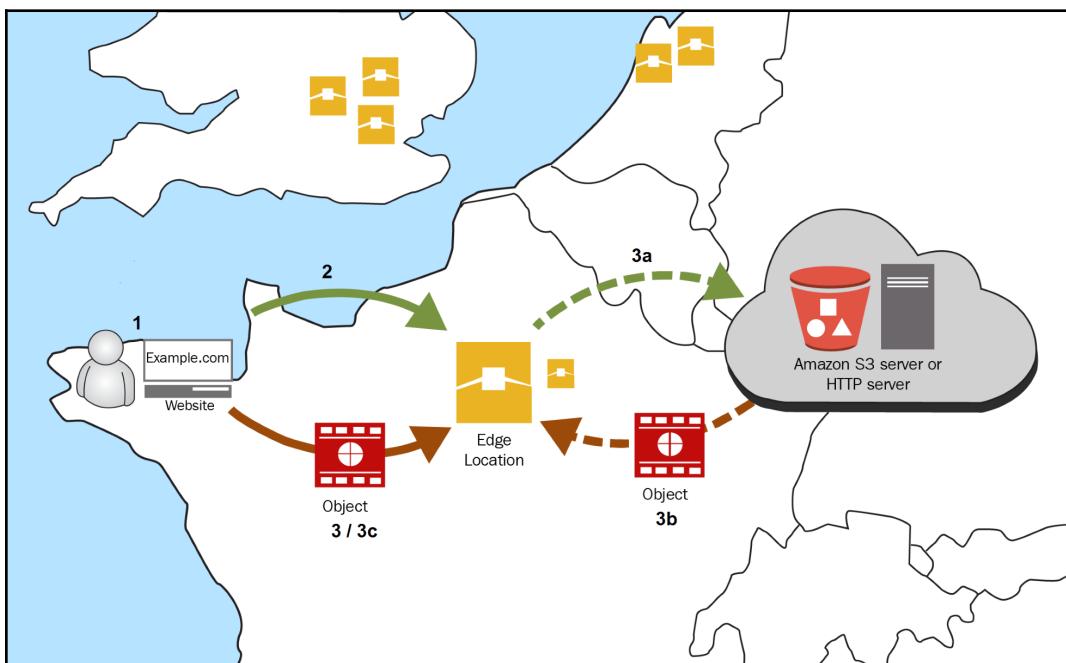
Figure 8.31: Review

To enable cross-region replication, it is essential to enable versioning on the bucket. If you haven't already enabled versioning on the bucket, do so, and follow the subsequent steps to enable cross-region replication.

CloudFront

Amazon CloudFront is a fast **content delivery network (CDN)** that aims to deliver image, PDF, audio/video files, live broadcasting, dynamic web content (.html, .css, .js), and APIs. Lambda@Edge is an extension of AWS Lambda that executes functions to customize the content delivered by CloudFront.

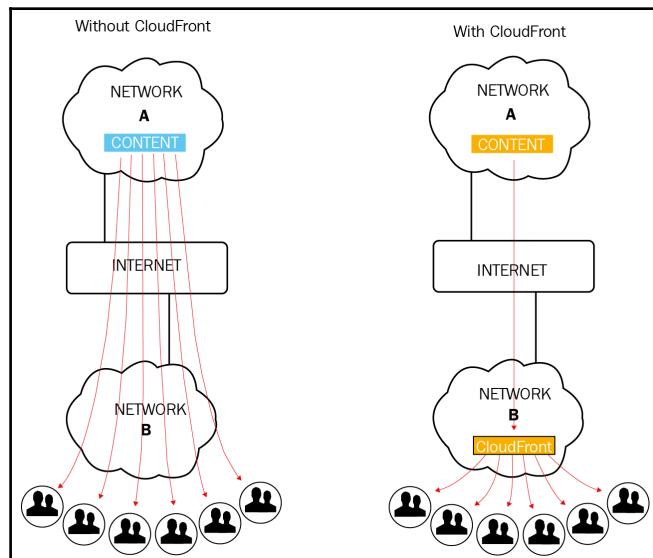
A CDN consists of a geographically distributed group of caching servers. Each geographical location situated servers are called an **Edge Location or Point of Presence (PoP)**. It reduces the load on an application origin by caching frequently used content. The aim is to minimize the latency and improve the speed. The following image may help you to understand how the CDN works at a high level, once it is configured:



The preceding diagram can be interpreted as follows:

1. An end user accesses the website or an application hosted on a Amazon S3, AWS EC2, or a custom origin. A custom origin allows on-premises servers to act as a point of origin. In other words, a point of origin can be Amazon S3, an AWS EC2 instance, or a custom origin (on-premises server), to serve a hosted website or an application.

2. With the help of global DNS, an end user is routed to the nearest CloudFront edge location that can serve the content to minimize the latency.
3. Once an end users' request for the content has landed at the nearest edge location, CloudFront will check its cache for the requested files. Once the files are found in the cache, they are directly served. If the files are not found in the cache, it will do the following:
 1. Amazon CloudFront will check the distribution configuration to identify the applicable origin server for the corresponding file type, and will forward the request to the origin to fetch the file(s). For example, distribution can be configured to fetch image files from an Amazon S3 bucket and HTML files from the web server.
 2. As soon as the origin server receives a request, it will respond with the files to the CloudFront edge locations.
 3. As soon as the CloudFront edge location receives the first byte of the requested file from the origin, it will forward the same to the end user. It will also keep these files in the cache. Next time, if it receives such a request, it will serve it immediately. The following diagram can help you to understand this scenario:



As shown in the preceding diagram, when CDN is not enabled, all the end user requests through global DNS are directly routed to the origin server. It may consume resources on the origin server, such as CPU, RAM, disk, or network I/O. As a result, resources on a origin requirement is directly related with the end user's request. It can be resolved by configuring autoscaling and ELB on the EC2 instance.

In the preceding diagram, the right-hand side of the image illustrates that when CDN is enabled only once, a request is routed to the origin server. The returned files for the requests are cached into the end location for a specified **Time To Live (TTL)**, and within that time, when other end users request the same files or content, they are directly served from the edge location. It helps to minimize the load and resource requirement on the origin. Enabling Amazon CloudFront CDN helps to minimize AWS billing and latency.



It is important to remember that these CloudFront edge locations will cache this content for a specified time. This specified time is called the TTL. In any case, if the content at the origin is changed and the website is still delivering old content, you may need to invalidate the old content from the cache to delete the old cache from the edge locations. Alternatively, use file versioning to remove files from the cache right away.

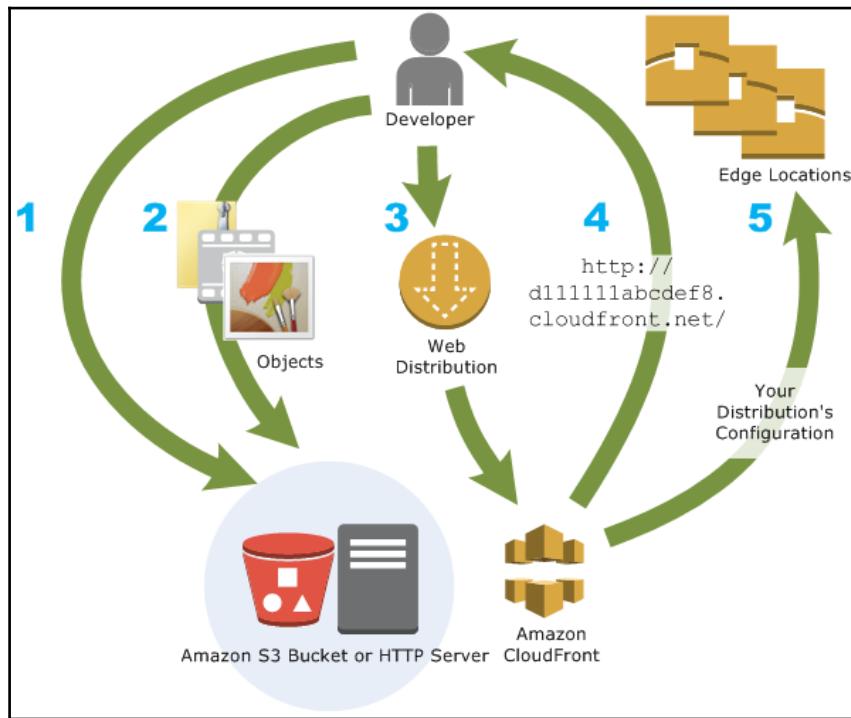
CloudFront regional edge caches

A **regional edge cache** is another type of edge location with a higher cache than any edge location. These locations sit between origin web servers and other edge locations. In general, in edge locations, as the popularity of the content (files) reduces, they evict such content to store popular content. As content may get evict from edge locations but as regional edge caches have higher cache than any individual edge location, object remains in the cache for the longer time at the nearest regional edge cache location.

As a result, when an end user requests any content, first, it checks the edge location. If it is not found in the edge location, before going to the origin server, it will check for content availability in the regional edge cache. If it is also not found in the regional edge cache, it will fetch content from the origin, and as soon as the first byte arrives to the regional edge location, it will forward it to the end user and it will also be stored in the edge location.

Setting up CloudFront content and delivery

Creating a CloudFront distribution is a mechanism to configure CloudFront about a content to be delivered, and how to track and manage content delivery. The following diagram should help you to understand this at a high level:



The preceding diagram can be explained as follows:

1. The **Developer**, or administrator, specifies the origin servers. For example, the origin server could be an Amazon S3 bucket, an Amazon EC2 instance, or an on-premises server. The origin helps CloudFront to obtain content (that is, files) to be distributed on edge locations around the world. Amazon CloudFront also supports Adobe Media Server RTMP protocol to distribute media files on demand from an Amazon S3 bucket.
2. All the contents (that is, files) are uploaded to the origin servers. The files could be a web page, an image, or media files. These are called **objects**. When these objects are stored in an Amazon S3 bucket, make the bucket public, so that anyone who knows the CloudFront URL can access the bucket.

3. At this step, create a CloudFront distribution. It specifies the origin server where all the contents are stored and end user's request from where to obtain. It also specifies where you would like to store CloudFront logs.
4. CloudFront assigns a domain name for the newly created distribution. Usually, it uses a `cloudfront.net` domain. Optionally, it is also possible to use a custom domain name, such as `example.com`.
5. CloudFront will distribute the configuration to all the edge locations, but it will not distribute contents stored in the origin server. The content will only be stored to edge locations when end users request it.

Some CloudFront use cases are as follows:

- To speed up the delivery of static website content, such as images, style sheets, JavaScript, media files, and documents, this content can be stored in the Amazon S3 bucket, and CloudFront can point to the S3 bucket as the origin. Optionally, it also makes it possible to restrict direct access to S3 bucket files (that is, objects) and can only be accessed from a signed CloudFront URL or cookies by enabling an **Origin Access Identity (OAI)**.
- For on-demand or live-streaming video, it supports common formats such as MPEG, Apple HLS, DASH, Microsoft Smooth Streaming, and CMAF. Enabling CloudFront distribution on live-streaming can eliminate the load on the origin server. It minimizes the latency and overall infrastructure expenses.
- A CloudFront distribution can be configured with HTTPS to make end to end secured communication between origin server and end user. At the same time, it is possible to encrypt certain fields to add an additional layer of security, so only certain applications at the origin can see the data. Field encryption can be enabled by simply specifying a set of fields while adding a public key to the CloudFront distribution.
- Customize the content at the edge by running serverless code (that is, a Lambda function). This is also called Lambda@Edge, and it has endless possibilities. For example, it can be used to display an error page when content is not available on the origin server. Alternatively, it can be used to distribute private content from signed URLs or signed cookies.

Summary

- There are three broad types of storage services: block storage, file storage, and object storage.
- Block storage is a type of storage that may not be physically attached to a server, but is accessed as a local storage device, just like a hard disk drive.
- File storage is also known as file-based storage. It is a highly available, centralized place for storing your files and folders.
- Object storage is a type of storage architecture where the data is stored as objects. Each object consists of the data, metadata, and a globally unique identifier.
- Block storage works well for creating filesystems and installing operating systems and databases.
- Unlike in block storage, you do not have access to format the file storage, create a filesystem, and install an operating system on it.
- Unlike in block storage, you do not have access to format the object storage, create a filesystem, and install an operating system on it.
- S3 is a cloud-based object storage service from Amazon.
- S3 is highly scalable and makes it easy to access storage over the internet.
- Currently, the permissible object size in S3 is from 0 bytes to 5 TB.
- A bucket is a logical unit in S3, just like a folder. It is a container in which you can store objects, and also folders.
- Buckets are created at the root level in S3 with a globally unique name.
- The key is a name that is assigned to an object.
- Amazon provides two types of consistency models for S3 data when you perform various input/output operations with it: read-after-write consistency and eventual consistency.
- Bucket ownership cannot be transferred to another AWS account or another user within the same AWS account
- There is no limit to the number of objects that can be created in a bucket.
- S3 Transfer Acceleration provides a fast, easy, and secure way to transfer files between S3 and any other source or target of such data transfers.
- Amazon provides an option with which you can configure your bucket as a Requester Pays bucket.
- S3 allows you to keep multiple versions of an object in a bucket.

- When you overwrite or delete an object in an S3 bucket where versioning is enabled, it keeps multiple copies of the object with version numbers.
- Amazon S3 provides a number of storage classes for different storage needs: S3 Standard storage, S3 IA, S3 One Zone-IA storage, S3 RRS, S3 Intelligent-Tiering, and Glacier.
- S3 Standard storage is used as general-purpose storage for frequently accessed data.
- S3-IA storage is meant for data that is less frequently used, but needs to be available immediately when it's needed.
- Amazon S3 One Zone-IA only stores data in one AZ. It is best suited to storing infrequently accessed and reproducible data.
- S3 RRS provides reduced levels of redundancy, as opposed to S3 Standard storage. It is suitable for storing non-critical and reproducible data.
- S3 provides an Intelligent-Tiering storage class that can automatically optimize your storage cost by segregating your data to the most cost-effective S3 storage class, based on its usage pattern.
- Glacier is a very low-cost, secure, and durable data archival storage.
- Glacier is ideal for storing long-term data, backups, archives, and data for disaster recovery.
- Life cycle management is a mechanism in S3 that enables you to either automatically transition an object from one storage class to another storage class, or automatically delete an object, based on the configuration.
- Amazon S3 allows you to host a static website. A static website can contain web pages with static content, as well as client-side scripts.
- The purpose of a cross-origin policy is to prevent any malicious script embedded on one page to access sensitive data on another web page.
- For configuring CORS on a bucket, you need to create an XML document that defines the rule to allow cross-origin access on your bucket.
- Amazon S3 enables you to automatically and asynchronously copy objects from a bucket in one AWS region to another AWS region. This is called cross-region replication.
- Amazon CloudFront is a fast CDN that aims to deliver images, PDFs, audio/video files, live broadcasting, dynamic web content (.html, .css, and .js), and APIs.

9

Other AWS Storage Options

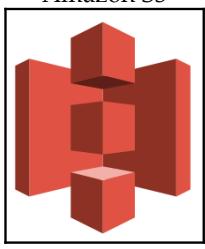
AWS offers a variety of highly available, scalable, reliable, and secure storage services to address various organizational needs. It provides a rich web console for all of the services that is easy to access and navigate. Its easy-to-use UI complements efficient services for quickly performing day-to-day administrative tasks. AWS also provides a set of API and CLI interfaces. You can use APIs and CLIs to perform advanced operations or to automate various tasks using customized applications and scripts.

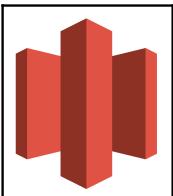
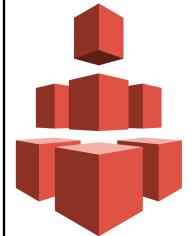
The following topics are covered in the chapter:

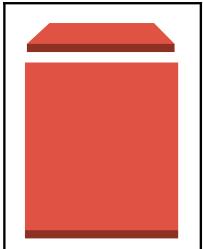
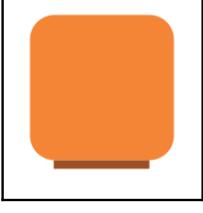
- Storage and backup services provided by AWS
- **Amazon Elastic File System (EFS)**
- AWS Storage Gateway
- AWS Snowball
- AWS Snowmobile

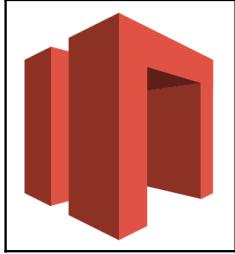
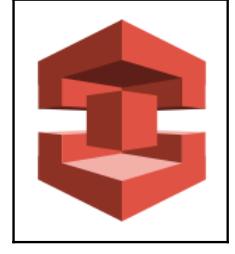
Storage and backup services provided by AWS

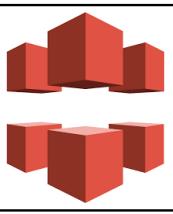
The following table describes a number of storage and backup services provided by AWS:

AWS services	Description
 Amazon S3	<p>S3 is a cloud-based object storage service that can be used over the internet. It is suggested for storing static content, such as graphics files, documents, log files, audio, video, and compressed files. Virtually any type of data in any file format can be stored on S3. Currently, the permissible object size in S3 is 0 bytes to 5 TB. Objects in S3 are stored in a bucket. A bucket is a logical unit in S3 that is just like a folder. Buckets are created at root level in S3 with a globally unique name. You can store objects and also folders inside a bucket. Any number of objects can be stored in each bucket. There is a soft limit of 100 buckets per account in S3.</p> <p>Common usage: S3 can be used for content storage and distribution, static website hosting, big data object store, backup and archival, storing application data, as well as for Disaster Recovery (DR). Using JavaScript SDK and DynamoDB, you can also host dynamic applications in S3.</p>

<p>Amazon Glacier</p> 	<p>Glacier is a highly secure, durable, and very low-cost cloud storage service for archiving data and taking long-term backups. Each file or object stored in Amazon Glacier is called an archive. These stored archives are immutable, which means that contents of the archive cannot be modified. If required, another version of the archive can be stored and the existing version can be deleted. The size of each archive can range from 1 byte to 40 TB. With the help of the S3 lifecycle rules, objects from S3 can be automatically transferred to Glacier. These archives can be logically isolated in containers called vaults. A maximum of 1,000 vaults per account per region can be created.</p> <p>The following are a few important characteristics of Glacier:</p> <ul style="list-style-type: none">• It is very economical for storing long-term archival data, which is rarely accessed.• Retrieval incurs charges and may take a minimum of three to four hours or more depending on the size of the data.• Amazon charges early deletion fees if data is deleted within three months from the date of storing. <p>Common usage: Glacier can be mainly used for data archival. It is widely used for media asset archiving, healthcare information archiving, regulatory and compliance archiving, scientific data storage, digital preservation, magnetic tape replacement, and more. It is rarely retrieved for audit or other business purposes.</p>
<p>Amazon EFS</p> 	<p>AWS Elastic File System (EFS) is a simple-to-use and scalable file storage service that can be used with EC2 instances. It is a fully managed storage service from AWS that can be used for storing GBs to TBs of data. EFS volumes can be mounted and accessed by multiple EC2 instances at the same time. It uses the Network File System Version 4.1 (NFSv4.1) protocol. When using any EFS volume for the first time, you simply need to mount and format it to the desired filesystem. Subsequently, you can mount this volume on other EC2 instances directly and start using it. EFS volumes can also be accessed from on-premises environments using Direct Connect. You cannot access it from an on-premises environment over VPN connectivity. EFS is available in two modes—General Purpose mode and Max I/O mode.</p> <p>Common usage: EFS is designed to provide very high disk throughput. It can be used for big data and analytics, media, content management, web serving, and home directories.</p>

<p>Amazon EBS</p> 	<p>EBS is a persistent, block-level storage service from Amazon. Persistent storage is a type of storage that retains the data stored on it even after power to the device is turned off. Block-level storage is a type of storage that can be formatted to support a specific filesystem, such as NFS, NTFS, SMB, or VMFS. EBS volumes can be attached to an EC2 instance. Because of its persistent nature, data on an EBS volume remains intact even after restarting or stopping an EC2 instance.</p> <p>There are five variants of EBS:</p> <ul style="list-style-type: none">• General Purpose SSD (gp2)• Provisioned IOPS SSD (io1)• Throughput optimized HDD (st1)• Cold HDD (sc1)• Magnetic (standard) <p>Each of these variants differs in terms of price and performance. EBS volumes are connected as a network storage to an EC2 instance. It can be sized from 1 GB to 16 TB. You can take a snapshot of an EBS volume. A snapshot is a point-in-time backup of an EBS volume. Snapshots can be used to restore the volume as and when required.</p> <p>Common usage: EBS volumes can be used as a root partition and for installing operating systems. It is also used for storing enterprise applications, application data, and databases.</p>
<p>Amazon EC2 instance store</p> 	<p>Instance store is a temporary block-level storage service from Amazon. Unlike EBS, an instance store is temporary in nature. Data stored in an instance store volume is deleted when the EC2 instance is either restarted, stopped, or terminated. Instance store volumes are directly attached to the underlying hosts where an EC2 instance is provisioned. Instance store volumes are faster than EBS; however, it is a temporary data store. The performance of the instance store volume attached to an EC2 instance, the size of each of the volumes, and the number of such volumes that can be attached to an EC2 instance, depend on the EC2 instance type.</p> <p>Common usage: This is widely used to store swap files, temporary files, or in applications where good disk throughput is required but data persistence is not required.</p>

<p>AWS Storage Gateway</p> 	<p>AWS Storage Gateway is a hybrid storage service that connects on-premises environments to cloud storage using a software appliance. It seamlessly connects on-premises environments with Amazon's block-level and object-level storage services such as EBS, S3, and Glacier. Storage Gateway uses standard storage protocols such as NFS and iSCSI. It provides low-latency for exchanging data from on-premise to S3, Glacier, or EBS volumes, and vice versa. Storage Gateway can provide high performance for frequently accessed data by caching it at source in on-premise environments.</p> <p>Common usage: Storage Gateway can be configured for use as a file server in conjunction with S3. It can also be used as a virtual tape library for backup on S3 and virtual tape shelf for archival on Glacier. It can also be configured to be used as a local iSCSI volume. Storage Gateway can also be handy for transferring data from on-premise environments to AWS or transferring the data from AWS to on-premises environments.</p>
<p>AWS Snowball</p> 	<p>AWS Snowball is a petabyte-scale data transport solution that uses physical appliances to transfer large-scale data from on-premises environments to the AWS cloud and vice versa. A single Snowball appliance can transport up to 80 TB of data. Snowball comes in two sizes: 50 TB and 80 TB. Data can be copied over to multiple physical appliances and transported to and from an AWS. Transferring large-scale data over the internet can take a significant amount of time, depending on the size of the data. The purpose of the Snowball service is to minimize the data transfer time by transferring the data using a physical medium rather than transferring data over the internet. Snowball can efficiently compress, encrypt, and transfer data from the on-premises host to the intended Snowball device. Once the data is copied over to one or more snowball devices, these devices are transported back to the nearest AWS data center. Subsequently, AWS transfers data from Snowball devices to S3.</p> <p>Common usage: Snowball is used for rapidly and securely transferring bulk data between on-premises data centers and the AWS cloud at a very economical rate.</p>

<p>AWS Snowmobile</p> 	<p>AWS Snowmobile is an exabyte-scale data transport solution that uses physical containers to transfer extremely large-scale data from on-premises environments to the AWS cloud and vice versa. A Snowmobile container literally comes in a truck that can transfer up to 100 PB of data per Snowmobile. The truck carries a high cube shipping container that is 45 feet long, 8 feet wide, and 9.6 feet tall. If your data is more than 100 PB, you can ask Amazon for more than one Snowmobile. At a time, more than one Snowmobile can be connected to the on-premises network for transferring data. When connected to an on-premises network, the Snowmobile appears as a standard NFS mounting point on the network. It may require up to 350 KW of power. Once the data is transferred from the on-premises network to the Snowmobile, it returns to the nearest AWS data center in the region and, subsequently, the data is transferred to the S3 of the respective customer account.</p> <p>Common usage: Snowmobile is used for rapidly and securely transferring extremely large-scale data between the on-premises data center and the AWS cloud at a very economical rate.</p>
<p>Amazon CloudFront</p> 	<p>Amazon CloudFront is a Content Delivery Network (CDN) offered by AWS. It is a system of distributed servers spread across edge locations. It is mainly used for caching static content, such as web pages, style sheets, client-side scripts, and images. It can also speed-up dynamic content distribution. When a user hits a URL that is served through CloudFront, it routes the user request to the nearest edge location. The nearest edge location gives minimum latency in serving the request and provides the best possible performance to the user.</p> <p>Common usage: CloudFront is used for providing seamless performance on the delivery of a website or web application for a user base spread across multiple geographic locations. It can be used for distributing software or other large files, streaming media files, offering large downloads, and delivering live events.</p>

S3, Glacier, EBS, EC2 instance store, and CloudFront are explored in other relevant chapters. The subsequent sections of this chapter touch upon EFS, AWS Storage Gateway, AWS Snowball, and AWS Snowmobile.

Amazon EFS

AWS EFS is a simple-to-use and scalable file storage service that can be used with EC2 instances. It is a fully-managed storage service from AWS that can be used for storing GBs to TBs of data. EFS volumes can be mounted and accessed by multiple EC2 instances at the same time. It uses the NFSv4.1 protocol. When using any EFS volume for the first time, you simply need to mount and format it to the desired filesystem. Subsequently, you can mount this volume on other EC2 instances directly and start using it. EFS volumes can also be accessed from an on-premises environment using Direct Connect. You cannot access it from an on-premises environment over VPN connectivity. EFS is available in two modes—**General Purpose mode** and **Max I/O mode**.

In the industry, it is a common requirement to share filesystems across the network, which can be used as a common data source. EFS is a simple, secure, fully managed, scalable, and reliable block storage to fulfil common file storage requirements. To use EFS with a Linux EC2 instance, you may need to install the latest NFS packages. AWS recommends using the NFSv4.1 client on EC2 instances. Unlike EBS, EFS does not require the provision of a fixed volume size in advance. Being a managed service, you can store as much data as you need and pay only for what you use.



Currently, EFS does not support Windows-based EC2 instances.

An EFS volume is created at the VPC level. At the time of creating an EFS volume, you can specify the AZ from where it can be accessed. EC2 instances from all selected AZs within the same VPC can access the EFS volume. Optionally, you can add tags to your EFS volume. It is recommended to provide a relevant and meaningful name to your EFS volume along with tags for better identification and reference. While creating an EFS volume, it is essential to select the type of EFS volume. Types of EFS volume are General Purpose and Max I/O. The default EFS volume type is General Purpose. Once an EFS volume is successfully created, it returns a DNS endpoint. You can mount the EFS volumes on an EC2 instance or an on-premises environment using the endpoint. Remember, you can mount EFS volumes on an on-premises network only if you use Direct Connect.

Successful creation of an EFS volume also creates mount points in each AZ. EFS carries properties such as mount target ID, the filesystem ID, private IPv4 address, the subnet ID in which it is created, and the mount target status. It is possible to mount EFS volumes using a DNS name. The next diagram explains EFS in more detail:

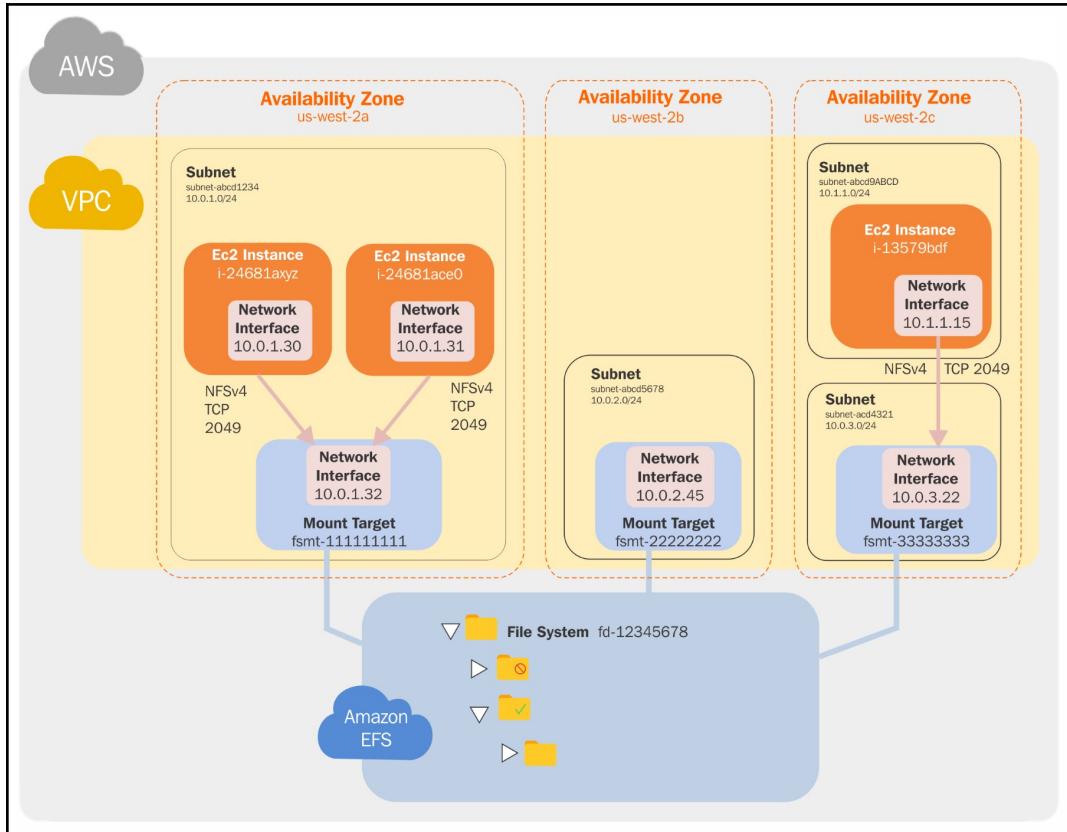


Figure 9.1: EFS

An EC2 instance does not require a public or elastic IP to mount an EFS volume. You can enable or disable any existing EFS volume as required. You can perform all such changes from the **Manage file system access** option. Once you delete an EFS volume, it cannot be recovered.

Snapshots can be created for EFS volumes. It is also possible to design a backup solution using **AWS Data Pipeline** for copying data from one EFS volume to another EFS volume. You can also configure a copy operation schedule.

AWS Storage Gateway

AWS Storage Gateway is a hybrid storage service provided by Amazon. With Storage Gateway services, your on-premises applications can seamlessly use AWS cloud storage. The following are some of the important points of AWS Storage Gateway:

- AWS Storage Gateway connects on-premises software appliances with the AWS cloud storage to provide a seamless integration experience and data security between the on-premises data center and the AWS storage services.
- It is a scalable and cost-effective storage solution that also maintains data security.
- It provides an iSCSI interface, which can be used to mount a volume as a local drive to easily integrate it with the existing backup applications.
- AWS Storage Gateway uses incremental EBS snapshots for backing up data to AWS.
- AWS provides a VM image for running Storage Gateway on an on-premises data center; you can also run it as an EC2 instance on AWS, and, in case of any issues, such as if the on-premises data center goes offline, you can deploy the gateway on an EC2 instance.
- You can use a Storage Gateway hosted on an EC2 instance for DR, data mirroring, and as an application storage.
- By default, Storage Gateway uploads data using SSL and provides data encryption at rest using AES-256 when the data is stored on S3 or Glacier.
- Storage Gateway compresses data in-transit and at-rest for minimizing the data size.

AWS Storage Gateway provides three types of solutions—**file gateways**, **volume gateways**, and **tape-based gateways**. A file gateway creates a file interface into Amazon S3. It allows you to access S3 using the **Network File System (NFS)** protocol. When using volume gateways, you can mount a volume as a drive in your environment. Tape-based gateways can be used similarly to a tape drive for backup.

File gateways

A file gateway creates a file interface in Amazon S3. It allows you to access S3 using the NFS protocol. When you opt for a file gateway, a software appliance is hosted in the on-premises environment on a virtual machine running on VMware ESXi. Once the file gateway is created, it enables you to directly access S3 objects as files using an NFS volume mounted on a server, as shown in the following diagram:

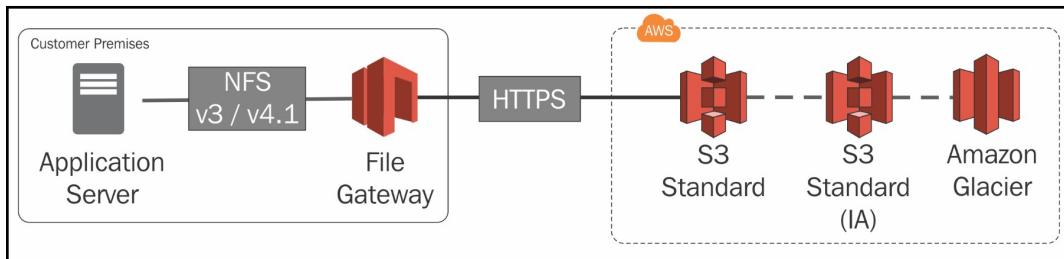


Figure 9.2: File gateway

Here is what file gateways can do for you:

- They allow you to directly store files on S3 using the NFS 3 or NFS 4.1 protocol.
- You can directly retrieve files from S3 using the same NFS mount point.
- They also allow you to manage S3 data with life cycle policies, manage cross-region replication, and enable versioning on your data.

Volume gateways

When you create a volume gateway, it creates a cloud-backed storage volume that you can mount as an iSCSI device on your on-premises servers, where iSCSI stands for **Internet Small Computer System Interface**. Volume gateways store all data securely on AWS. There are two types of volume gateway, which determine how much data is stored on-premises and how much data is stored on AWS storage, and are discussed in the following subsections.

Gateway-cached volumes

Cached volumes enable you to store complete data on S3 and cache a copy of only frequently used data on-premises. By reducing the amount of data stored on on-premises environments, you can reduce the overall storage cost. It also boosts performance by providing low-latency access to frequently accessed data using a cache:

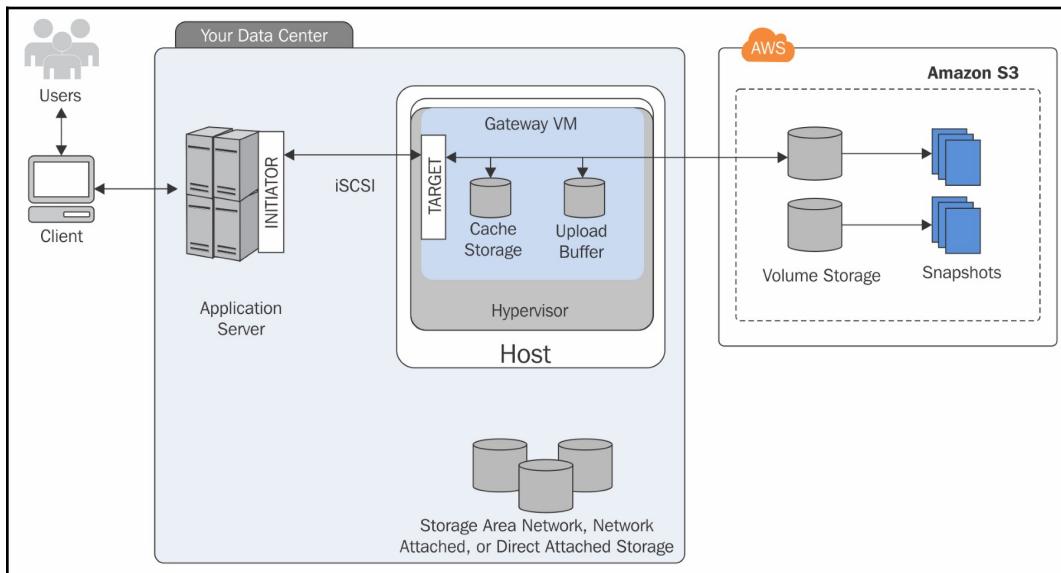


Figure 9.3: Gateway-cached volumes

The key features of gateway-cached volumes are as follows:

- A cached volume stores data in S3 and serves as primary data storage.
- It creates a local copy of frequently accessed data, which provides low-latency access for subsequent data access requests from applications.
- By reducing the amount of data stored on an on-premise environment, you can reduce the overall storage cost.
- You can create up to 32 gateway-cached volumes in a single storage gateway. You can store from 1 GB to 32 TB in each volume with a maximum storage volume limit of 1,024 TB (1 PB).

- You can attach gateway-cached volumes as iSCSI devices on on-premises application servers.
- You can take incremental snapshots of gateway-cached volumes.
- Gateway-cached volume snapshots are stored on S3 as EBS snapshots.
- Gateway-cached volume snapshots can be restored as gateway storage volumes, or you can create an EBS volume out of them and use it on an EC2 instance.
- The maximum size of an EBS volume created out of a snapshot is 16 TB and you cannot create an EBS volume out of the snapshot if it is more than 16 TB in size.
- AWS stores gateway-cached volume data and snapshots in Amazon S3 and the data is encrypted at rest with **Server-Side Encryption (SSE)**; you cannot access the data with S3 APIs or any other tools.
- Gateway VM allocates storage in two parts:
 - **Cache storage:**
 - It serves as on-premises durable storage.
 - It caches the data locally before uploading it to S3.
 - It provides low-latency access to frequently accessed data.
 - **Upload buffer:**
 - The upload buffer serves as a staging location prior to uploading the data to S3.
 - It uploads data on an encrypted SSL connection to AWS and stores it in an encrypted format on S3.

Gateway-stored volumes

You can use **gateway-stored volumes** when you need low-latency access to your entire dataset. It stores all your data locally first and then asynchronously takes a point-in-time backup of this data as a snapshot to S3. It is generally used as an inexpensive off-site backup option for DR:

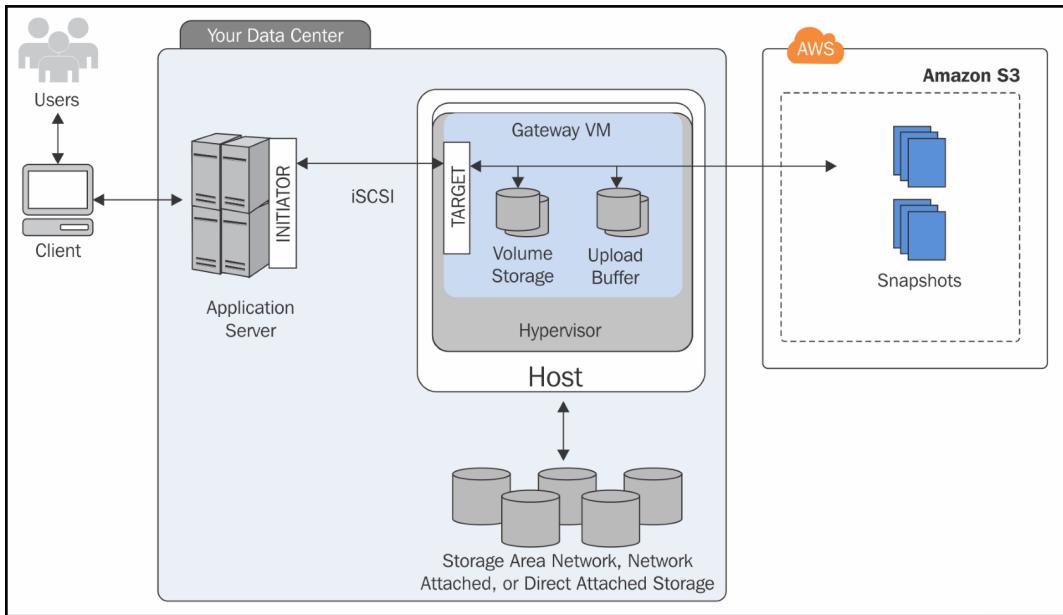


Figure 9.4: Gateway-stored volumes

The key features of a gateway-stored volume are as follows:

- It maintains the entire dataset locally, providing low-latency access to the data.
- It stores all your data locally first and then asynchronously takes a point-in-time backup of this data as a snapshot to S3.
- You can attach gateway-stored volumes as iSCSI devices on on-premises application servers.
- It supports up to 32 gateway-stored volumes per storage gateway application.
- Each gateway-stored volume can be from 1 GB to 16 TB in size with a total volume storage limit of 512 TB.
- Gateway-stored volumes can be restored as an EBS volume on an EC2 instance.
- Gateway-stored volume snapshots can be restored as a gateway storage volume, or you can create an EBS volume out of it and use it on an EC2 instance.

- The maximum size of an EBS volume created out of a snapshot is 16 TB and you cannot create an EBS volume out of the snapshot if it is more than 16 TB in size.
- AWS stores gateway-stored volume data and snapshots in Amazon S3 and the data is encrypted at rest with SSE; you cannot access the data with S3 APIs or any other tools.
- Gateway VM allocates storage in two parts:
 - **Volume storage:**
 - It is used for storing actual data.
 - You can map it to an on-premises **Direct-Attached Storage (DAS)** or **Storage Area Network (SAN)**.
 - **Upload buffer:**
 - The upload buffer serves as a staging location before uploading the data to S3.
 - It uploads data on an encrypted SSL connection to AWS and stores it in an encrypted format on S3.

Tape-based storage solutions

A **tape gateway** serves as a replacement for an on-premises tape drive for backup purposes. It stores data on Amazon Glacier for long-term archiving. It provides a virtual tape that can scale based on requirement. It also reduces the burden of managing the physical tape infrastructure.

There are two types of tape-based storage solutions—**Virtual Tape Library (VTL)** and **Virtual Tape Shelf (VTS)**.

VTL

VTL is a scalable and cost-effective virtual tape infrastructure, which seamlessly integrates with your existing backup software:

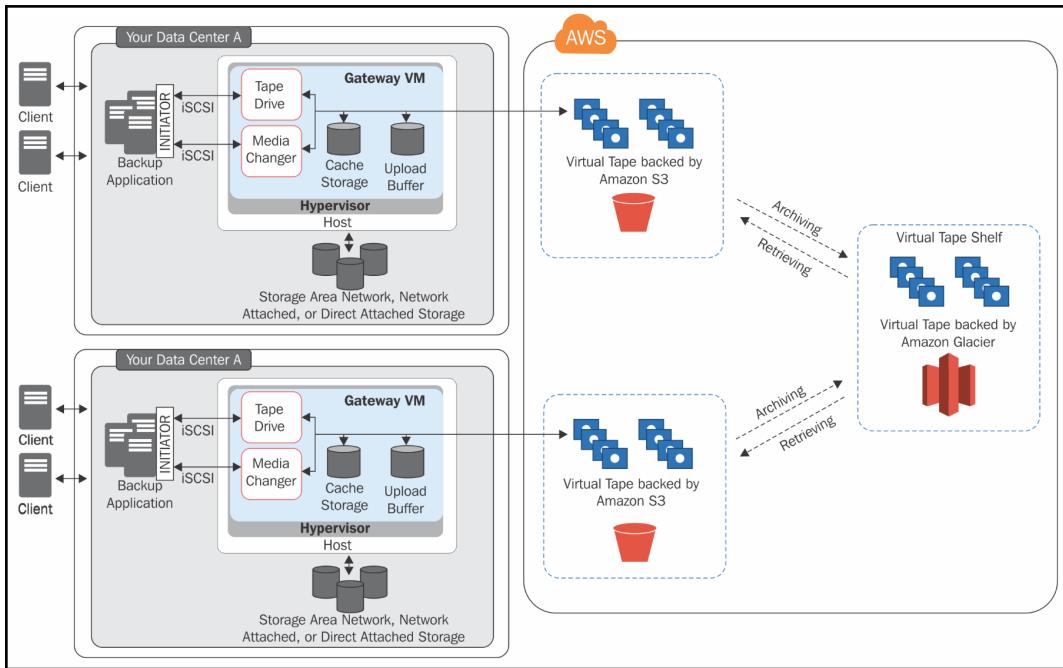


Figure 9.5: Gateway – virtual tape library

The key features of virtual tape library are as follows:

- It provides a low-cost and long-duration archival option in Glacier.
- It provides a virtual tape infrastructure that can scale based on requirements. It also reduces the burden of managing the physical tape infrastructure.
- It allows you to continue using your existing tape-based backup software for storing data on virtual tape cartridges, which can be created on a gateway-VTL.
- Each gateway-VTL is equipped with preconfigured media changer and tape drives. These are made available to existing backup applications as iSCSI devices. You can add tape cartridges as needed to archive the data.
- A gateway-VTL contains the following components:
 - **Virtual tape:**
 - Virtual tape emulates a physical tape cartridge wherein the data is stored in AWS storage solutions.

- You can have up to 1,500 tapes in each gateway or up to 150 TB of total tape data.
- Each tape can store from 100 GB to a maximum of 2.5 TB of data.

- **VTL:**

- VTL emulates a physical tape library wherein the data is stored in S3.
- When a backup software writes data to the gateway, at first the data is stored locally and, subsequently, it is asynchronously uploaded to virtual tapes in S3.

- **VTS:**

- VTS works just like an off-site tape-holding facility.
- In VTL, data is stored on S3 ,while VTS stores data in Glacier.
- As it uses Glacier for data archiving, it becomes an extremely low-cost data archival option.
- VTS resides in the same region where the Storage Gateway is created, and there is always only one VTS irrespective of the number of gateways created in an AWS account.
- The gateway moves a virtual tape to VTS when the backup software ejects a tape.
- You can retrieve tapes from VTS only after retrieving the tapes from VTL, and it takes around 24 hours for the tapes to be available in the VTL.
- Gateway allocates storage in the following two parts:
 - **Cache storage:**
 - It serves as an on-premises durable storage.

- It caches the data locally before uploading it to S3. It provides low-latency access to frequently accessed data.
- **Upload buffer:**
 - The upload buffer serves as a staging location, prior to uploading the data to S3.
 - It uploads data on an encrypted SSL connection to AWS and stores it in an encrypted format on S3.

AWS Snowball

AWS Snowball comes in hardware form and can be used with the AWS dashboard or API. It is available in two different sizes: 50 TB and 80 TB. It can be used to transfer PBs of data into and from AWS S3. Dedicated Snowball software is made available by AWS to perform data transfers in a compressed, encrypted, and secure manner.

You can attach multiple AWS Snowball devices at the same time to an on-premises network backbone. Perform the following steps to obtain AWS Snowball:

1. Sign into your AWS account and create a job inside the AWS Snowball management console. While creating a job, you need to provide information, such as shipping details, to receive Snowball device(s), job details mentioning the region, the AWS S3 bucket name, and so on. You also need to provide security details such as the ARN of the AWS IAM role and the master key from AWS KMS.

2. Once the job is created, the Snowball device is shipped to the given shipping address. An image of a Snowball device along with its features can be found at <https://image.slidesharecdn.com/clouddatamigration1272016final-160127210855/95/aws-january-2016-webinar-series-cloud-data-migration-6-strategies-for-getting-data-into-aws-14-638.jpg?cb=1466106757>.
3. Once the device is received, connect it to the network. It has two panels: one in the front and another in the back. Flipping the front panel on the top gives access to the E Ink-based touch screen to operate. Network and power cables can be connected to the back.
4. When Snowball is connected to an on-premises network, it becomes ready to transfer data. Snowball requires credentials to start a data transfer job. These credentials can be retrieved from the AWS dashboard or API. These credentials are encrypted with a manifest file and an unlock code. Without the manifest file and unlock code, it is not possible to communicate with Snowball. AWS provides a Snowball client to transfer data from on-premises to the Snowball device.
5. It is highly recommended that you do not delete the on-premises copy of the data until the data is successfully migrated to the AWS S3 bucket.
6. Once the job is complete, disconnect the device from the network and return the device to the shipping address displayed on the display panel. When you create a job, regional shipping carriers are assigned; for India, it is Amazon Logistics, and for the rest of the world, UPS is the shipping carrier partner.
7. Once the device is shipped back to AWS, the job progress status indicating the movement of data from Snowball to the AWS S3 bucket can be tracked on the AWS dashboard or through APIs.

AWS Snowmobile

AWS Snowmobile is an exabyte data transfer service. This hardware comes in a high cube shipping container that is 45 feet long, 8 feet wide, and 9.6 feet tall. Each Snowmobile truck can store up to 100 PB of data, and multiple Snowmobile trucks can be connected to an on-premises infrastructure at the same time. It uses 256-bit encryption, and a master key for encryption can be managed with AWS KMS. It comes with GPS tracking, alarm monitoring, 24/7 video surveillance, and an optional escort security vehicle while in transit.

When you request a Snowmobile, AWS performs an assessment and subsequently transports the Snowmobile to your designated location. An AWS resource configures it so that you can access it as network storage. During the entire period that the Snowmobile is at your location, AWS personnel work with your team. The AWS personnel connect a network switch from your Snowmobile to your local network. Once the setup is ready, you can start the data transfer process from multiple sources in your network to the Snowmobile.

Summary

- S3 is a cloud-based object storage service that can be used over the internet.
- Glacier is a highly secure, durable, and very low-cost cloud storage service for archiving data and taking long-term backups and is very economical for storing long-term archival data, which is rarely accessed.
- Retrieval incurs charges and may take a minimum of three to four hours or more depending on the size of the data.
- Amazon charges early deletion fees if data is deleted within three months from the date of storing.
- AWS EFS is a simple-to-use and scalable file storage service.
- EFS volumes can be mounted and accessed by multiple EC2 instances at the same time.
- EFS uses the **Network File System Version 4.1 (NFSv4.1)** protocol.
- EFS is available in two modes—General Purpose mode and Max I/O mode.
- EBS is a persistent, block-level storage service from Amazon.
- Persistent storage is a type of storage that retains the data stored on it even after power to the device is turned off.
- Block-level storage is a type of storage that can be formatted to support a specific filesystem, such as NFS, NTFS, SMB, or VMFS. EBS volumes can be attached to an EC2 instance.
- There are five variants of EBS: General Purpose SSD (gp2), Provisioned IOPS SSD (io1), Throughput optimized HDD (st1), Cold HDD (sc1) and Magnetic (standard).
- Instance store is a temporary block-level storage service from Amazon.

- An instance store is temporary in nature. Data stored in an instance store volume is deleted when the EC2 instance is either restarted, stopped, or terminated.
- AWS Storage Gateway is a hybrid storage service that connects on-premises environments with cloud storage using a software appliance.
- AWS Snowball is a petabyte-scale level data transport solution that uses physical appliances to transfer large-scale data from on-premises environments to the AWS cloud and vice versa.
- AWS Snowmobile is an exabyte-scale data transport solution that uses physical containers to transfer extremely large-scale data from on-premises environments to the AWS cloud and vice versa.
- Amazon CloudFront is a **Content Delivery Network (CDN)** offered by AWS. It is mainly used for caching static content, such as web pages, style sheets, client-side scripts, and images.

10

AWS Relational Database Service

AWS Relational Database Service (RDS) is a fully managed relational database service from Amazon. RDS makes it easier for enterprises and developers who want to use a relational database in the cloud without investing a lot of time and resources in managing the environment. AWS RDS supports six database engines—**Amazon Aurora, PostgreSQL, MySQL, MariaDB, Oracle, and Microsoft SQL Server**. It provides easy-to-use, cost-effective, and scalable relational databases in the cloud.

The following topics will be covered in the chapter:

- Introducing RDS
- RDS engine types
- Creating an Amazon RDS MySQL DB instance
- Monitoring RDS instances
- Creating a snapshot
- Restoring a DB from a snapshot
- Changing an RDS instance type
- Amazon RDS and VPC
- Connecting to an Amazon RDS DB instance
- RDS best practices

Introducing RDS

We have already seen what RDS in the introduction of this chapter and how it is useful. The advantages of Amazon RDS are as follows:

- It's a fully managed service that automatically manages backups, software and OS patching, automatic failover, and recovery.
- It also allows us to take a manual backup of the database as a snapshot. Snapshots of a database can be used to restore a database as and when required.
- RDS provides fine-grained access control with the help of AWS IAM.

AWS RDS does not provide root access to the RDS instance. In short, RDS does not allow the user to access the underlined host OS. That means that you cannot log into the server operating system. It also confines access to certain system procedures and tables that may require advanced privileges.

After launching RDS in its service offerings, AWS was not providing an option to stop an RDS instance for a very long time. Recently, an option to stop the RDS instance was introduced by Amazon. However, unlike EC2 instances, there are some limitations in stopping an RDS instance, which are as follows:

- You can stop and start any RDS DB engine.
- You can stop and start an RDS instance irrespective of whether it is in single-AZ or multi-AZ except for SQL Server. SQL Server RDS in multi-AZ cannot be stopped.
- An RDS instance can be stopped for a maximum of seven consecutive days. After seven days, the instance is automatically restarted.
- Any RDS DB instance with a Read Replica cannot be stopped.
- You cannot stop a Read Replica.
- Any stopped RDS DB instance cannot be modified.
- RDS does not allow you to delete an option group of a stopped database instance.
- RDS does not allow you to delete a parameter group of a stopped database instance.
- RDS does not allow you to stop and start the Aurora cluster that is part of the Aurora global database.
- RDS does not allow you to stop and start the Aurora cluster that uses the Aurora parallel query feature.

This way, by stopping an RDS instance, you can cut costs for a limited period of time. However, there is no limitation on restarting the instance or terminating all unused RDS DB instances to stop incurring the cost. If a manual snapshot is not taken before terminating the RDS DB instance, it prompts you to take a final snapshot. Once an RDS DB instance is deleted, it cannot be recovered.

Amazon RDS components

The **Amazon RDS components** are detailed in the following subsections.

DB instances

Each Amazon RDS engine can create an instance with at least one database in it. Each instance can have multiple user-created databases. Database names must be unique to an AWS account and are called **DB instance identifiers**. Each DB instance is a building block and an isolated environment in the cloud. These databases can be accessed using the same tools that are used to access standalone databases hosted in a data center. On top of standard tools, AWS RDS instances can also be accessed by the AWS Management Console, the API, and the CLI.

Each DB engine has its own version. With the help of a DB parameter group, DB engine parameters can be configured. These parameters help to configure DB instance performance. One DB parameter group can be shared among the same instance types of the same DB engine and version. These sets of allowed parameters vary according to the DB engine and its version. It is recommended that you create individual DB parameter groups for each database to have legacy to fine-tune each of them individually as per a business's needs. When you choose an RDS instance type, it determines how many CPUs and how much memory is allocated to it. The most suitable instance type can be selected based on the performance need. The general-purpose storage range for each DB engine can be categorized into two categories, as follows:

- One group of DB engines, such as MySQL, MariaDB, Oracle, and PostgreSQL, that support a minimum of 20 GB and a maximum of 32 TB.
- The second group of DB engines, such as MS SQL Server for Enterprise, Standard, Web, and Express editions, that support 20 GB to 16 TB storage.

Also, AWS periodically keeps revising this limit for different RDS engines. The minimum and maximum supported storage capacity may vary for each instance type. RDS supports **magnetic, general-purpose (SSD)**, and **provisioned IOPS (SSD)** storage types. RDS instances can be deployed within VPC. Based on the architectural needs, RDS can be deployed in a public subnet to be accessed over the internet or in a private subnet to access it within the network.



For more information, refer to https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_Storage.html.

Regions and AZs

AWS hosts its computing resources in data centers spread across the globe. Each geographical location where the data centers are located is called a **region**. Each region comprises multiple distinct locations that are called AZs. Amazon creates AZs in isolated locations so that a failure in one AZ does not impact other AZs in the region. AZs are interconnected with low-latency network connectivity within a region. When you launch an application in multi-AZs, it provides you with high availability and protects you from the failure of an AZ.

An RDS DB instance can be provisioned in several AZs by selecting the multi-AZ deployment option. It can also be used for DR sites. It is advisable to create RDS in multi-AZs for avoiding single points of failure. It automatically maintains synchronous replicas across multi-AZs. RDS synchronizes DBs between primary and secondary instances. If a primary instance fails, the load is automatically shifted to a secondary instance.

Security groups

Security groups act like a firewall. They control access to an RDS DB instance by specifying the allowed source port, protocol, and IPs. Three types of security groups can be attached with Amazon RDS DB instances—**DB security groups**, **VPC security groups**, and **EC2 security groups**.

In general, a DB security group is used when the RDS instance is not in the VPC. The VPC security group is used when the RDS instance is within the VPC. The EC2 security group can be used with EC2 instances as well as RDS instances.

DB parameter groups

Over a period when an RDS instance is used in enterprise applications, it may be required to tune certain allowed and common parameters to optimize the performance based on the data insertion and retrieval pattern. The same DB parameter group can be attached to one or more DB instances of the same engine and version type. If not specified, then the default DB parameter group with default parameters will be attached. Before creating an RDS instance, it is recommended that you create DB parameter groups.

DB option groups

DB options groups are used to configure RDS DB engines. With the help of the DB option groups, some of the DB engines can provide additional features for data management, and database management, and they can also provide additional security features. RDS supports DB option groups for **MariaDB**, **Microsoft SQL Server**, **MySQL**, and **Oracle**. Before creating an RDS instance, it is recommended that you create DB option groups.



Amazon RDS charges are based on instance type, running time, storage size, type and I/O requests, total backup storage size, and data in and out transfers.

RDS engine types

Amazon RDS supports six DB engine types—**Amazon Aurora**, **MySQL**, **MariaDB**, **Microsoft SQL Server**, **Oracle**, and **PostgreSQL**. The following table helps us to understand the connecting port and protocol for each of these DB instances:

Amazon RDS engine types	Default port	Protocol
Aurora DB	3306	TCP
MariaDB	3306	TCP
Microsoft SQL	1433	TCP
MySQL	3306	TCP
Oracle	1521	TCP
PostgreSQL	5432	TCP

The Amazon RDS engine for Microsoft SQL Server and Oracle supports two licensing models: **license included** and **Bring Your Own License (BYOL)**. If you are already invested in purchasing licenses for such databases, it can also be used as a BYOL with Amazon RDS to minimize monthly billing.

Supported instance types may vary for each Amazon RDS engine.

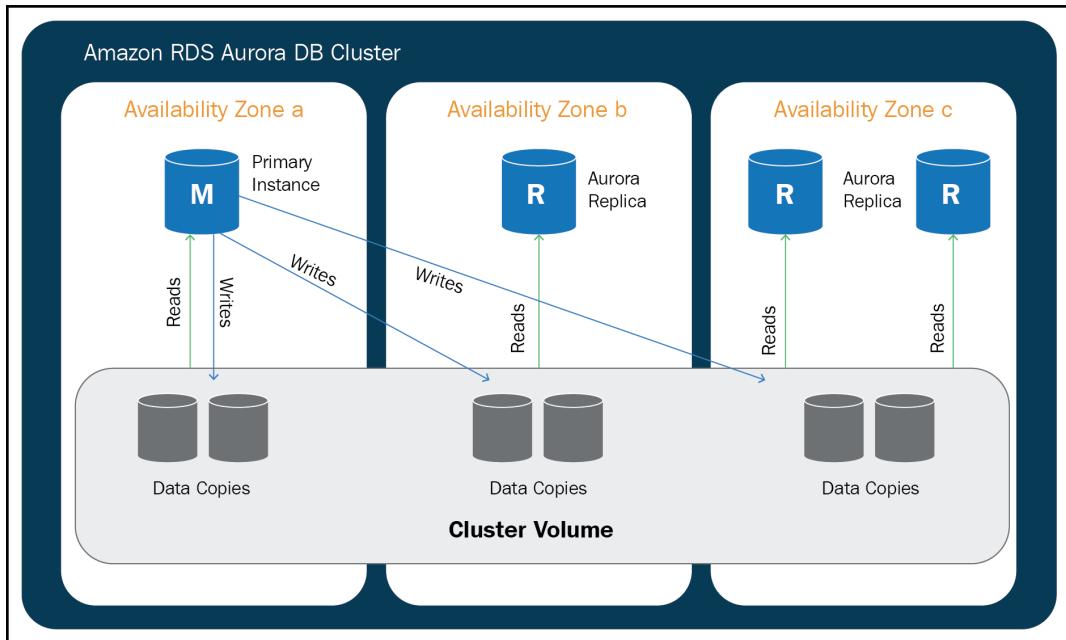


Amazon Aurora DB

Amazon Aurora is a MySQL and PostgreSQL-compatible, fully managed **Relational Database Management System (RDBMS)**. It gives an uncommon blend of the performance and reliability of commercial databases and the cost viability of open source databases. Your existing Amazon RDS for MySQL applications is converted to Amazon Aurora by providing push-button migration tools. It is also possible to use the code, tools, and applications you use today with your existing PostgreSQL databases with Aurora (PostgreSQL).

Creating an Amazon Aurora DB instance will create a DB cluster. It may consist of one or more instances along with a cluster volume to manage the data. These clusters consist of two types of instance—**primary instance** and **Aurora Replica**. Actually, the Aurora cluster volume is a virtual database storage volume of type SSD and it spans across multi-AZs in the same region. Each AZ will have a copy of the cluster data. Each Aurora cluster grows automatically as the amount of data in the database grows. It can grow up to 64 TB. The table size is limited to the cluster volume size, so the table can grow up to 64 TB in size:

- **Primary instance:** Performs read, writes, and modifies data to the cluster volume. Each Aurora DB cluster has one primary instance.
- **Aurora Replica:** Performs only read operations. Each Aurora DB cluster supports up to 15 Aurora Replicas plus one primary instance. Amazon RDS Aurora instance availability can be increased by spreading Aurora Replicas across multi-AZs. This diagram helps explain this:



Amazon RDS Aurora primary and replica

With the help of various endpoints, such as the cluster endpoint, reader endpoint, and instance endpoint, it is possible to connect to the Aurora DB cluster. Each endpoint consists of a domain name and port separated by a colon, and both are discussed as follows:

- **Cluster endpoint:** To connect to primary instances to perform data read, write, and modification operations. The primary instance also has its own endpoint. An advantage of the cluster endpoint is that it always points to the current primary instance.
- **Reader endpoint:** To connect to one of the Aurora Replicas to perform read operations. This endpoint automatically loads a balanced connection across available Aurora Replicas. If a primary instance fails, then one of the Aurora Replicas will be promoted as a primary instance, and, in that situation, all the read requests will be dropped.
- **Instance endpoint:** To directly connect with the primary or Aurora Replica instance.



An *endpoint* is a URL to access an AWS resource. It can be used to access the DB instance from an application, script, or as a CNAME in a DNS.

Aurora DB is designed to be highly durable, fault tolerant, and reliable, and provides the following features:

- **Storage auto-repair:** Aurora DB maintains multiple copies of data in three AZs to minimize the risk of disk failure. It automatically detects the failure of a volume or a segment and fixes it to avoid data loss and point-in-time recovery.
- **Survivable cache warming:** Aurora DB *warms* the buffer pool page cache for known common queries every time a database starts or is restarted after failure to provide performance. It is managed in a separate process to make it survive independently of the database crash.
- **Crash recovery:** Aurora DB instantly recovers from a crash asynchronously on parallel threads to make a database open and available immediately after the crash.

Amazon RDS upgrades the newer major version of Aurora to the cluster only during the system maintenance windows. Timing may vary from region to region and depending on cluster settings. Once a cluster is updated, the database restarts and may experience downtime for 20 to 30 minutes. It is highly recommended to configure maintenance window settings to match an enterprise's business requirements to avoid unplanned downtime. But, in the case of a minor version upgrade, Amazon RDS schedules an automatic upgrade for all Aurora DB database engines for all Aurora DB clusters. It is optional to allow that update at that scheduled time. It can be manually selected and updated at the desired schedule. Otherwise, it gets applied at the next automatic upgrade for a minor version release.



Amazon Aurora offers **lab mode**. By default, it is disabled. It can be enabled for testing current instances and available features in the currently offered version. New features can be tested before applying them to the production instance.

Comparing Amazon RDS Aurora to Amazon RDS MySQL

The following table helps understand the differences between Aurora and MySQL DB engines:

Features	Amazon RDS Aurora	Amazon RDS MySQL
Read scaling	Supports up to 15 Aurora Replicas with minimal impact on the write performance.	Supports up to only five Read Replicas with some impact on the write operation.
Failover target	Aurora Replicas are automatic failover targets with no data loss.	Manually, Read Replicas are promoted as a master DB instance with potential data loss.
MySQL version	Supports only MySQL Version 5.6.	Supports MySQL Version 5.5, 5.6, 5.7, and 8.0.
AWS region	May not be available in some regions.	Available in all regions.
MySQL storage engine	It supports only the InnoDB storage engine type. Tables from other types of storage engine are automatically converted to InnoDB.	Supports both MyISAM and InnoDB.
Read Replicas with a different storage engine than the master instance	MySQL (non-RDS) Read Replicas that replicate with an Aurora DB cluster can only use InnoDB.	Read Replicas can use both MyISAM and InnoDB.
Database engine parameters	Some parameters apply to the entire Aurora DB cluster and are managed by DB cluster parameter groups. Other parameters apply to each individual DB instance in a DB cluster and are managed by DB parameter groups.	Parameters apply to each individual DB instance or Read Replica and are managed by DB parameter groups.

Detailed comparison between Amazon RDS Aurora and Amazon RDS MySQL

MariaDB

MariaDB is a community version of MySQL RDBMS under a GNU GPL license. It maintains a high level of compatibility with MySQL.

Amazon RDS MariaDB manages versions as $x.y.z$, where $x.y$ denotes a major version and z is the minor version. For example, a version change from 10.0 to 10.1 is considered a major version change, while a version change from $10.0.17$ to $10.0.24$ is a minor version change. In general, within three to five months, it will be introduced in Amazon RDS MariaDB. Amazon RDS Management Console, CLIs, or APIs can be used to perform common tasks, such as creating an instance, resizing the DB instance, and creating and restoring a backup.

Minor version support may not be available in all AWS regions. Refer to https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_MariaDB.html for the following changes in storage engines supported by AWS RDS MariaDB.

Not all storage engines are optimized by Amazon RDS MariaDB for recovery and durability. At present, it fully supports the InnoDB and XtraDB storage engines. The default storage engine is InnoDB. It supports point-in-time restore and snapshot restore. It also supports the **Aria** storage type engine, but it may have a negative impact on recovery in the case of instance failure. Aria storage engine is mainly used for managing spatial geographical data. It is not recommended to use the Aria storage engine for general use.



For more information, refer to <https://aws.amazon.com/about-aws/whats-new/2018/03/amazon-rds-is-available-on-m4-instances-in-the-Govcloud-region/>.

Amazon RDS supports two kinds of upgrade for running instances—**major version upgrades** and **minor version upgrades**. Minor version upgrades can take place automatically when an automatic minor version upgrade is enabled from the instance configuration options. In all other cases, upgrading minor versions or major versions requires manual upgrades.

Microsoft SQL Server

It is possible to run Microsoft SQL Server as an RDS instance. It supports various versions of Microsoft SQL such as from SQL Server 2008 R2 to SQL Server 2017. There are a few limitations for Microsoft SQL Server DB instances, which are as follows:

- Each Amazon RDS Microsoft SQL instance can have a maximum of 30 databases. Master and model DBs are not counted as databases in this count.
- Some ports are reserved for internal purposes and cannot be used for general purposes.
- It is not possible to rename a database when an RDS instance with Microsoft SQL Server is deployed in multi-AZ mirroring.
- The minimum storage required is 20 GB with a maximum of 400 GB for the Web and Express editions. For the Enterprise and Standard editions, a minimum of 200 GB and a maximum of 16 TB of storage is required. Larger storage can be achieved with the help of sharding across multiple DB instances.



For more information, refer to https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_SQLServer.html for maximum storage size support.

- It is recommended that you allocate storage based on future considerations. Once storage volume is allocated, it cannot be increased due to the extensibility limitations of striped storage attached to Windows Server.
- It doesn't support some of the features of SQL Server such as SQL Server Analysis Services, SQL Server Integration Services, SQL Server Reporting Services, Data Quality Services, and Master Data Services. To use these features, you need to configure Microsoft SQL Server on an Amazon EC2 instance.
- Due to the limitations of Microsoft SQL Server, point-in-time restore may not work properly until the database has been dropped successfully.

Amazon RDS Microsoft SQL instances support two licensing options—**License Included** and **BYOL**. The License Included mode is good for the enterprise if you have not already purchased a license. If you have already purchased a license and are using it in an existing infrastructure, when migration to AWS cloud has been done and the instance is running with the help of a management console or CLI, BYOL can be implemented. When it is deployed in a multi-AZ mode, the secondary instance is passive and only provides read operations until failover takes place. BYOL is supported for the following Microsoft SQL Server database editions:

- Microsoft SQL Server Standard Edition (2017, 2016, 2014, 2012, 2008 R2)
- Microsoft SQL Server Enterprise Edition (2017, 2016, 2014, 2012, 2008 R2)

You may need to upgrade the Amazon RDS Microsoft SQL Server instance; Amazon RDS supports major version and minor version upgrades. In either type, it is essential to perform such upgrades manually. It requires downtime, and the total time depends on the engine version and the size of the DB instance.

MySQL

Amazon RDS supports various versions of MySQL. It is also compliant with many leading industry standards, such as HIPAA, PHI, and BAA. MySQL versions are organized as `x.y.z`, where `x.y` indicates a major version and `z` indicates a minor version. Most of the major versions are supported in most of the regions, but it is recommended that you check the availability of desired major and minor versions in a region where you are planning to create primary and DR sites. A new version of MySQL is available with the Amazon RDS MySQL instance usually within three to five months. While upgrading MySQL to a newer version, it is possible to maintain compatibility with specific MySQL versions. Major versions can be upgraded from MySQL 5.5 to MySQL 5.6 and then from MySQL 5.6 to MySQL 5.7. Usually, major version upgrades complete within 10 minutes, but it may vary based on the DB instance type. Minor versions automatically get updated when `AutoMinorVersionUpgrade` is enabled. Amazon RDS policy on deprecation of MySQL is as follows:

- Major versions are supported for three years from the release, such as 5.5, 5.6, 5.7, and upcoming versions.
- Minor versions are supported for a year from release, such as MySQL 5.546.
- Three months of grace are provided from the date of the version deprecation date.



It is essential to perform an OS update (if any are available) before upgrading an Amazon RDS MySQL 5.5 DB instance to MySQL 5.6 or later.

Amazon RDS MySQL 5.6 and later versions support memcached in an option group. Amazon RDS MySQL also supports various storage engines, but point-in-time recovery is only supported by InnoDB. Amazon RDS MySQL 5.7.23 and the later MySQL 5.7 version supports **Global Transaction Identifiers (GTIDs)**. GTIDs are not supported for Amazon RDS MySQL 5.5., 5.6, or 8.0, and it currently does not support the following MySQL features:

- Plugin for authentication.
- Password strength plugin.
- Persisted system variables.
- Replication filters.
- Plugin for error logging to the system log.
- Group replication plugin.
- InnoDB Tablespace encryption plugin.
- MariaDB Audit plugin (not supported for Amazon RDS, MySQL Version 8.0 only). It is supported for Amazon RDS MySQL versions 5.5, 5.6, and 5.7.
- Semi-synchronous replication.
- X Plugin.
- Transportable tablespace.



For more information, refer to https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_SQL.html.

It is possible to create a snapshot for an Amazon RDS MySQL instance storage volume. Each snapshot is based on the MySQL instance engine version. As it is possible to upgrade the version of an Amazon RDS MySQL instance, it is also possible to upgrade the engine version for DB snapshots. It supports DB snapshot upgrades from MySQL 5.1 to MySQL 5.5.

Oracle

At the time of writing, the following Oracle RDBMS versions are supported by the Amazon RDS Oracle engine:

- Oracle 12c, version 12.2.0.1
- Oracle 12c, version 12.1.0.2
- Oracle 11g, version 11.2.0.4

The Amazon RDS Oracle engine also supports the following Oracle RDBMS versions, but soon they will be deprecated:

- Oracle 12c, version 12.1.0.1
- Oracle 11g, version 11.2.0.3 and version 11.2.0.2

Oracle RDS can be deployed within VPC and can perform point-in-time recovery and scheduled or manual snapshots. It can optionally be deployed in multi-AZ to get high availability and failover. At the time of creating a DB instance, the master user gets DBA privileges with some limitations. For example, `SYS` user, `SYSTEM` user, and other DB administrative user accounts cannot be used.

Amazon RDS Oracle instances support two licensing options—**License Included** and **BYOL**. Once an instance is running with the help of a management console or CLI, BYOL can be implemented. In the case of License Included, it supports the following Oracle database versions:

- Oracle Database **Standard Edition One (SE1)**
- Oracle Database **Standard Edition Two (SE2)**

BYOL supports the following license models:

- Oracle Database **Enterprise Edition (EE)**
- Oracle Database **Standard Edition (SE)**
- Oracle Database **Standard Edition One (SE1)**
- Oracle Database **Standard Edition Two (SE2)**

Amazon allows you to change Oracle RDS instance types; however, if your DB instance uses a deprecated version of Oracle, you cannot change the instance type. Such RDS instances are automatically updated to a new version based on a cut-off date provided by Amazon. For more details on supported versions, deprecated versions, and cut-off dates for upgrading the deprecated versions, go to http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_Oracle.html.

This supports both major and minor version upgrades. While upgrading an Amazon RDS Oracle instance from 11g to 12c is a major version upgrade, it has to be done manually and it requires downtime. This downtime may vary based on the current engine version and size of the DB instance. While upgrading the engine version, it takes two snapshots:

- The first snapshot is taken just before upgrading in the case of failure for any reason. It can be used to restore the database.
- The second snapshot is taken just after the engine upgrade is completed.

Once the DB engine has successfully been upgraded, it cannot be undone. If there is any requirement to rollback to a previous version, you can create a new instance with the snapshot taken before upgrading the version. The Oracle engine upgrade path may vary depending on the current version running on the instance.

PostgreSQL

The Amazon RDS PostgreSQL engine supports various versions of PostgreSQL. It also supports point-in-time recovery using periodically or manually taken snapshots, multi-AZ deployment, provisioned IOPS, Read Replicas, SSL connection to DB, and VPC. Applications such as pgAdmin or any other tool can be used to connect to PostgreSQL and run SQL queries. It is also compliant with many industry leading standards such as HIPAA, PHI, BAA, and many others.

At the time of creating an Amazon RDS PostgreSQL instance master user (super user), a system account is assigned to the `rds_superuser` role with some limitations.



More details about various PostgreSQL supported versions and their features can be obtained from https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_PostgreSQL.html.

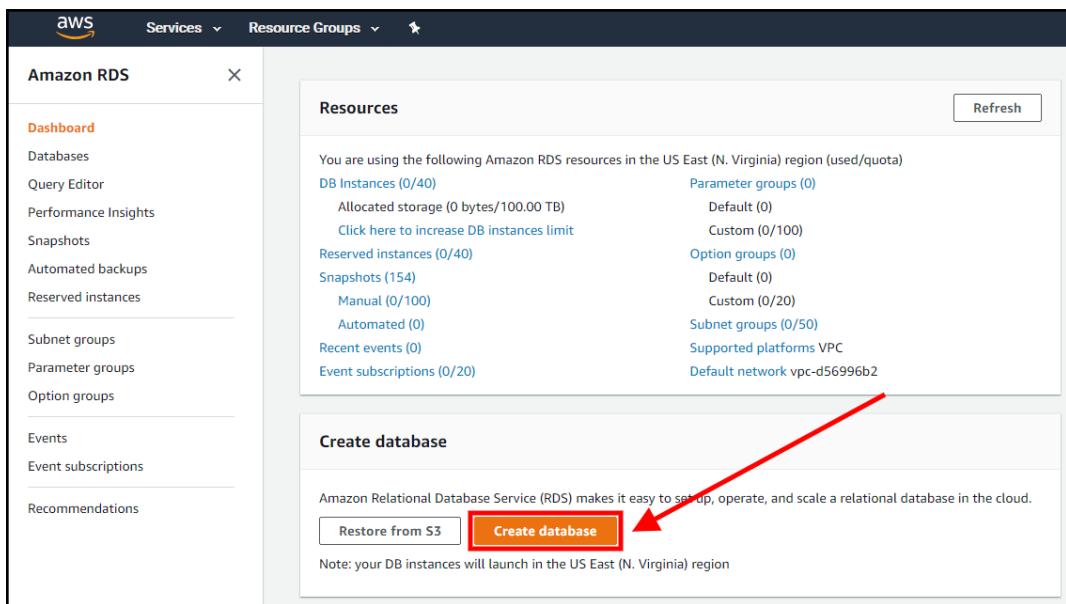
Amazon RDS supported and unsupported database engines can be installed and configured on Amazon EC2 instances as well. Compared to Amazon RDS, installing a DB on Amazon EC2 gives more power to fine-tune database engines, as it gets root-level access. Usually, when an enterprise is looking for a managed service solution, Amazon RDS is preferred, and when enterprise are looking for more detailed fine-tuning, hosting on Amazon EC2 is preferred.

Now that you have a fair understanding of different RDS engine types, it is time to look understand how to create an RDS instance. The next topic takes you through the process of creating an RDS instance with the MySQL database engine.

Creating an Amazon RDS MySQL DB instance

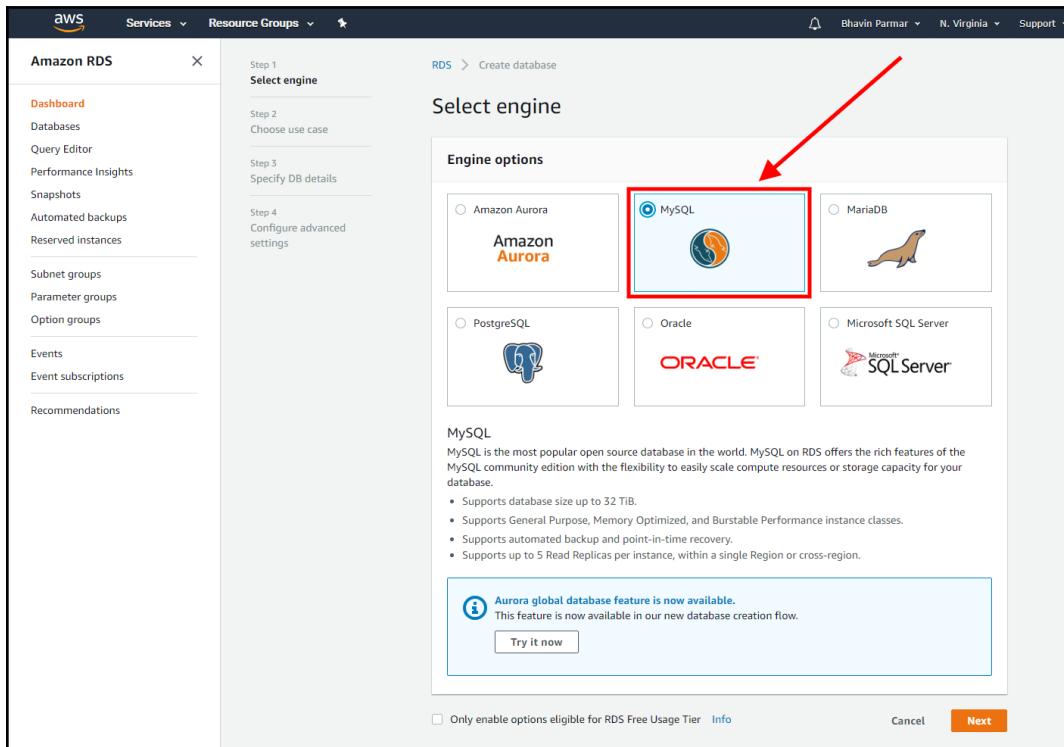
Amazon RDS MySQL DB instances can be created using the Amazon Management Console, CLIs, or APIs, and the steps are as follows:

1. Log into the AWS Management Console with the appropriate user privileges and go to the Amazon RDS dashboard.
2. Select **Create database**, as shown:



Select Create database

3. Select the engine type as MySQL, as shown:



Select the Amazon RDS engine type as MySQL

The **Only enable options eligible for RDS Free Usage Tier** option is only allowed during the first year of the free tier, such as `t2.micro` single-AZ instance, as shown:

MySQL

MySQL is the most popular open source database in the world. MySQL on RDS offers the rich features of the MySQL community edition with the flexibility to easily scale compute resources or storage capacity for your database.

- Supports database size up to 32 TiB.
- Supports General Purpose, Memory Optimized, and Burstable Performance instance classes.
- Supports automated backup and point-in-time recovery.
- Supports up to 5 Read Replicas per instance, within a single Region or cross-region.

 **Aurora global database feature is now available.**
This feature is now available in our new database creation flow.

[Try it now](#)

Only enable options eligible for RDS Free Usage Tier [Info](#)

[Cancel](#) [Next](#)

Optionally, select Only enable options eligible for RDS Free Usage Tier

4. Select the **Production - MySQL** use case to have multi-AZ deployment or **Dev/Test - MySQL**, as shown. It is also suggested that you switch to Amazon Aurora, as it is seamlessly compatible with MySQL:

The screenshot shows the 'Choose use case' step of the AWS RDS 'Create database' wizard. On the left, a vertical navigation bar lists steps: Step 1 (Select engine), Step 2 (Choose use case, which is currently active), Step 3 (Specify DB details), and Step 4 (Configure advanced settings). The main content area is titled 'Choose use case' and contains a section titled 'Use case'. It asks, 'Do you plan to use this database for production purposes?'. Three options are listed: 'Production - Amazon Aurora' (Recommended), 'Production - MySQL', and 'Dev/Test - MySQL'. The 'Dev/Test - MySQL' option is selected and highlighted with a red box and a red arrow pointing to it. Below the options, a note states, 'This instance is intended for use outside of production or under the RDS Free Usage Tier.' At the bottom, there are 'Cancel', 'Previous', and 'Next' buttons.

Select Amazon RDS MySQL instance to deploy in a single-or multi-AZ deployment

5. Specify the Amazon RDS MySQL DB details as follows:

- **License model:** At present, there is only one license model – **general-public-license**.
- **DB engine version:** The Amazon RDS MySQL engine supports various versions. Based on the enterprise IT requirement, the optimum and latest can be selected.
- **Known Issues/Limitations:** This will open a new window to list known issues and limitations for working with MySQL on Amazon RDS.
- **Free tier:** This will only enable options on a web console suitable for free tier usage. For example, a replica of an RDS instance is not in the free tier; it would be disabled when the free tier had been selected.
- **DB instance class:** Select the RDS instance type. It decides the size of RAM, CPU, network performance, and EBS performance.
- **Multi-AZ deployment:** Select **Yes** to enable a standby replica of the DB instance in another AZ for failover support.

- **Storage type:** Supports three types – **Magnetic**, **General Purpose (SSD)**, and **Provisioned IOPS (SSD)**. The storage type can be selected based on the required number of read/write operations.
- **Allocated storage:** The size of the storage volume to attach to the Amazon RDS DB instance.
- **Estimated monthly costs:** Suggests the approximate cost for hosting a configuration.
- **DB instance identifier:** This is a unique DB name for each DB within the AWS account.
- **Master username and Master Password:** The master user is the user with the highest level of privileges within each Amazon RDS instance. It is used to create enterprise-level users and grant them privileges to perform day-to-day activities and applications. It also defines passwords:

The screenshot shows the 'Specify DB details' step of the AWS RDS 'Create database' wizard. The left sidebar lists steps: Step 1 (Select engine), Step 2 (Choose use case), Step 3 (Specify DB details, currently active), Step 4 (Configure advanced settings), and Step 5 (Review and create).

Instance specifications:

- DB engine: MySQL Community Edition
- License model: general-public-license
- DB engine version: MySQL 5.7.23

Known Issues/Limitations: A note about potential compatibility issues with specific database versions.

Free tier: A note about the Amazon RDS Free Tier providing a single db.t2.micro instance as well as up to 20 GB of storage, allowing new AWS customers to gain hands-on experience with Amazon RDS. It also mentions the RDS Free Tier and instance restrictions.

Allocated storage: Set to 20 GiB.

Multi-AZ deployment: Set to 'Create replica in different zone'.

Storage type: General Purpose (SSD).

Allocated storage: Set to 20 GiB.

Estimated monthly costs:

DB Instance	24.82 USD
Storage	4.60 USD
Total	29.42 USD

Billing estimate is based on on-demand usage as described in [Amazon RDS Pricing](#). Estimate does not include costs for backup storage, I/Os (if applicable), or data transfer.

Settings:

- DB instance identifier:** learnAmazonMySQL
- Master username:** admin
- Master password:** [REDACTED]
- Confirm password:** [REDACTED]

Master Password must be at least eight characters long, as in "mypassword". Can be any printable ASCII character except "/", "<", ">", or ":".

Buttons: Cancel, Previous, Next.

Amazon RDS MySQL DB instance details

6. Configure advanced settings from the subsequent screen as per the details given next:

- **Virtual Private Cloud (VPC):** Select a suitable network VPC.
- **Subnet group:** The subnet selection depends on the architectural design. It can be public or private based on requirements.
- **Public accessibility:** It should be selected as **Yes** if you want to allow the access to the DB from the internet. It creates a public DNS endpoint, which is globally resolvable. Select **No** if you want this DB instance to be accessible only from within the network or VPC.
- **Availability zone:** If you have any preference on which AZ you want to launch your instance, you choose the specific AZ as required. If you do not have any preferred AZ, you can select **No preference**. In this case, Amazon automatically launches the instance in the appropriate AZ to balance the resource availability.
- **VPC security groups:** Security groups act as a software firewall. One or more security groups can be attached to each Amazon RDS instance.
- **Database name:** It can be a maximum of 64 alpha-numeric characters. For a given name, it will create a database within the DB instance. It can also be blank.
- **Port:** For Amazon RDS MySQL DB instances, the default port is 3306.
- **DB parameter group:** This helps to configure DB engine parameters. It is recommended that you create the DB parameter group before creating a DB instance. Once it is created, it will appear in an available drop-down list to use at the time of creating a DB instance. If it is not created before creating a DB instance, then the default DB parameter group will be created. This group of parameters can be applied to one or more DB instances of the same engine type. When any dynamic parameter value is changed in the DB parameter group, it gets applied immediately whether **Apply immediately** has been enabled or not. If there is any change in the static parameter value, it is required to manually reboot the DB instance for the change to become effective on the instance.

- **Option group:** It is supported by the MariaDB, Microsoft SQL Server, MySQL, and Oracle Amazon RDS engines. With the help of option groups, it is possible to fine-tune databases and manage data. Option groups can consist of two types of parameter—**permanent** and **persistent**. To change the persistent option's value, the DB instance needs to be detached from the DB option group. When the option group is associated with any DB snapshot, to perform point-in-time recovery using that DB snapshot, a new DB instance needs to be created with the same DB options group. On the other hand, it is not possible to remove permanent options from an option group. Also, an option group with permanent options cannot be detached from a DB instance.
- **IAM DB authentication:** Select **Yes** to manage database user credentials through AWS IAM users and roles.
- **Encryption:** Enabling this option will encrypt data at rest in the DB instance's storage volume and subsequent snapshots. The industry standard AES-256 encryption algorithm is used. Amazon RDS automatically takes care of authentication and encrypts/decrypts data with a minimal impact on performance.
- **Backup retention period:** You specify the snapshot retention period here in a number of days. Snapshots that are older than the specified number of days are automatically deleted. Any snapshot can be retained for a maximum of 35 days.
- **Backup window:** An automated backup time window can be specified in a UTC. During this scheduled time, a snapshot will be taken every day. When any snapshot exceeds the backup retention period, it automatically gets deleted. This helps to achieve an organizational backup and retention policy and optimizes AWS billing by removing the obsolete and old snapshots.
- **Copy tags to snapshots:** In a backup window, and it creates an instance-level backup (that is, a snapshot) for the entire volume. By enabling this parameter, each snapshot will copy tags from the DB instance. This metadata can be very helpful to manage access policies.

- **Enable enhanced monitoring:** Amazon RDS maintains various performance metrics in Amazon CloudWatch. The main difference between normal and enhanced monitoring is the source of the data. In the case of normal monitoring, CPU utilization of data is derived from the hypervisor, but for enhanced monitoring, it is derived from the agent installed on a hypervisor. Data collection from these two sources may vary. In the case of small instance types, this difference can be bigger.
- **Log exports:** Error logs are enabled by default. Optionally, **Audit log**, **General log**, and **Slow query log** can be enabled.
- **Auto minor version upgrade:** Amazon RDS instances can have two types of upgrade—major and minor. Major version upgrades may require downtime, so they are not performed automatically. It may also not be possible to reverse a major version upgrade. Minor version upgrades are compatible with previous versions and may not require downtime, so they are performed automatically during scheduled maintenance.
- **Maintenance window:** Amazon allows you to specify a maintenance window. During the maintenance window, Amazon may upgrade the DB instance's minor version or the DB cluster's OS. Upgrade of the underlined OS or DB version may bring performance implications. Considering this, you should carefully define the maintenance window. The maintenance window definition allows you to define the starting day of the week, hour of the day, minute of the hour, and the total allocated time to perform the maintenance activity. Once the maintenance activity begins, and if it requires more time to complete the maintenance, it doesn't terminate in between. It stops only after completing the maintenance tasks.
- **Deletion protection:** The Amazon RDS instance can be protected from accidental deletion if required.

Now that you know how to create an RDS instance, let's learn about monitoring RDS instances in the next section.

Monitoring RDS instances

Once an Amazon RDS instance is created as per the present need, it is very important to observe its performance with constantly changing business requirements and application loads. It is possible to monitor the instance's CPU utilization, DB connections, free storage space, free memory, and many other parameters. It helps to identify bottlenecks and will also give you the opportunity to minimize monthly billing by reducing the resource size if it is underutilized.

An alarm can be configured to take action at a specified threshold. For example, if CPU usage is above 70% for a specified consecutive time period, then send SNS notifications to the DBA. Such an alarm can be created either from the CloudWatch dashboard or from the Amazon RDS dashboard.

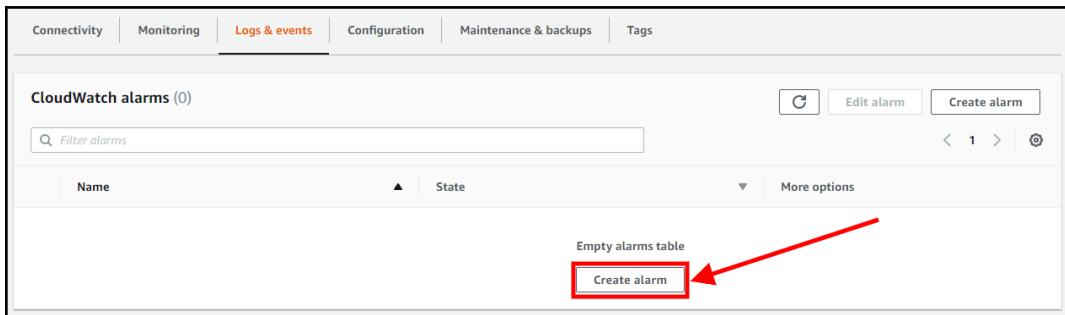
To create a CloudWatch alarm from the Amazon RDS dashboard, perform the following steps:

1. Go to the Amazon RDS dashboard and select the desired DB instance from the list of running DB instances.
2. Click **Logs & events** to get the list of events and CloudWatch alarms as shown:

The screenshot shows the Amazon RDS Dashboard for a database named 'learnamazonrdsmysql'. The 'Logs & events' tab is highlighted with a red box and an arrow pointing to it. The 'Summary' section displays various metrics: DB Name (learnamazonrdsmysql), Role (Instance), CPU (15.83%), Current activity (0 Connections), Info (Backing-up), Engine (MySQL), Class (db.t2.micro), and Region & AZ (us-east-1b). Below the summary, there are tabs for Connectivity, Monitoring, Logs & events, Configuration, Maintenance & backups, and Tags. The 'Logs & events' tab is active. Under 'CloudWatch alarms (0)', there is a 'Create alarm' button. At the bottom, a note says 'Select Logs & events to create a new CloudWatch alarm and existing list'.

Select Logs & events to create a new CloudWatch alarm and existing list

3. Click on **Create alarm** as shown in the following screenshot:



4. Create an alarm by specifying the threshold and other relevant details, such as the SNS topic to use to send notifications, the CPU utilization threshold, consecutive time period, and alarm name, as shown:

The screenshot shows the 'Create alarm' dialog box. At the top, the path 'RDS > Databases > learnamazonrdsmysql' is visible. The main title is 'Create alarm' with a sub-instruction: 'You can use CloudWatch alarms to be notified automatically whenever metric data reaches a level you define.' Below this is a 'Settings' section with a 'Refresh' button. Under 'Send notifications', the 'Yes' radio button is selected. In the 'Send notifications to' section, the 'ARN' radio button is selected. The 'ARN' field contains a placeholder 'ARN to send notifications to.' The 'Metric' section shows 'Average' selected for the metric type and 'CPU Utilization' selected for the metric name. The 'Threshold' section shows a threshold of '65 Percent'. The 'Evaluation period' section shows '1 consecutive period(s) of 5 Minutes'. To the right, a chart titled 'CPU Utilization Percent' shows a single data series named 'learnamazonrdsmysql' with a value of 65 at three time points: 12/18 16:00, 12/18 18:00, and 12/18 20:00. The 'Name of alarm' field contains the name 'awsrds-learnamazonrdsmysql-High-CPU-Utilization'. At the bottom right are 'Cancel' and 'Create alarm' buttons, with the 'Create alarm' button highlighted with a red box.

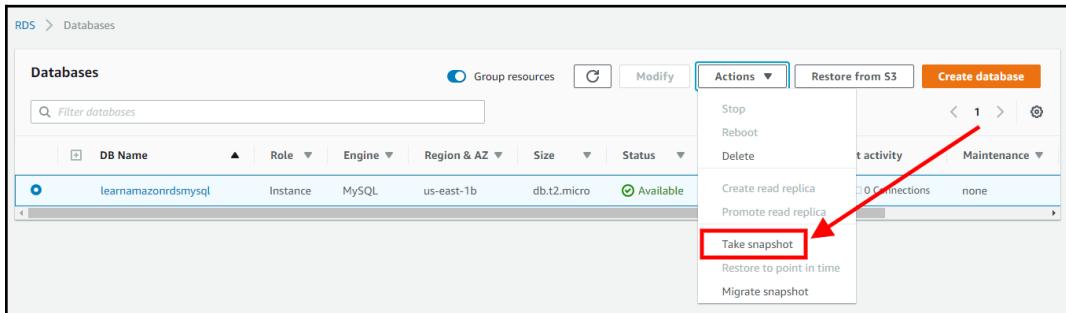
Create a CloudWatch alert and action from the Amazon RDS dashboard

Creating a snapshot

A **snapshot** is a frozen image of the DB instance's storage volume. It helps to restore a database to a particular point in time. Usually, point-in-time recovery is performed when a database is corrupted or by mistake some data has been dropped (that is, deleted), to bring a database back to the last healthy state. At the time of creating an Amazon RDS instance, a daily snapshot schedule has already been configured, but it may sometimes be required to take a manual snapshot of the DB instance before performing any maintenance tasks on the database. A snapshot will back up an entire DB instance including all databases, tables, and other resources existing on it.

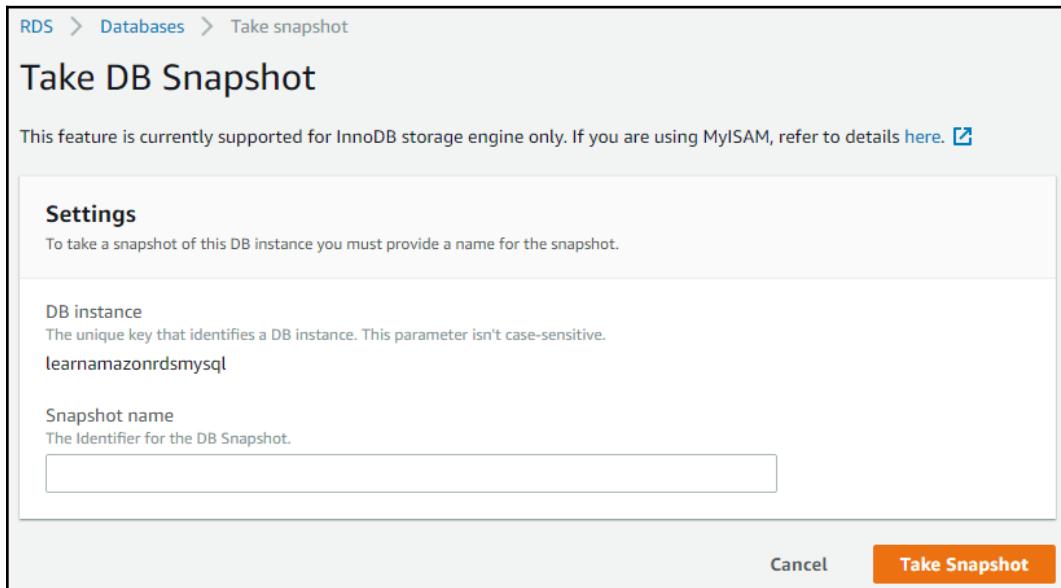
Creating a snapshot for a multi-AZ DB instance doesn't bring many performance implications, but taking a snapshot for a single-AZ DB instance may suspend DB I/O, for a few seconds to minutes. Manual snapshots can be taken using the Amazon Management Console, CLI, or APIs. To take a manual snapshot using the management console, perform the following steps:

1. Select the desired DB instance.
2. Select **Take snapshot** from the instance's **Actions** drop-down menu, available above the list of the running RDS instances:



Take a manual Amazon RDS DB instance snapshot

3. Provide the relevant **Snapshot name**, as shown in the next screenshot, and then click on **Take Snapshot**:



Provide snapshot name, while taking a manual snapshot

Now that you have created a snapshot, let's have a look at how we can restore a DB from a snapshot.

Restoring a DB from a snapshot

A snapshot can only be restored by creating a new instance. You cannot restore a snapshot to an existing instance. While restoring the snapshot to a new RDS instance, you can have a different storage volume type from the one used in the snapshot.

Creating an RDS DB instance from a snapshot automatically attaches a default parameter group and security group to it. Once a DB instance is created, it is possible to change the attached parameter group and security group for that instance. By restoring a snapshot, the same option group associated with the snapshot will get associated to the newly created RDS DB instance. Option groups are platform-specific—VPC or EC2-Classic.

Creating an RDS DB instance inside a particular VPC will link a used option group with that particular VPC. It means that when the snapshot is created for that DB instance, it cannot be restored in a different VPC. To do that, it requires us to either attach a default option group or create a new option group and attach it to the newly created DB instance from the snapshot.

Creating a DB instance from a snapshot also requires us to provide parameters such as **DB engine**, **License model**, **DB instance class**, **Multi-AZ deployment**, **Storage type**, **DB instance identifier**, **Virtual Private Cloud (VPC)**, **Subnet group**, **Public accessibility**, **Availability zone**, **Database name**, **Port**, **Option groups**, and other parameters that we define at the time of creating a new Amazon RDS DB instance.

It is also possible to copy and share an Amazon RDS snapshot from one region to another and share it among multiple AWS accounts, respectively. It may require us to create a DB instance from a snapshot in a different region or AWS account.

Changing an RDS instance type

An RDS instance type is generally changed to accommodate additional resource requirements or to downgrade an existing instance type that is underutilized. To change the instance type, perform the following steps:

1. Select the desired Amazon RDS instance to change type and click on **Modify** as shown:

The screenshot shows the AWS RDS Databases console. In the top navigation bar, 'RDS' is selected, followed by 'Databases' and 'learnamazonrdsmysql'. On the left, there's a sidebar with 'Summary' and 'Actions ▾'. The main area displays the 'Summary' of the 'learnamazonrdsmysql' instance. The instance details include:

- DB Name: learnamazonrdsmysql
- Role: Instance
- CPU: 1.36%
- Current activity: 0 Connections
- Info: Available (green)
- Engine: MySQL
- Class: db.t2.micro
- Region & AZ: us-east-1b

A red arrow points to the 'Modify' button in the top right corner of the summary card.

Select Modify to change the Amazon RDS instance configuration

2. From the list of RDS DB instances, select the desired instance to modify the instance type and select **Modify** from the instance's **Actions** drop-down menu.

3. Modifying a DB instance does not only allow us to change the DB instance type; it also allows us to change many other parameters that are provided at the time of creating a DB instance, such as subnet group, security group, and many more options. At the end of the parameters that can be changed, an option is available to apply changes now or wait for an upcoming maintenance window, as shown:

The screenshot shows the 'Summary of modifications' section with one item: 'DB instance class' changed from 'db.t2.micro' to 'db.t2.small'. In the 'Scheduling of modifications' section, the 'Apply immediately' option is selected. A warning box states: 'If you choose to apply changes immediately, please note that any changes in the pending modifications queue are also applied. If any of the pending modifications require downtime, choosing this option can cause unexpected downtime.' At the bottom are 'Cancel', 'Back', and a large orange 'Modify DB Instance' button.

Summary of modifications
You are about to submit the following modifications. Only values that will change are displayed. Carefully verify your changes and click Modify DB Instance.

Attribute	Current value	New value
DB instance class	db.t2.micro	db.t2.small

Scheduling of modifications

When to apply modifications

Apply during the next scheduled maintenance window
Current maintenance window: tue:05:38-tue:06:08

Apply immediately
The modifications in this request and any pending modifications will be asynchronously applied as soon as possible, regardless of the maintenance window setting for this database instance.

Potential unexpected downtime
If you choose to apply changes immediately, please note that any changes in the pending modifications queue are also applied. If any of the pending modifications require downtime, choosing this option can cause unexpected downtime.

Cancel Back **Modify DB Instance**

Apply DB instance parameter modifications immediately or wait for the next maintenance window

4. If DB performance is throttling, you can change the DB instance parameters and apply them immediately. If you do not apply them immediately, the changes are automatically applied during the next maintenance window. Some modifications, such as parameter group changes, may require us to reboot DB instances. It is advisable to test any changes in a test environment first, before making the direct changes in production environments.

Amazon RDS and VPC

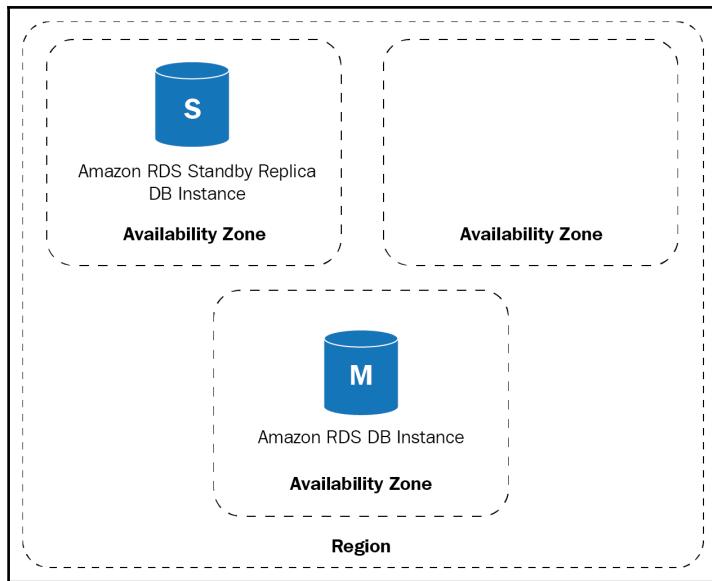
Before 2013, AWS supported EC2-Classic. All AWS accounts created after December 4, 2013 only support EC2-VPC. If an AWS account only supports EC2-VPC, then a default VPC is created in each region and a default subnet in each AZ. Default subnets are public in nature. To meet enterprise requirements, it is possible to create a custom VPC and subnet. This custom VPC and subnet can have a custom CIDR range and can also decide which subnet can be public and which one can be private. When an AWS account only supports EC2-VPC and it has no custom VPC created, then Amazon RDS DB instances are created inside a default VPC.

Amazon RDS DB instances can also be launched into a custom VPC just like EC2 instances. Amazon RDS DB instances have the same functionality in terms of performance, maintenance, upgrading, recovery, and failover detection capability, irrespective of whether they are launched in a VPC.

Amazon RDS and high availability

ELB and auto-scaling can be used with Amazon EC2 to perform load balancing and launching or terminating an EC2 instance to match the load requirement. Auto-scaling cannot be used with Amazon RDS. Amazon RDS supports multi-AZ deployment to provide high availability and failover. By enabling multi-AZ deployment, Amazon RDS creates two instances of the same instance type and configuration with individual endpoints in two separate AZs. The sole purpose of another DB instance is to maintain a synchronous standby replica. The standby replica receives traffic only when failover takes place. It cannot be used for load balancing or serving read-only traffic.

For serving read-only traffic, Read Replicas can be created, which is different than creating multi-AZ instances. At the time of writing this book, Amazon RDS supports six DB engines, that is, Aurora, MySQL, Oracle, SQL Server, PostgreSQL, and MariaDB. Four out of the six DB engines, namely Oracle, PostgreSQL, MySQL, and MariaDB, can perform failover from a primary DB instance to a secondary DB instance using Amazon's failover mechanism. The Microsoft SQL Server RDS engine uses SQL Server mirroring for high availability. The Amazon Aurora cluster creates at least three copies of data across multi-AZs within the same region, which can fulfil the high-availability requirement. In Amazon Aurora, one of the Aurora Replicas is promoted as a primary if the primary DB instance fails:



Amazon RDS DB instance in a multi-AZ

The preceding diagram helps us to understand the Amazon RDS DB primary and secondary instance in a VPC, where the primary instance is denoted as **M** and the secondary instance is denoted as **S**.



You must have a license for both the primary instance and the standby replica, when you are using the BYOL licensing model.

Connecting to an Amazon RDS DB instance

Once the Amazon RDS DB instance is created, you can connect to it to perform read/write operations as well as day-to-day maintenance activities. Before connecting to the DB instance, ensure that the port to connect with the DB instance is allowed in the firewall or security group. Also, ensure that the source IP from where you need to connect to the DB instance is allowed in the security group. This topic describes how you can connect to various RDS DB instances with different database engine types.

Connecting to an Amazon Aurora DB cluster

Aurora DB clusters consist of a primary instance and an Aurora Replica. A separate endpoint is available for the primary instance, Aurora Read instance, or a group of Aurora Read instances. In line with the task you want to carry out, it is possible to use any of these endpoints in scripting, application, or manually connecting them. Tools used to connect with MySQL databases can be used to connect to Amazon Aurora cluster DB instances.

You can refer to the following syntax for connecting to an Aurora DB:

```
mysql -h <aurora-cluster-endpoint> --ssl-ca=[full path]rds-combined-ca-bundle.pem --ssl-verify-server-cert
```

Connecting to a MariaDB instance

Every Amazon RDS MariaDB instance that is up and running has a valid endpoint. It can be used with an application, client, or tool to connect with the DB instance. By default, it uses port 3306 and the TCP protocol. Amazon RDS MariaDB instances can be accessed from the mysql command-line utility. **HeidiSQL** is a GUI-based utility and it can be used to connect MariaDB instances.

The following MySQL command helps to understand the syntax:

```
mysql -h <endpoint> -p 3306 -u <masteruser> -p
```

It is possible to provide a password immediately after the -p parameter; however, it is suggested to avoid using the -p parameter. It is a best practice to only provide a password during the runtime when the system prompts for it. Also remember that the number of concurrent connections that can be established with a DB instance depends on the memory available with the instance type. Based on the available memory (instance type), the number of concurrent connection limit is derived. DB instances with higher memory have a higher number of concurrent connection limit. In MariaDB, you can limit the number of concurrent connections in the max_connections parameter.

Connecting to a MySQL instance

By providing an Amazon RDS endpoint, MySQL DB instances can be connected using a standard MySQL client application or utility. Connecting to MySQL DB instances is similar to connecting to MariaDB using a MySQL command-line tool:

```
mysql -h <endpoint> -p 3306 -u <masteruser> -p
```

Amazon RDS DB instance endpoints can be obtained from the RDS console or by using `describe-db- instances` at the CLI. Optionally, it is also possible to use SSL encryption to connect to an Amazon RDS MySQL DB instance. The `--ssl-ca` parameter is used to provide a public key (`.pem`) for SSL-encrypted communication.

It is possible to pass a password immediately after the `-p` parameter. The best practice is not to provide it with the command but to provide it at runtime when prompted. The connection limit for a MySQL instance is dependent on the instance type. DB instances with higher memory have a higher connection limit. MySQL's maximum possible connection number is defined in the `max_connections` parameter.

Connecting to an Oracle instance

`SQL*Plus` is an Oracle command-line utility that can be downloaded from the Oracle website. Before connecting to the Amazon RDS Oracle DB instance, it is essential to find out the Amazon RDS endpoint, port, and protocol. When connecting for the first time, we must connect using master user credentials. Subsequently, we can create relevant users for application and maintenance purpose. It is recommended that you use separate users for applications as well as day-to-day maintenance activity rather than using the master user credentials. The following `sqlplus` command-line example helps us to understand this:

```
sqlplus 'mydbusr@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=<dns name  
of db instance>)(PORT=<listener port>))(CONNECT_DATA=(SID=<database  
name>)))'
```

The preceding parameters are defined as follows:

- `mydbuser`: This could be the master user or any other valid user.
- `PROTOCOL`: TCP is a protocol and the protocol used remains as TCP only.
- `PORT`: By default, Oracle DB can be connected to on 1521.
- `SID`: The database name, intended to connect where one instance may have more than one database.

RDS best practices

RDS best practices are as follows:

- Create an individual AWS IAM user to perform DBA tasks. Grant the minimum privileges required to perform day-to-day tasks. Remove unused access key and secret key. Have a strong password policy and rotate the password periodically.
- Before creating an RDS instance, identify Amazon RDS essential characteristics to be specified such as VPC, security group, failover or Read Replica requirement, the region and AZs to use, and storage and backup requirements.
- Before creating an RDS instance, it is recommended that you create a DB options group and DB parameter group.
- Monitor Amazon RDS instance resources, such as CPU, memory, and storage, to avoid performance bottlenecks.
- It is recommended that you keep some extra buffer in memory and a storage volume while choosing RDS instance types.
- It is recommended that you test your environment for failover, as it may take different lengths of time depending on the use case, instance type, and underlined data size.
- Amazon RDS provides an endpoint to connect to the RDS instance. The IP address beneath that endpoint may change after failover takes place. So, if an application caches the DNS IP address, set the TTL to under 30 seconds in your application environment.

Summary

- An RDS is a fully managed relational database service.
- An RDS provides fine-grained access control with the help of AWS IAM.
- An RDS does not allow the user to access the underlined host operating system.
- You can stop and start any RDS database engine.
- You can stop and start an RDS instance irrespective of whether it is in single-AZ or multi-AZ, except for SQL Server. SQL Server RDS in multi-AZ cannot be stopped.
- An RDS instance can be stopped for a maximum of seven consecutive days. After seven days, the instance is automatically restarted.

- RDS supports the Aurora, MySQL, MariaDB, PostgreSQL, Oracle, and SQL Server database engines.
- You can create a Read Replica for an RDS instances.
- Read Replica can be created for Aurora, MySQL, PostgreSQL, MariaDB, and Oracle RDS database instances.
- Amazon Aurora is a MySQL-and PostgreSQL-compatible, fully managed RDBMS.
- MariaDB is a community version of MySQL RDBMS under the GNU GPL license. It maintains a high level of compatibility with MySQL.
- Microsoft SQL Server RDS supports various versions of MS SQL Server starting from SQL Server 2008 R2 to SQL Server 2017.
- Amazon RDS supports various versions of MySQL. It is also compliant with many leading industry leading standards, such as HIPAA, PHI, and BAA.
- Oracle RDBMS versions are supported by the Amazon RDS Oracle engine – Oracle 12c, versions 12.2.0.1 and 12.1.0.2. Also, it supports Oracle 11g, version 11.2.0.4.
- Oracle RDS can be deployed within VPC and can perform point-in-time recovery and scheduled or manual snapshots.
- You can schedule an automated snapshot of an RDS instance.
- The Amazon RDS PostgreSQL engine supports various versions of PostgreSQL. It also supports point-in-time recovery using periodically or manually taken snapshots.
- Amazon allows you to specify a maintenance window for your RDS instance. This time window is used by Amazon for initiating any system maintenance tasks on the instance or underlying OSes.
- RDS allows you to monitor the instance's CPU utilization, DB connections, free storage space, free memory, and many other parameters.
- RDS retains a database snapshot, by default, for seven days. This can be changed to up to 35 days.
- RDS allows you to change DB instance type; however, changing the DB instance type requires some downtime for change to take affect.
- RDS can be configured in a multi-AZ environment for high availability.

11

AWS DynamoDB - A NoSQL Database Service

DynamoDB is an easy-to-use and fully managed NoSQL database service provided by Amazon. It provides fast and consistent performance, highly scalable architecture, flexible data structure, event-driven programmability, and fine-grained access control.

Before we go into detail about DynamoDB, we'll explore some fundamental characteristics of **Relational Database Management Systems (RDBMSes)/Structured Query Language (SQL)** and NoSQL databases. For a long time, the developer community has been working with RDBMSes and SQL. If you have used RDBMSes and SQL, you will probably want to understand the fundamental similarities and differences between the SQL and NoSQL databases.

We will cover the following topics in this chapter:

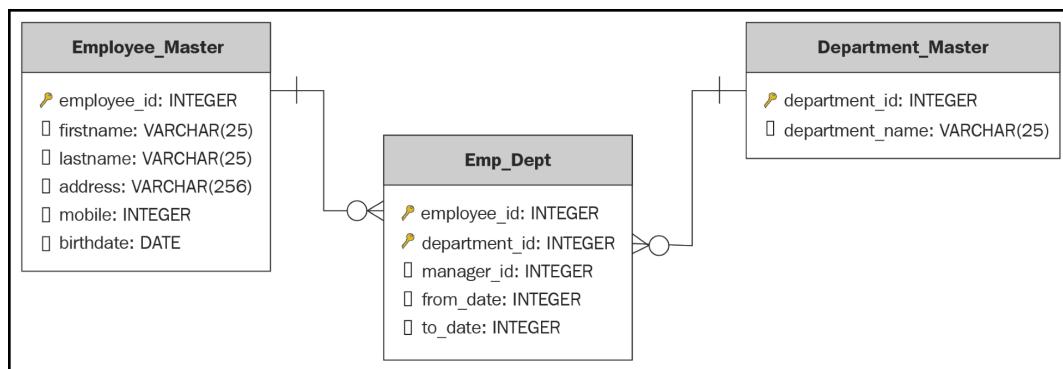
- Understanding RDBMSes
- Introducing DynamoDB
- DynamoDB components
- Read consistency model
- Naming rules and data types
- Creating a DynamoDB table
- Methods of accessing DynamoDB
- DynamoDB-provisioned throughput
- Partitions and data distribution
- Global and local secondary indexes

- DynamoDB Query and Scan
- Reading and writing an item from and to a DynamoDB table
- User authentication and access control
- DynamoDB best practices

Understanding RDBMSes

An RDBMS enables you to create databases that can store related data. A database is a collection of information that stores data in tables. A table is a collection of related data entries consisting of columns and rows.

An RDBMS enables you to create a link between these tables by establishing a relationship between them. This relational model helps in obtaining related information from multiple tables using SQL. You can see in the following diagram that there are three tables: **Employee_Master**, **Department_Master**, and **Emp_Dept**. All these tables are related with a key field that is called the **primary key**. In the following example, you can see how the **Emp_Dept** table, which provides department details for employees, is linked to the **Employee_Master** and **Department_Master** tables:



Relationship between tables in an RDBMS

In a nutshell, this is how an RDBMS stores data in tables.

Understanding SQL

SQL is a standardized language to interact with relational databases. It can execute queries against a database and retrieve data from one or more tables. It can insert, update, and delete records from database tables and perform many other database-related activities. In short, SQL can help you manage your databases.

Here's a simple example of a SQL statement:

```
Select * from Employee_Master
```

The preceding example simply retrieves all the records from the Employee_Master database. Let's look at one more simple example of a SQL statement, which retrieves related information from multiple tables:

```
Select a.employee_id, b.firstname, b.lastname, c.department_name from Emp_Dept a, Employee_Master b, Department_Master c where a.employee_id = b.employee_id and a.department_id = c.department_id
```

The preceding example retrieves related employee information from three different tables. The key to retrieving the information is in the relationship between them. The relationship is established based on key fields in the respective tables.

In a nutshell, this is how RDBMSes and SQL work.

Understanding NoSQL

A NoSQL database provides a way to store and retrieve data that is in a non-tabular format. It is also referred to as a **Non-SQL**, **Non-Relational**, or **Not-only-SQL** database. NoSQL databases are used for managing large sets of data that are frequently updated in a distributed system. They eliminate the need for the rigid schema associated with an RDBMS.

There are basically four types of NoSQL databases, which are as follows:

- **Key-value pair** databases
- **Document** databases
- **Graph** databases
- **Wide column stores**

Key-value pair databases

A key-value pair database uses a very simple data model that stores data in a pair of unique keys and the associated value. Commonly, it is used for storing time-series data, click-stream data, and application logs.

Examples of key-value pair databases are **DynamoDB**, **Riak**, **Redis**, and **Aerospike**. Examples of key-value pairs are given here:

Key	Value
Name	Abhishek
Mobile	0987654321
Address	Mumbai

DynamoDB is a key-value pair database. In subsequent sections of the chapter, we will look more deeply at key-value pair databases.

Document databases

A document database stores data elements in a structure that represents a document-like format, such as JSON, XML, or YAML. Document databases are commonly used for content management and monitoring applications. Examples of document databases are **MongoDB**, **CouchDB**, and **MarkLogic**.

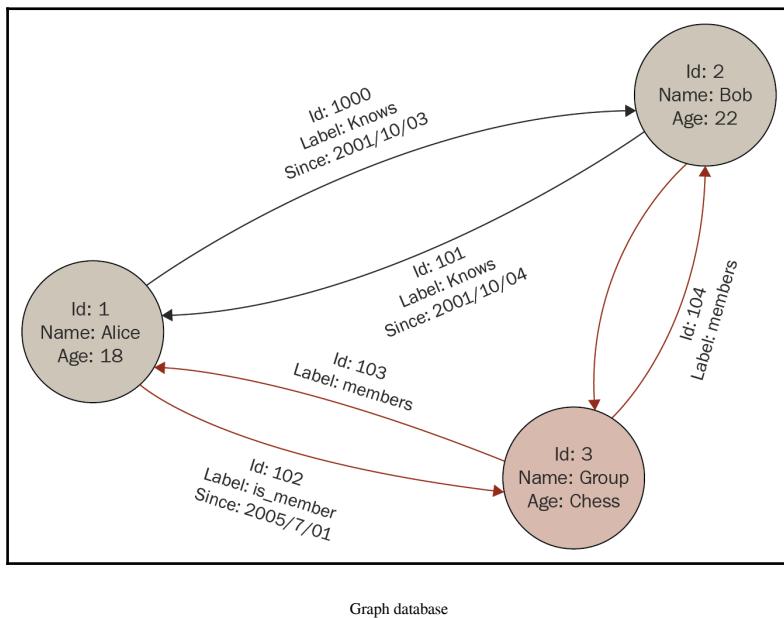
Unlike RDBMSes, the document database schema design is flexible and can combine multiple entities in a single schema. The following example shows how employee information can be stored in a MongoDB document:

```
db.employee.insert(
{
  {
    employee_id:'10001', firstname: 'Tony', lastname: 'Stark',
    birthdate: '1965-04-04'
  },
  {
    employee_id:'10002', firstname: 'Thor', lastname: 'Odinson',
    birthdate: '1983-08-11'
  },
  {
    employee_id:'10003', firstname: 'Natalia', lastname: 'Romanoff',
    birthdate: '1984-11-22'
  }
})
```

Graph databases

A graph database is a NoSQL database type that uses graph structures and stores related data in nodes. It emphasizes on the connection between the data elements to accelerate query performance. It is mainly used for storing geographical data and recommendation engines. Examples of graph databases are **AllegroGraph**, **IBM graph**, and **Neo4j**.

The following diagram shows the graph database:



Wide column databases

A wide column database is a type of NoSQL database that stores data using a column-oriented model. It is also called a **table-style database** or a **column-store database**. It stores data in a table-like structure and it can store large numbers of columns.

Wide column databases are generally used for storing data related to internet searches and other similar large-scale web applications.

Examples of wide column databases are **Cassandra**, **HBase**, **SimpleDB**, **Accumulo**, and **Hypertable**. In a wide column database, each column is stored in a separate file, as shown here:

Employee_id	Firstname	Lastname	Birthdate
10001	Tony	Stark	1965-04-04
10002	Thor	Odinson	1983-08-11
10003	Natalia	Romanoff	1984-11-22

Using NoSQL databases

A relational database stores data in one or more related tables. The relational model and tabular format minimizes data duplication. However, scaling relational databases can become very resource-intensive. In contrast, NoSQL databases store related data in a single document, which can improve accessibility and scalability. A NoSQL database sacrifices some of the query and transaction capabilities of RDBMSes for better performance and high scalability.

NoSQL is generally used for big data, advertisement technologies, gaming, mobile applications, time series data, logs, the **Internet of Things (IoT)**, and many other applications where heavy write performance, reduced latency, and a dynamic schema are required.

SQL versus NoSQL

The following are some of the key differences between SQL and NoSQL databases:

SQL	NoSQL
Database systems are termed RDBMSes.	Database systems are termed non-relational or distributed systems.
It follows a rigid and pre-defined schema model.	It uses a dynamic schema.
It stores data in a tabular form and is also known as a tabular database.	It stores data in a collection of key-value pairs, graph databases, documents, and wide column stores.
Databases can be scaled vertically.	Databases can be scaled horizontally.
It uses SQL for defining and managing data.	It uses an unstructured query language that varies from database to database.

It is best suited to complex queries.	It is not suitable for complex queries, as it does not use the relational data model.
It is not suitable for hierarchical data stores.	It is best suited for hierarchical data stores.
Oracle, SQL Server, MySQL, and PostgreSQL are SQL databases.	DynamoDB, MongoDB, Bigtable, Cassandra, HBase, and CouchDB are NoSQL databases.

Introducing DynamoDB

Amazon DynamoDB is a fully managed NoSQL database service from Amazon, providing a fast and flexible NoSQL database service for applications that need consistent and low-latency access at any scale. It supports key-value and document data models. It provides a dynamic schema model and predictable performance. DynamoDB is best suited to big data, advertisement technologies, gaming, mobile applications, time-series data, logs, IoT, and other applications where heavy write performance, reduced latency, and a dynamic schema are required.

DynamoDB allows you to store any amount of data and handle any level of user traffic. It allows you to scale up or down a table's read/write capacity without affecting the uptime and performance of the table. You can use the management console for monitoring DynamoDB resource utilization and its effective performance metrics.

It helps you reduce storage usage by automatically deleting the expired items from a table. Since it can help you automatically delete expired data, the cost for storing data can be significantly optimized.

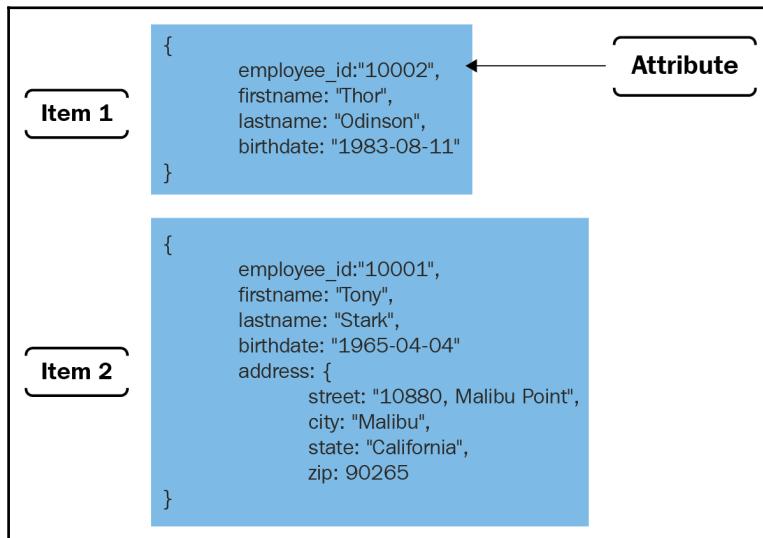
DynamoDB tables are spread across a cluster of servers that are sufficient for handling the desired throughput and the required storage for consistent and reliable performance. It stores data in **Solid State Drive (SSD)** and the data is replicated across multiple **Availability Zones (AZs)** for obtaining high availability and data durability.

DynamoDB components

There are basically three core components of a DynamoDB table: tables, items, and attributes:

- **Tables:** DynamoDB stores data in an entity called a table. A table consists of a set of data; for example, the following screenshot shows how you can store employee information in a DynamoDB table.

- **Item:** A table consists of multiple items. An item consists of a group of attributes. An item is like a record or row in an RDBMS table. In the following screenshot, you can see the data of two employees. Each employee's data represents an item in DynamoDB.
- **Attributes:** An item in a table consists of multiple attributes. An attribute is the basic data element of an item. It is similar to a field or a column in an RDBMS. However, unlike the case in an RDBMS, attributes in a table item can have sub-attributes. You can see that in the following screenshot, the address attribute is further broken down into multiple attributes for representing specific data:



As you can see in the preceding screenshot, the first record in the table does not contain an address, whereas the next record has an address and its subset attributes in the record. This shows the flexibility in the schema of a DynamoDB table. You can have different attributes in subsequent records based on need.

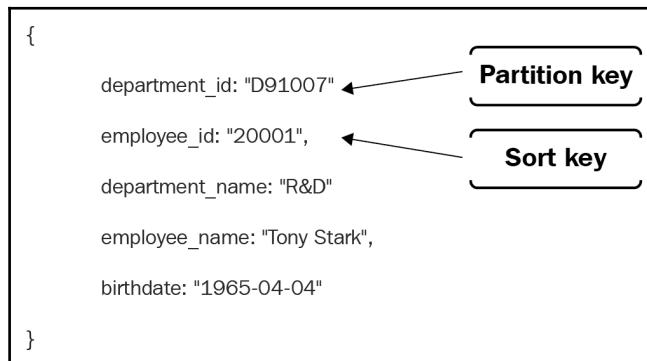
Primary key

While creating a DynamoDB table, you need to specify the table name and the primary key of the table. It is mandatory to define a **primary key** in the DynamoDB table. A primary key is a mechanism to uniquely identify each item in a table. A primary key does not allow two items with the same key value in a table. There are two types of primary key:

- **Partition key:** This is a simple primary key that is composed of a single attribute named a partition key. DynamoDB partitions the data in sections based on the partition key value. The partition key value is used as an input to internal hash functions, which determines in which partition the data is stored. This is the reason the partition key is also called the *hash* key. No two items in a table can have the same partition key. Since the data is divided into partitions based on its partition key value, data retrieval becomes faster. In the following screenshot, you can see that **employee_id** is an example of a simple primary key. You can rapidly access employee information from a table by providing the **employee_id**.
- **Partition key and sort key:** The partition key and sort key are composed of two attributes and that's why they are also called a **composite primary key**. As the name suggests, the first attribute of the key is the partition key and the second attribute is the sort key. Just like the partition key, the composite key also uses the partition key as an input to an internal hash function. This hash function determines the place of an item in a partition. The partition is a physical store, which is internal to DynamoDB, for arranging the item to get the best possible performance from the table.

In a partition, items are ordered based on a sort key value. Thus, the partition key determines the first level of sort order in a table, whereas the sort key determines the secondary sort order of the data stored in a partition. A sort key is also called a **range** key.

A table with both a partition key and a sort key can have more than one item with the same partition key value; however, it must have a different sort key. In the following example, you can see that `department_id` is a partition key and `employee_id` is a sort key. A department can have multiple employee records, which leads to repeating the `department_id`; however, `employee_id` cannot repeat with the same department name:



Partition key and sort key

In a nutshell, there can be two types of primary keys:

- A single-attribute partition key
- A double-attribute partition key and sort key

Secondary indexes

DynamoDB allows you to create **secondary indexes** on a table. This is an alternate way to query table data in addition to querying it using the primary key. It is not necessary to use indexes, but using secondary indexes provides some flexibility in querying the data.

There are two types of secondary indexes:

- **Global Secondary Index (GSI):** This consists of a partition key and a sort key, which have to be different than the primary keys defined on the table.
- **Local Secondary Index (LSI):** This uses the same partition key as the table but uses a different sort key.

DynamoDB allows you to create 20 GSIs and five LSIs on a table:

- Every index is associated with a table, called the **base table**, for the index.
- DynamoDB automatically maintains the indexes. Whenever data is added, updated, or deleted from the base table, DynamoDB adds, updates, or deletes the corresponding item in the related indexes.
- While defining the index, you can choose which attributes are copied to the index. At a minimum, DynamoDB projects at least the key attributes from the table.

DynamoDB Streams

DynamoDB Streams provides an optional feature that can capture data modification events whenever a DynamoDB table is changed. The event data is captured in the stream in near-real time in chronological order as the event occurs. Each of the events is recorded by a stream record.

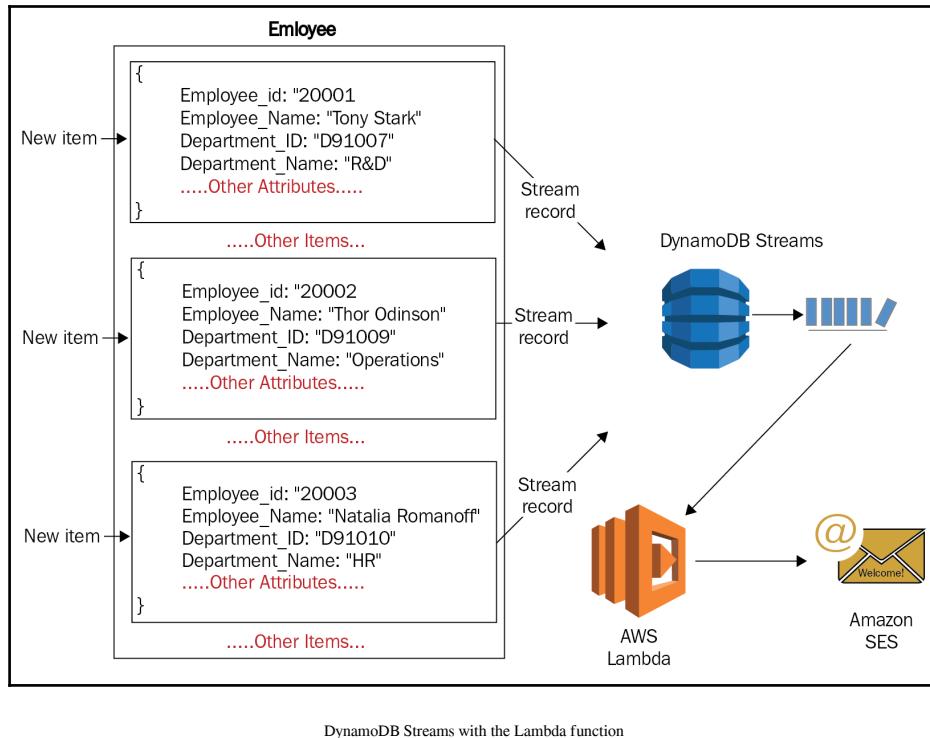
When you enable a stream on a DynamoDB table, it writes a stream record as and when one of the following events occurs:

- When a new item is added to a table, DynamoDB Stream receives a complete image of the item with all the item attributes.
- When an existing item is modified from a table, DynamoDB Stream receives before and after values of all the attributes that are updated during the operation.
- When a delete operation is performed on a table item, the stream receives the entire item before it's deleted.

A stream record consists of the name of the table, the timestamp when the event occurs, and other metadata. A stream record can last for 24 hours; after that, it is automatically deleted from the stream.

You can also use **AWS Lambda** to create a trigger along with DynamoDB Streams. A Lambda function can execute whenever a defined event occurs in a stream. Let's consider the employee table used in the previous examples. When a new employee record is created, you want to send an email to all the employees in the department to welcome the new person. In these cases, you can enable a stream on the employee table and then associate the stream with a pre-defined Lambda function, which sends an email to all the employees in a department where a new employee joins.

The Lambda function executes whenever there is a new record available in the stream; however, it processes only new items added to the employee table. The Lambda function invokes the Amazon **Simple Email Service (SES)** for sending emails to the users in the department. The following diagram illustrates this scenario:



Apart from triggers, DynamoDB Streams can be used for data replication within a specific region or across multiple regions. They can also be used for **materialized views** of DynamoDB tables. A materialized view is a database object that contains the result of a query, just like **views** in RDBMS terminology. DynamoDB Streams can also be used for data analysis with Kinesis-materialized views.

Read consistency model

Amazon provides DynamoDB in multiple AWS regions across the globe. All of these regions are independent and physically isolated from one another. If you create a DynamoDB employee table in the us-east-1 region and similarly create another employee table in the us-west-2 region, these tables are two separate and isolated tables. An AWS region consists of multiple AZs. Each of the AZs are isolated from failures in any of the AZs in a region. Amazon provides an economical and low-latency network connection between all the AZs in a region.

Whenever you write data to a DynamoDB table, AWS replicates this data across multiple AZs to provide high availability. After writing data to a DynamoDB table, you get an `HTTP 200` response; `HTTP 200 (OK)` indicates that the data has safely updated to all the replicated copies stored in different AZs. AWS provides two types of read consistency models:

- **Eventually consistent reads:** While reading data from a DynamoDB table using eventually consistent reads, the result you may get might not reflect any recently completed write operations. Since the data is stored in multiple AZs, it takes a few seconds to synchronize the data in multiple locations. Eventually, the consistency of data is achieved. In such cases, if you repeat your read operation after a short while, it returns the latest copy of the data.
- **Strong consistent reads:** If you opt for a strongly consistent read on a DynamoDB table, it provides the response with the most up-to-date data. It reflects changes from all prior write operations that were successful. While working with strongly consistent reads, if there is any outage or delay in the network, the data may not be available immediately.
By default, DynamoDB uses eventually consistent reads. If you need to use strongly consistent reads, you need to specify this while creating the table. Read operations, such as `GetItem`, `Query`, and `Scan` provide a `ConsistentRead` parameter. DynamoDB uses strongly consistent reads when this parameter is set to true.

These models are based on the mechanism of how soon the data is replicated across the AZs.

Naming rules and data types

DynamoDB supports a number of data types. This section describes these data types as well as the DynamoDB naming rules.

Let's start by understanding naming rules.

Naming rules

Each table, attribute, and any other object in DynamoDB should have a name. All the names that you use in DynamoDB should be concise and meaningful. For example, a table can be named `Employee`, `Department`, `Books`, and so on. Just like these names, whatever name you use should be self-explanatory.

Here are some of the naming rules for DynamoDB:

- Names are case-sensitive and must be encoded in UTF-8.
- A table name and an index name may range from 3 to 255 characters.
- Names may contain only the following characters:
 - a-z.
 - A-Z.
 - 0-9.
 - _ (underscore).
 - - (dash).
 - . (dot).
- Attributes can have a number of characters, from 1 to 255.



There are a number of special characters and reserved words in DynamoDB. These reserved words can be found at <http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ReservedWords.html>.

Apart from these reserved words, the following characters have special meaning in DynamoDB:

- # (hash)
- : (colon)

Although DynamoDB allows you to use this list of reserved names and special characters, it is recommended you do not use them for naming purposes in DynamoDB.

Data types

DynamoDB supports a number of data types for attributes in a table. These data types can be categorized into three parts:

- Scalar types
- Document types
- Set types

You need to specify names and respective data types for each of the primary key attributes while creating a DynamoDB table. In addition to that, each primary key, be it a partition key or sort key, must be defined as one of the following scalar data types—string, number, or binary.

DynamoDB is a schemaless NoSQL database. While creating a table, you need to define the primary key and its respective data types. Apart from that, you do not need to define other attributes and their data types while creating a table. In contrast, while creating a table in RDBMS, you need to specifically define a schema with field names and data types. DynamoDB has a flexible schema that allows you to directly store the data without defining the schema. You just need to define the primary key; the rest of the attributes can be taken care of dynamically as you store the data in the table.

The following section describes each of these data types along with an example format.

Scalar data types

The **scalar** data type represents one value. It includes data types such as number, string, binary, Boolean, and null. We will look at the characteristics of each of the data types:

- **String:** The following table depicts the characteristics of the string data type:

Encoding	Unicode with UTF-8 binary encoding
Length	Greater than 0 and less than 400 KB
Partition key length	Maximum of 2,048 bytes
Sort key length	Maximum of 1,024 bytes
Usage	To represent a string, alternatively date, and timestamp in string format

- **Number:** The following table depicts the characteristics of the number data type:

- **Binary:** The following table depicts the characteristics of the binary data type:

Can store	Binary data, such as images, compressed text, and encrypted data
Length	Greater than 0 and less than 400 KB
Binary attribute as partition key	Maximum length of 2,048 bytes
Binary attribute as sort key	Maximum length of 1,024 bytes
Supported encoding	Application must send the data in a Base64-encoded format

- **Boolean**: Boolean is also a scalar data type and it can store as either *true* or *false* values.
 - **Null**: A null data type indicates an undefined or unknown state of the data.

Document types

A **document data type** contains a complex structure with nested attributes. Examples of these structures are **JSON** and **XML**. DynamoDB supports two document data types—**List** and **Map**. You can create a complex data structure by nesting these data types up to 32-levels deep. DynamoDB does not limit the number of values it can store in a list or map data types; however, a table item must not exceed a total item size of 400 KB. DynamoDB does not support empty scalar data types or set data types; however, it does support empty list and map data types. The document data types are explained in detail here:

- **List:** A list data type can store a collection of values in square brackets [. . .]. You can compare the list data types with a JSON array. You can store any data types within the list, and all elements in the list may or may not be of the same data type.

- Here's a simple list example:

```
TechGiants: ["Amazon", "Apple", "Google", "Facebook"]
```

- Here's a multiple data type list example:

```
DeckOfCards: ["Ace", 2, 3, 4, 5, 6, 7, 8, 9, 10,  
"Jack", "Queen", "King"]
```

- **Map:** A map data type attribute can consist of a collection of name-value pairs. This collection of values can be in any order. Map values are stored in curly brackets: { . . . }. You can compare map with a JSON format. DynamoDB does not restrict you from storing any data type in a map element. You can also store multiple data types in an element. They do not need to be the same size.

Generally, maps are used for storing JSON documents in DynamoDB. The following code block is an example of a map with a nested list:

```
{  
employee_id: "D10007",  
employee_name: "Tony Stark",  
address: [  
    home:  
    {  
        street: "10880, Malibu Point",  
        city: "Malibu",  
        state: "California",  
        zip: 90265  
    },  
],  
}
```

```
        office:  
        {  
            street: "890, Fifth Avenue, Manhattan",  
            city: "New York",  
            state: "New York",  
            zip: 10019  
        }  
  
    }  
]
```

Set types

A **set** data type contains a group of scalar values. The set types supported by DynamoDB are string set, number set, and binary set.

When you use a set type, all the values within a set must be of the same data type. If you declare a set of type string, the set can contain only string values. If you declare an attribute of type number, the set can contain only number values within the set.

DynamoDB does not restrict you on the number of values within a set; however, the item containing the set data cannot exceed the DynamoDB item size limit of 400 KB. The set must contain unique values. DynamoDB does not preserve the order of the set values.

Considering that, your application should not rely on any static element order in a set; also, it is important to note that DynamoDB does not support an empty set.

Examples of string, number, and binary sets, respectively, are as follows:

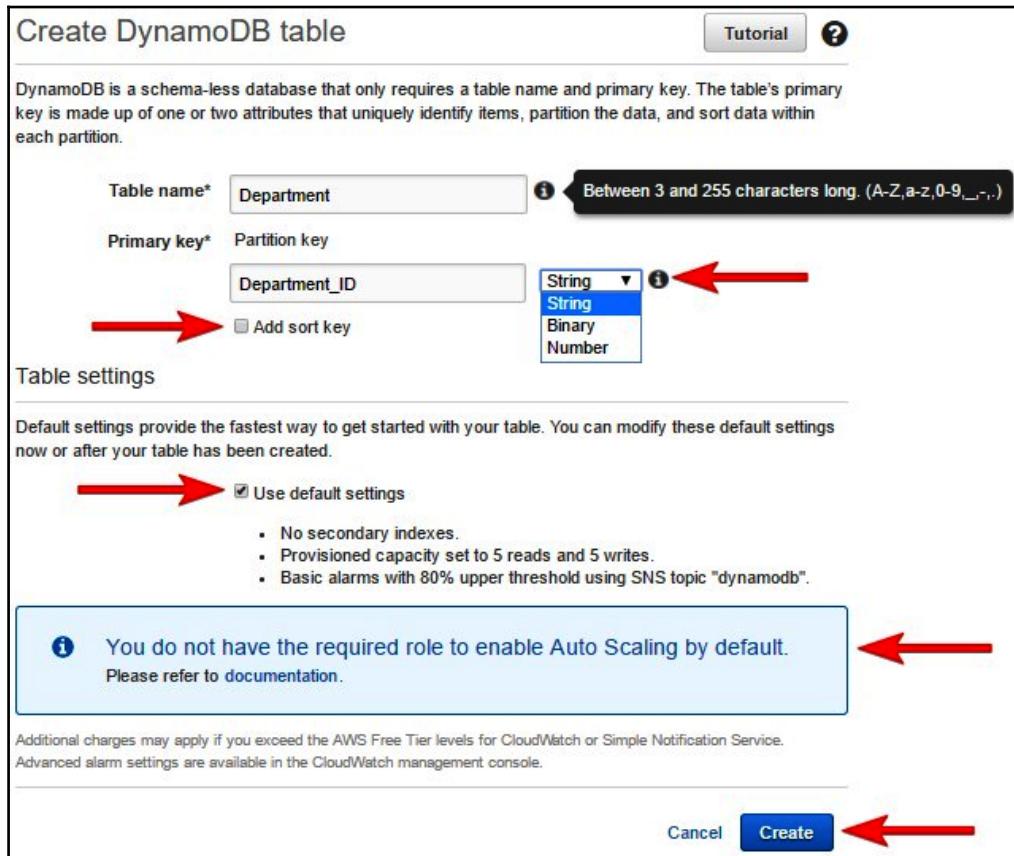
```
["Amazon", "Google", "Microsoft", "Facebook", "Apple"]  
[-3.14159, 0, 1, 1.4142, 2, 4, 8, 16, 32]  
["eNrLz0sFAAKRAUM=", "eNorKc8HAAK8AVs=", "eNoryShKTQUABm4CGQ=="]
```

Creating a DynamoDB table

After going through the core components of a DynamoDB table, it's time to create a table. This section describes the process of creating a DynamoDB table:

1. Log into your AWS account and open the DynamoDB console at <https://console.aws.amazon.com/dynamodb/>.
2. Click the **Create table** button.

3. In the **Create DynamoDB table** dialog, you can choose various options, as shown:



Creating a DynamoDB table

You can use following guidelines to complete *step 3*:

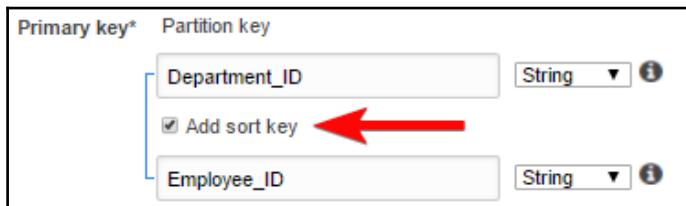
1. Enter the desired table name, as shown in the preceding screenshot. The **Table name** can be between 3 and 255 characters long.
2. Enter the partition key name and select the key type from the drop-down menu. Any primary key can only be a scalar data type, that is, **String**, **Binary**, or **Number**.

3. Check the **Add sort key** checkbox, if required, and enter the sort key name and data type. This is required only if you want to create a composite primary key with a combination of a partition key and a sort key.
 4. With the default settings, Amazon creates a table without any secondary indexes and provisions a five read and five write capacity. It also creates a basic alarms with 80% upper threshold using SNS topic dynamodb. You can customize these settings by unchecking the **Use default settings** checkbox.
 5. As you can see, there is a warning saying **You do not have the required role to enable Auto Scaling by default**. This warning comes only if you do not have this role in your IAM roles. You can safely ignore this, as DynamoDB creates a new role when you create a table with default settings. In the advanced settings, there is an option to create a new Auto Scaling role for DynamoDB or to use an existing one. This is explained in the subsequent *Auto Scaling* section.
4. Click **Create**, after choosing the appropriate options.

Now let's look at some of the advanced settings in the table creation process.

Adding a sort key while creating a DynamoDB table

You can add a sort key while creating a table by selecting the **Add sort key** checkbox, as shown here:



Adding a sort key

Using advanced settings while creating a DynamoDB table

With the default setting, Amazon creates a table without any secondary indexes and provisions a five read and five write capacity. It also creates a basic alarms with 80% upper threshold using the SNS topic dynamodb. You can customize these settings by unchecking the **Use default settings** checkbox. Let's look at the options in the advanced settings.

On the create table screen, uncheck **Use default settings**. It displays the screen shown here:

The screenshot shows the 'Table settings' page for creating a new DynamoDB table. The 'Use default settings' checkbox is unchecked. Several sections are highlighted with red arrows:

- + Add index**: Points to the 'Secondary indexes' section.
- Read/write capacity mode**: Points to the 'Provisioned capacity' section where 'Provisioned (free-tier eligible)' is selected.
- Auto Scaling**: Points to the 'Auto Scaling' section.
- IAM Role**: Points to the 'IAM Role' section which includes a note about IAM permissions for Auto Scaling.
- Encryption At Rest**: Points to the 'Encryption At Rest' section where 'DEFAULT' is selected.

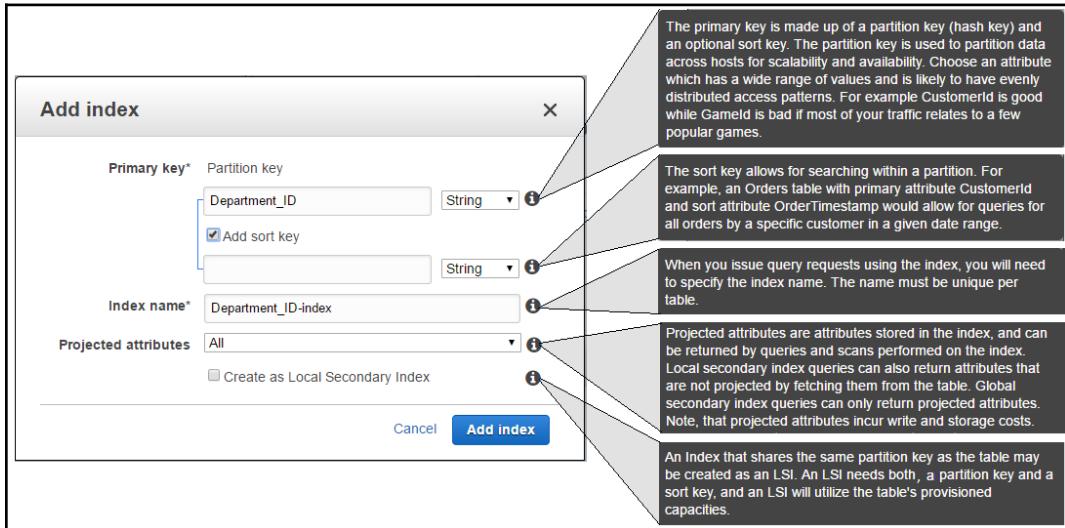
Creating a table screen – advanced settings

As you can see in the advanced settings, there is a provision to create secondary indexes and set up Auto Scaling, and there's also an option to create a **DynamoDB Autoscaling Service Linked Role** or to choose an existing role for Auto Scaling permissions. Let's look at each of these options.

Creating secondary indexes

The create table screen allows you to create secondary indexes while creating a table. For creating secondary indexes, click **Add index** in the **Table settings** screen, as shown next.

Clicking **Add index** brings up the following screen with further options:



Creating a secondary index

As shown, you can enter the **Partition key** name and optionally a sort key name based on the requirement. The **Index name** is automatically populated with the **Partition key** name and the **-index** suffix. You can change the index name, if required. The index name must be unique for each table.

Projected attributes are attributes that are stored in the index. While creating an index, you can specify which attributes you want to add to the index. You can select these projected attributes from the drop-down box. There are three options in the drop-down box. You can either choose **All** attributes, **Keys only**, or you can select **Include** and add specific attributes that you want to add to the index. Whatever attributes you choose here are returned by a query and a scan is performed on the index.

Read/write capacity mode

At the time of writing this book, Amazon DynamoDB offers two read/write capacity modes. They can be configured at the time of creating a table or later. The on-demand mode is a flexible billing option and can support thousands of requests per second without capacity planning. On other hand, the provisioned mode requires you to configure the maximum number of read/write operations per second. It doesn't scale automatically and incurs fixed charges to the AWS account.

Provisioned capacity

If you choose to customize the default settings, you can configure the **Provisioned capacity** for the table. Provisioned capacity settings are disabled if Auto Scaling is enabled. For a custom read or write capacity, you need to disable the **Read capacity** and **Write capacity** checkboxes. After disabling the Auto Scaling, you can choose the specific **Read capacity** units and **Write capacity** units. You can refer to the *Using advanced settings while creating a DynamoDB Table* section. The table's read and write capacity is automatically taken care of if you enable Auto Scaling.

Auto Scaling

You can configure the Auto Scaling options from **Table settings**. You can selectively choose Auto Scaling for **Read capacity** and/or **Write capacity** as per your needs.

There are three options you need to configure in order to use Auto Scaling:

- **Target utilization:** The default value for **Target utilization** is 70 percent. You can change this value based on your needs. DynamoDB automatically scales up the read/write capacity of the table, if the utilization goes beyond the target utilization percentage configured here. The table capacity scales up to the **Maximum provisioned capacity** configured. You can individually specify **Target utilization** thresholds for read as well as write operations.
- **Minimum provisioned capacity:** While creating the table, you can specify the **Minimum provisioned capacity** for a table. Irrespective of utilization, DynamoDB provisions the minimum read and/or write capacity as configured here. DynamoDB does not scale itself down beyond **Minimum provisioned capacity**. You can separately specify **Minimum provisioned capacity** for read as well as write operations.

- **Maximum provisioned capacity:** While using Auto Scaling on a table, you can configure the maximum scaling capacity of a table. DynamoDB does not scale beyond this capacity during Auto Scaling events. You can separately specify the **Maximum provisioned capacity** for read as well as write operations:

The screenshot shows the 'Table settings' section of the AWS DynamoDB console. It includes sections for Secondary indexes, Read/write capacity mode (set to Provisioned (free-tier eligible)), Provisioned capacity (with Read capacity units set to 5 and Write capacity units set to 5), and Auto Scaling. The Auto Scaling section has 'Read capacity' checked. In the IAM Role section, 'DynamoDB AutoScaling Service Linked Role' is selected. The Encryption At Rest section shows 'DEFAULT' selected. Red arrows highlight several key areas: the 'Read capacity' checkbox under Auto Scaling, the IAM role selection, and the 'Write capacity' checkbox under Auto Scaling.

Creating a table – Auto Scaling settings

You can enable or disable Auto Scaling for **Read capacity** and **Write capacity** separately, as shown. Unchecking **Read Capacity** disables Auto Scaling for read capacity. Similarly, unchecking **Write Capacity** disables Auto Scaling for write capacity.

For Auto Scaling the table capacity, DynamoDB requires permission. As shown in the previous screenshot, you need to either select **DynamoDB AutoScaling Service Linked Role** or choose **Existing role with pre-defined policies**. If you select **Existing role with pre-defined policies**, you need to specify an existing role name that carries sufficient permissions for Auto Scaling the DynamoDB table.

For more details on creating a role for DynamoDB Auto Scaling, you can refer to <http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/AutoScaling.CLI.html#AutoScaling.CLI.CreateServiceRole>.



Encryption at rest

Amazon DynamoDB also supports **encryption at rest**. The default mode is server-side encryption using the AWS-owned **Customer Master Key (CMK)**. It is owned by DynamoDB and no additional charges apply to the AWS account. Optionally, you can also use server-side encryption using the AWS-managed CMK. The key is managed by AWS **Key Management Service (KMS)** and charges may be applied. In the DynamoDB table, whether encrypted or not, the same code can access the items with the same millisecond response time without changing the code.

Methods of accessing DynamoDB

Amazon provides a DynamoDB console, CLI, and API interface to access DynamoDB resources.

Let's look at each of these interfaces.

DynamoDB console

Amazon provides a DynamoDB AWS Management Console. You can access the console here: <https://console.aws.amazon.com/dynamodb/home>.

Here's what you can do with the DynamoDB AWS Management Console:

- Access the DynamoDB dashboard for monitoring recent alerts, the total provisioned capacity of tables, health of the service, and latest news on DynamoDB.
- Create, update, and delete tables. It also provides a capacity calculator that can help you estimate the capacity units you may need based on the information you provide.
- Manage DynamoDB Streams.
- See items stored in a table, add new items, update existing items, and delete items.
- Manage **Time To Live (TTL)** for items stored in a table. TTL is defined for automatically deleting an item from the table when it expires.
- Query items in a table and perform a scan on a table.
- Create and view alarms for monitoring a table's capacity usage.
- View in real time a table's top monitoring metrics graph, directly from CloudWatch, on to the DynamicDB console.

- Change the provisioned capacity of a table and create and delete GSIs.
- Create triggers that can connect DynamoDB Streams to a Lambda function and add tags to your DynamoDB resources for better identification and categorization of resources.
- Purchase reserved capacity.

The console also provides a number of navigation tabs. The following list provides a quick reference to each of the tabs available on the console:

Navigation tab	Description
Overview	Displays table details and manages streams and TTL
Items	Manages table items and executes queries and scans against table and indexes
Metrics	Views and monitors CloudWatch metrics related to DynamoDB
Alarms	Views and manages CloudWatch alarms
Capacity	Views and updates provisioned capacity of a table
Indexes	Views and manages GSIs
Triggers	Manages triggers that can connect DynamoDB Streams to a Lambda function
Access control	Manages fine-grained access control and configures web identity federation
Tags	Adds tags to resources for better identification and categorization of resources

DynamoDB CLI

AWS provides a **Command-Line Interface (CLI)** to manage AWS services using the command line. You can use the CLI for a number of purposes, such as creating a script to automate a task or creating a table and performing many other DynamoDB tasks using utility scripts.



If you have not already set up AWS CLI, you can follow the instructions for downloading and configuring it here: <http://aws.amazon.com/cli>.

Working with the DynamoDB CLI is very simple, if you understand the basics. Here's the syntax for working with the DynamoDB CLI:

```
aws dynamodb <operation-name> <--parameters-name> <parameter-value>
... <-- parameters-name> <parameter-value>
```

The following command creates a DynamoDB table named `employee` with the `Employee_ID` and `Employee_Name` attributes. It also creates a partition key on the `Employee_ID` attribute:

```
aws dynamodb create-table \
--table-name employee \
--attribute-definitions \
    AttributeName=Employee_ID,AttributeType=S \
    AttributeName=Employee_Name,AttributeType=S \
--key-schema AttributeName=Employee_ID, KeyType=HASH \
--provisioned-throughput ReadCapacityUnits=1,WriteCapacityUnits=1
```

Similarly, the following commands add new items to the `employee` table:

```
aws dynamodb put-item \
--table-name employee \
--item \
    '{"Employee_ID": {"S": "10001"}, "Employee_Name": {"S": "Vipul \
Tankariya"}, "Country": {"S": "India"}}' \
--return-consumed-capacity TOTAL

aws dynamodb put-item \
--table-name employee \
--item \
    '{"Employee_ID": {"S": "10002"}, "Employee_Name": {"S": "Bhavin \
Parmar"}, "Country": {"S": "India"}}' \
--return-consumed-capacity TOTAL

aws dynamodb put-item \
--table-name employee \
--item '{ \
    "Employee_ID": {"S": "10003"}, \
    "Employee_Name": {"S": "Gajanan Changadkar"}, \
    "Country": {"S": "India"} }' \
--return-consumed-capacity TOTAL
```

Sometimes, the complex JSON format may create problems on the command line. AWS provides a way to handle this format using a file as an argument on the command line. The following example shows how you can run the CLI with JSON file arguments.

Let's assume that all the command line arguments are stored in a JSON file called `condition-file.json` for creating items in the `employee` table:

```
aws dynamodb query --table-name employee --key-conditions  
file://condition- file.json
```

AWS provides a number of commands to work with DynamoDB.



You can refer to <http://docs.aws.amazon.com/cli/latest/reference/dynamodb/index.html> for more commands.

Working with APIs

Apart from the AWS Management Console and CLI, AWS also provides APIs to work with DynamoDB. These APIs can be used to develop applications that can manage various DynamoDB operations. For using APIs, you need to install AWS SDKs. AWS provides SDKs for a number of programming languages, such as Java, JavaScript in the browser, .NET, Node.js, PHP, Python, Ruby, C++, Go, Android, and iOS.

The following table describes where you can start working with these SDKs:

Language	Reference URL
Java	https://aws.amazon.com/sdk-for-java
JavaScript in the browser	https://aws.amazon.com/sdk-for-browser
.NET	https://aws.amazon.com/sdk-for-net
Node.js	https://aws.amazon.com/sdk-for-node-js
PHP	https://aws.amazon.com/sdk-for-php
Python	https://aws.amazon.com/sdk-for-python
Ruby	https://aws.amazon.com/sdk-for-ruby
C++	https://aws.amazon.com/sdk-for-cpp
Go	https://aws.amazon.com/sdk-for-go
Android	https://aws.amazon.com/mobile/sdk/
iOS	https://aws.amazon.com/mobile/sdk/

DynamoDB provisioned throughput

DynamoDB provides the Auto Scaling feature for automatically scaling the read and write capacity of a table; however, if you do not use it, you need to manually handle the throughput requirement of your table. DynamoDB measures the throughput capacity using read and write capacity units.

Read capacity units

DynamoDB processes the read operations based on the type of read consistency used. Using one read capacity unit, DynamoDB can process one strongly consistent read per second. In the same line, DynamoDB can process two eventual consistent reads per second using one read capacity unit. Using one read capacity unit, DynamoDB can process an item of up to 4 KB in size. If the item size is more than 4 KB, it requires an additional read capacity unit to process it. In short, the item size and consistency model determine the total number of read capacity units required to process it as shown here:

- One read capacity unit = one strongly consistent read.
- One read capacity unit = two eventual consistent reads.
- One read operation can process an item of up to 4 KB in size.
- If an item is more than 4 KB in size, it requires additional read operations.
- If an item is less than 4 KB in size, it still requires one read capacity unit.

Write capacity units

Using one write capacity unit, DynamoDB can process one write per second and write an item of a maximum of 1 KB in size. If the size of the item is greater than 1 KB, it requires additional write capacity units. In short, the number of write capacity units required to process an item depends up on the size of the item:

- One write capacity unit = one write operation of up to 1 KB in size.
- If an item is greater than 1 KB, it requires additional write capacity units.
- If an item is less than 1 KB in size, it still requires one write capacity unit.

Calculating table throughput

If you create a table with a throughput of five read capacity units and five write capacity units, then the following conditions are true:

- It can perform a strongly consistent read of up to 20 KB per second:

5 read capacity units \times 4 KB

- It can perform an eventual consistent read of up to 40 KB:

5 read capacity units \times 4 KB \times 2

- It can write 5 KB per second:

5 write capacity units \times 1 KB

- When you manually configure the throughput of a table, it determines the highest amount of capacity an application can utilize from a table or associated index. If your application consumes more throughput than configured in the provisioned throughput settings, application requests start throttling. This can either crash the application or give a lackluster performance.

Examples for understanding throughput calculation

Let's consider a couple of examples to understand the throughput calculation.

Example 1

You have an application that requires reading 15 items per second. Each item is 3 KB in size. If the application requires strongly consistent reads, what read capacity is required to address this need?

One read capacity unit can process one strongly consistent read of up to 4 KB item in size. If the item size is less than 4 KB, it still requires one read capacity to process the read operation.

Let's formulate this understanding:

```
Read throughput = (item size rounded-up in multiples of 4 KB)/4 KB x  
number of items  
Read throughput = 4/4 x 15 (item size is 3 KB, which is rounded-up to  
4 KB)  
Read throughput = 1 x 15 Read throughput = 15
```

The answer is 15 read capacity units.

In the same example, if the requirement changes to an eventual consistent read, then you just need to divide the result by 2, as one read capacity unit can process two eventual consistent reads.

Example 2

You have an application that requires reading 80 items per second. Each item is 5 KB in size. If the application requires using strongly consistent reads, what read capacity is required to address this need?

In this example, the item size is 5 KB, which is greater than 4 KB. We need to round it up to a multiple of 4 KB, which is 8 KB in this case. Remember, if an item is more than 4 KB in size, it requires additional read capacity. This is the reason we need to always use multiples of 4 while calculating the read throughput.

Let's add the values in the formula for this example:

```
Read throughput = (item size rounded-up in multiples of 4 KB)/4 KB x  
number of items  
Read throughput = 8/4 x 80  
= 2 x 80  
= 160
```

The answer is 160 read capacity units.

Similarly, if the requirement changes from strong consistent read to eventual consistent read, you need to divide the answer by 2.

Example 3

You have an application that requires reading 60 items per second. Each item is 3 KB in size and the application requires using eventual consistent reads. What read capacity is required to address this need?

Here's the formula for calculating the read capacity for eventual consistency:

```
Read throughput = ((item size rounded-up in multiples of 4 KB) / 4 KB  
x number of items)/2  
= (4 / 4 x 60) / 2  
= (1 x 60) / 2  
= 60/2  
= 30
```

The answer is 30 read capacity units.

Example 4

You have an application that writes 10 items per second, with each item being 8 KB in size. How many write capacity units are required to address this need?

Remember, one write capacity unit = one write operation of up to 1 KB in size. Here's the formula that can help us calculate the required write throughput:

```
Write Throughput = (item size rounded-up in multiples of 1 KB)/1 KB x  
number of items  
= Item size rounded-up in multiple of 1 KB x number of items  
= 8 x 10  
= 80
```

The answer is 80 write capacity units.

Partitions and data distribution

Table partitioning is a mechanism to segregate a large table into smaller, more manageable parts without creating a separate table for each part. A partitioned table physically stores data in groups of rows. These groups of rows are called partitions. You can access and maintain each partition separately.

DynamoDB also manages data in partitions. DynamoDB uses SSDs for storing data and automatically replicates data across multiple AZs in an AWS region. DynamoDB automatically manages partitions; you, as a consumer, do not need to manage the partitions.

When creating a table, DynamoDB allocates a sufficient number of partitions to the new table so that it can handle any provisioned throughput needs. However, DynamoDB can allocate additional partitions to a table in certain situations. The following are the scenarios when DynamoDB allocates additional partitions:

- If a table's provisioned throughput goes beyond what the existing partitions can handle
- If an existing partition consumes allocated storage space and more storage space is required

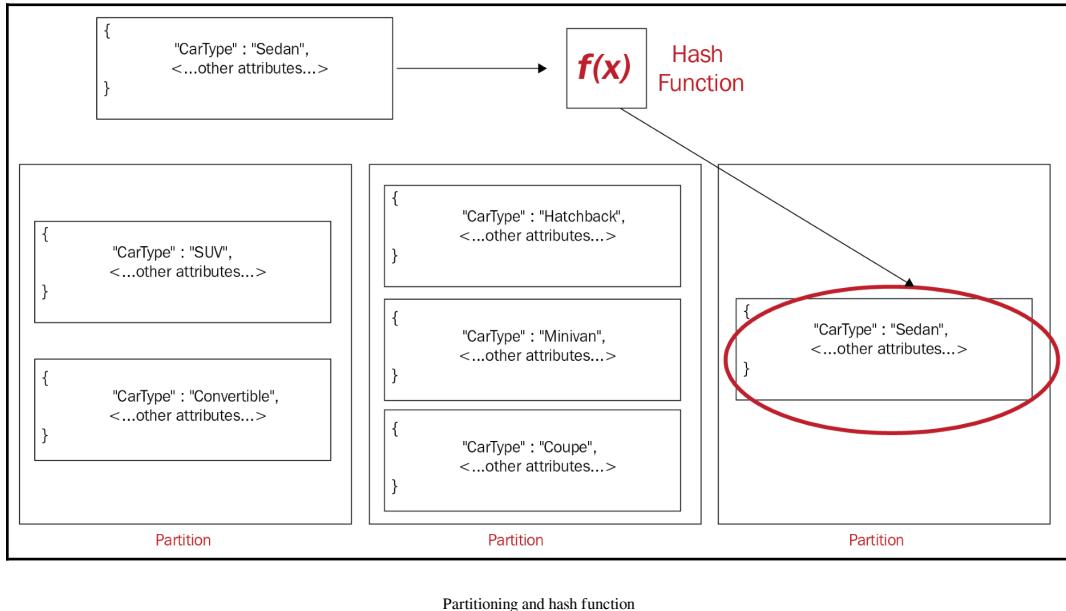
Partition management tasks are performed automatically in the background without affecting the provisioned throughput of a table. It is also important to note that GSIs are also segregated in partitions. The data in an index is stored separately from the base data of a table. Index partitions and table partitions act in a similar manner in DynamoDB.

Data distribution – partition key

As we have seen earlier in this chapter, DynamoDB allows you to create a primary key either with a single attribute partition key or with a composite key consisting of a combination of a partition key and sort key. If we create a table with only the partition key, each item in the table is retrieved based on the partition key's value.

DynamoDB uses an internal hash function for writing an item to the table. The partition key's value acts as an input to the hash function. Based on the partition key's value, the hash function determines the target partition for storing the item.

It is required to provide the partition key value for reading an item from the table. This value is used in the hash function to locate the partition where the item can be found:



Let's look at an example to understand how the items are stored in a partition. The preceding diagram describes the details of a `Cars` table. The table spans multiple partitions. The primary key for the table is `CarType`. For simplifying the concept, only the primary key attribute is included. While storing data in a table, DynamoDB uses an internal hash function to determine the target partition for storing a new item. In this example, the hash function determines the target partition based on the hash value of `CarType`, which is `Sedan`. Here, it is important to understand that the items are not stored in a sorted order. The location of each item is determined based on the value of the partition key.

The preceding example with `CarType` is given for simplifying the data distribution concept in the DynamoDB partition. While designing a DynamoDB table, you should choose a partition key that has a large number of distinct values. If the partition key values are similar, data may end up getting stored in some specific partitions and may not give optimum performance.

Let's try to understand this scenario with an example.

If you're creating a `Vendor` table, `Vendor_ID` is a good candidate for the partition key; however, you need to ensure that the values in `Vendor_ID` are distinct.

If you choose values in `Vendor_ID` as `V0001`, `V0002`, `V0003`, and so on, it works, and items are distributed in multiple partitions depending up on the table size. However, the best way to optimize partitioning is to use more distinct values, such as `FMG001`, `ITV001`, and `SRV001`. In this example, `Vendor_ID` includes the vendor type, such as **Fast-Moving consumer Goods (FMG)**, **IT Vendor (ITV)**, and **Service Vendor (SRV)**. These approaches create more distinct values in the table and distribute data optimally in more partitions.

Data distribution – partition key and sort key

When you use the composite key in a DynamoDB table, which includes the partition key and the sort key, the approach for calculating the hash value remains the same. In addition to keeping partition key values physically close to one another, DynamoDB orders the data by the sort key value.

As described in the previous section, while storing an item in a table, DynamoDB uses the partition key value. The partition key value is supplied to the hash function, which determines the target partition for storing the item. The target partition may already have a number of items. In these scenarios, DynamoDB stores the item in the ascending order of sort key values in the table.

For reading an item from the table, you need to supply the partition key value as well as the sort key value. DynamoDB uses the partition key value to determine the source partition where the item can be found. DynamoDB even allows you to read multiple items from a table using a single query. However, reading multiple items in a single query requires that the items have the same partition key value. While querying the table, you can optionally apply a condition to the sort key so that the times within a specific range value are returned.

GSIs and LSIs

DynamoDB provides primary keys for quickly accessing items in a table by supplying primary key values in a query. The primary key is useful, and it can certainly speed up data retrieval from the table; however, in certain scenarios, applications can take advantage of **secondary indexes**. Secondary indexes can speed up item retrieval from a table based on any attribute aside from the primary key. For fulfilling these requirements, you need to create secondary indexes on the DynamoDB table. Once a secondary index is created on an attribute, you can use a `Query` or `Scan` request on specific indexes to retrieve items.

Secondary index refers to a data structure that is made up of a subset of attributes in a table. The main purpose of a secondary index is to provide an alternate key for query operations. You can use secondary indexes to read data using a query, in the same manner as you query a DynamoDB table. DynamoDB allows you to create multiple secondary indexes, which can help you to access the data using different query patterns.

A secondary index is linked with a table, which becomes the source of data for it. The source table from which an index takes data is also called the base table for the index. When defining an index, you need to define an alternate key, which can be a partition key and sort key. You can also choose which attributes you want to associate with the index. You can choose all attributes, primary keys, or you can choose a specific set of attributes from the table. DynamoDB copies all the attributes you choose into the index along with primary key attributes. Once the index is created, you can query or scan the index in the same way you query a table.

DynamoDB automatically maintains secondary indexes. When you change anything on the base table, the change is automatically reflected in the indexes. If you add, modify, or delete any item in the table, DynamoDB automatically updates this change in the index.

There are two types of secondary indexes:

- **GSI:** An index that can have a different partition key and sort key, compared to the base table. With GSIs, you can query data spanned across all the partitions in a base table. In short, when any query is executed against a GSI, its scope spans across all the partitions in a table. This is the reason it is called a **Global Secondary Index**.

- **LSI:** An index that has the same partition key as its base table; however, it has a different sort key. Every partition in an LSI is mapped with a base table partition, which carries the same partition key value. In short, LSI's scope is associated with its base table partition and this is the reason it is called a **Local Secondary Index**.

The difference between GSIs and LSIs

The difference between GSIs and LSIs is as follows:

Characteristic	GSI	LSI
Key schema	The primary key of a GSI can be a single attribute key, which is called a partition key, or it can be a composite key with two attributes: the partition key and the sort key.	The primary key of a LSI has to be a composite key, a combination of the partition key and sort key.
Key attributes	The partition key and sort key of the index can be any attributes from the base table.	The partition key of the index must be the same as the base table partition key. The sort key can be any attribute from the base table.
Size restrictions per partition key value	There is no restriction on the size of the index.	The total size of all items for a specific partition key value must be less than or equal to 10 GB in size.
Online index operations	A GSI can be created while creating a table. DynamoDB also allows you to add additional GSIs to an existing table or delete an existing index as and when required.	LSIs can be created while creating a table; however, DynamoDB does not allow you to add additional LSIs to an existing table. It does not even allow you to delete an existing LSI.
Queries and partitions	You can query the entire table stored across all the partitions in a GSI.	You can query only a single partition with a partition key value specified in the query.
Read consistency	A GSI supports only eventual consistency reads.	An LSI supports eventual consistency or strong consistency reads based on your requirements.
Provisioned throughput consumption	You need to configure a separate provisioned throughput for GSIs. You need to explicitly specify the read and write capacity units while creating a GSI. When you query a GSI, it uses read and write capacity units on top of base table consumption.	An LSI does not have its own provisioned throughput. It utilizes read and write capacity units from the base table when you query or scan a table or write data to base table.

Projected attributes	While executing a query or scan or a write request against a GSI, you can specify only the attributes, which are projected while creating the index. It cannot handle any request for attributes that are not defined as part of the index.	While executing a query, scan, or write request against an LSI, you can specify any attributes from the table, even if these attributes are not projected while creating the index. When you query any additional attributes that are not part of the index, DynamoDB automatically gets these attributes from the base table.
----------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

DynamoDB Query

DynamoDB Query is a mechanism to request items from a table. Using queries, you can request data from a table or any secondary index that has a composite primary key. While querying a table or index, you have to provide the name of the partition key attribute and a value for the same. The query returns all the items with that partition key value. You can also select a sort key attribute and filter the search result using any of the comparison operators.

For working with Query in GUI, you need to go to the DynamoDB dashboard: <https://console.aws.amazon.com/dynamodb>. From the dashboard, you can select the **Tables** | **Movies** | **Items** tab. In this example, we are working with the **Movies** table. You need to select the specific table name that you want to work with on your dashboard.

In the following screenshot, you can see the **Query** window for the **Movies** table, which has **year** as the partition key and **title** as the sort key:

The screenshot shows the AWS DynamoDB console interface for the 'Movies' table. The 'Items' tab is selected, indicated by a red arrow. The 'Query' dropdown is also highlighted with a red arrow. The query parameters are set to find items where the partition key 'year' is '2014' and the sort key 'title' begins with a specific value. The results table lists four movies from 2014: '300: Rise of an Empire', 'Captain America: The Winter Soldier', 'Divergent', and 'Dumb and Dumber To'. Each item includes its primary key (year), title, and a truncated JSON representation of its info attribute.

	year	title	info
	2014	300: Rise of an Empire	{ "actors": { "L": [{ "S": "Sullivan Stapleton" }, { "S": ...] } } }
	2014	Captain America: The Winter Soldier	{ "actors": { "L": [{ "S": "Chris Evans" }, { "S": "Fran...] } } }
	2014	Divergent	{ "actors": { "L": [{ "S": "Shailene Woodley" }, { "S": ...] } } }
	2014	Dumb and Dumber To	{ "actors": { "L": [{ "S": "Jennifer Lawrence" }, { "S": ...] } } }

DynamoDB Query

Do remember to select **Query** from the drop-down box, as shown in the preceding screenshot. By default, the **Items** window is loaded with the **Scan** option selected instead of **Query**.

For querying the table, you need to provide the partition key value and optionally provide the sort key value. In this example, the partition key is **year**, and **2014** is given as the value. You can click the **Start search** button after entering the required values. As you can see, the table displays all items with the partition key value as **2014**.

For sort keys and filters, you can use a number of key condition expressions. These expressions are described in the following table:

Operator	Example	Description
=	a = b	True if the attribute a is equal to the value b
<	a < b	True if a is less than b
<=	a <= b	True if a is less than or equal to b
>	a > b	True if a is greater than b
>=	a >= b	True if a is greater than or equal to b
Between	a BETWEEN b AND c	True if a is greater than or equal to b, and less than or equal to c
Begins with	begins_with (a, substr)	True if a begins with a specified substring

While working with Query, you can sort the output in ascending or descending order. You can opt to display all the attributes of the table or you can choose to project specific attributes from the table.

Query with AWS CLI

Here are some examples of using DynamoDB Query with AWS CLI:

```
aws dynamodb query \
    --table-name Items \
    --key-condition-expression "Item_ID = :id" \
    --expression-attribute-values '{":id":{"S":"i10001"}}'
```

The query retrieves all records for Item_ID i10001 from the Items table . As you can see in the preceding example, for initiating a query request to DynamoDB, you need to use the aws dynamodb query command on AWS CLI. aws dynamodb query requires some parameters. Let's explore the query with the following table:

Command/expression	Description
aws dynamodb query	AWS CLI command to initiate a query request to DynamoDB.
\	The backward slash, \, is used for indicating continuation of code in next line.
--table-name	Indicates the name of the DynamoDB table.

--key-condition-expression	Describes the key condition. In this example, it indicates the <code>Item_ID</code> attribute with <code>id</code> as the expression attribute. Expression attributes can be any string to describe the expression. It is different than table attributes. It is a custom name given to an expression. The value of the custom expression is declared in <code>--expression-attribute-values</code> , as described in the example.
--expression-attribute-values	Describes the value for the key-condition-expression attribute defined in <code>--key-condition-expression</code> . In this example, a custom expression attribute, <code>id</code> , is declared. Here, you need to declare the value type and the value for the expression attribute. In this example, <code>id</code> is of type <code>S</code> (string) with the value <code>i10001</code> . For the numeric expression attribute, you need to specify <code>N</code> instead of <code>S</code> .



If you need to deep dive into queries, you can take a look at them here: <http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Query.html>.

DynamoDB Scan

When you use Query on DynamoDB, it uses only primary key attribute values to perform a search on the table. You can further refine the result by using filters on attributes other than primary keys. Unlike a query, the `Scan` operation can perform a search on any attribute of the table. It also allows you to refine the search by applying filters to the scan result.

When you perform a `Scan` operation on a table, it reads all the items in the table or indexes and by default returns all the items and attributes. If you do not want to retrieve all the attributes, you can use the `ProjectionExpression` parameter to retrieve only specific attributes.

Irrespective of whether the items are found with the matching criteria or not, Scan always returns a result set. If items with the specified criteria are found, it returns the result set with the items; otherwise, it returns an empty result set. Every Scan operation can retrieve a maximum of 1 MB of data. You can apply a filter to the scan for further narrowing down the result based on the filter conditions.

The following AWS CLI example scans the Movies table and returns only the items with ReleaseYear as 2017:

```
aws dynamodb scan \
    --table-name Movies \
    --filter-expression "ReleaseYear = :name" \
    --expression-attribute-values '{":name":{"N":"2017"}}'
```

Reading an item from a DynamoDB table

You can use the GetItem operation for reading an item from a DynamoDB table. While performing a GetItem operation, you need to provide a table name and the primary key of the table item.

The following example shows how to read an item from the employee table using AWS CLI. In this example, employee_id is the primary key of the table:

```
aws dynamodb get-item \
    --table-name employee \
    --key '{"employee_id":{"S":"E10001"}}'
```

For reading an item from the table, it is necessary to specify the entire primary key. If a table has a composite key, you need to specify the partition key as well as the sort key in get-item. It performs an eventual consistent read by default; however, you can use a strongly consistent read by using the ConsistentRead parameter. Also, by default, get-item returns all the attributes of a table. If you want to return specific attributes of the table, you can use the project expression parameter. You can also set the ReturnConsumedCapacity parameter to TOTAL for returning the number of read capacities used by the get-item operation.

The following code example reads an item from a table and returns the number of read capacities used by the get-item operation:

```
aws dynamodb get-item \
    --table-name employee \
    --key '{"employee_id":{"S":"E10001"}}' \
    --consistent-read \
    --return-consumed-capacity
```

```
--projection-expression "FirstName, LastName, JoiningDate, Gender,  
DateOfBirth" \  
--return-consumed-capacity TOTAL
```

Writing an item to a DynamoDB table

DynamoDB provides the following operations for creating, updating, and deleting an item from a table:

- PutItem
- UpdateItem
- DeleteItem

For performing any of these operations, you need to specify the complete primary key. If the table has just a partition key, you can provide the partition key. If the table has a composite key, you need to provide both the partition key as well as the sort key. Providing just the partition key or the sort key alone does not work for these operations. You need to specify both the keys for the composite key table.

If you want the operation to return the write capacity consumed by the operation, you can set the `ReturnConsumedCapacity` parameter with one of the following values:

- **TOTAL:** Indicates the total number of write capacity units consumed by the operation.
- **INDEXES:** Indicates the total number of write capacity units consumed by the table along with the secondary indexes affected by the operation.
- **NONE:** Does not return any details for write capacity consumed by the operation. If you do not explicitly specify the `ReturnConsumedCapacity` parameter, by default, it returns `NONE`.

PutItem

`PutItem` is used for writing a new item in the table. If the table already has an item with the same key, it is replaced with the new item.

Take a look at this code block:

```
aws dynamodb put-item \  
  --table-name employee \  
  --item file://employee-item.json
```

Details of the `--item` argument are stored in the `employee-item.json` file, as shown here:

```
{  
    "FirstName": {"S": "Gini"},  
    "LastName": {"S": "Davidson"},  
    "JoiningDate": {"S": "20120817"},  
    "Gender": {"S": "Female"},  
    "DateOfBirth": {"S": "19850719"}  
}
```

UpdateItem

`UpdateItem` is used for updating an item in a table. If you use an existing primary key with `UpdateItem`, it updates the existing item. If the specified key does not exist, it creates a new item in the table.

While you can specify the attributes that you want to modify with the update expression, along with the update expression, you can use expression attribute values that act as placeholders for the real values.

Take a look at this example:

```
aws dynamodb update-item \  
  --table-name employee \  
  --key file://employee-key.json \  
  --update-expression "SET FirstName = :fname, LastName = :lname, \  
  JoiningDate = :jdate" \  
  --expression-attribute-values file://expression-attribute-  
  values.json \  
  --return-values ALL_NEW
```

The values for the arguments against the `--key` parameter are stored in the `employee-key.json` file, as shown here:

```
{  
    "employee_id": {"S": "E10001"},  
    "DateOfBirth": {"S": "19850719"}  
}
```

The values for arguments against `--expression-attribute-values` are stored in the `expression-attribute-values.json` file, as shown here:

```
{  
    ":fname": {"S": "Johnson"},  
    ":lname": {"S": "David"},  
    ":jdate": {"S": "19850720"}  
}
```

DeleteItem

The `DeleteItem` operation is used for deleting an item from a DynamoDB table. You need to supply a specific key value as a parameter with the `DeleteItem` operation, as shown here:

```
aws dynamodb delete-item \  
--table-name employee \  
--key file://key.json
```

It is important to note that if the table has a composite key, you need to specify both the partition key and the sort key.

Here are the contents of the `key.json` file:

```
{  
    "employee_id": {"S": "E10001"},  
    "DateOfBirth": {"S": "19850719"}  
}
```

Conditional writes

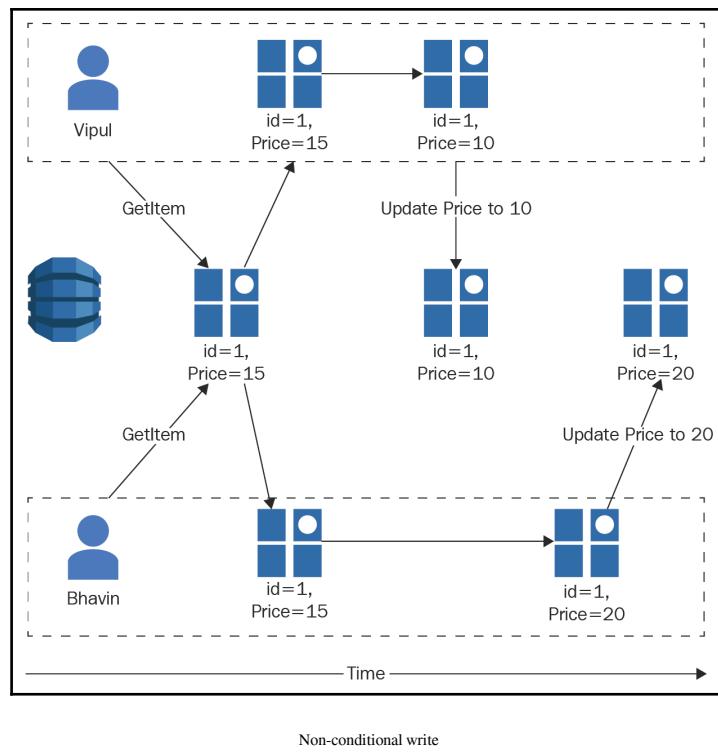
You can perform write operations in a DynamoDB table using `PutItem`, `UpdateItem`, and `DeleteItem`. By default, these operations are unconditional in nature; this means that when you perform write operations using these statements, they overwrite an existing item with the same primary key value specified in the operation.

If you do not want to overwrite an existing item, or write an item only based on a specific condition, you can use **conditional writes** with these operations. Conditional writes succeed only where they meet an expected condition; otherwise, they return an error. Conditional writes can be used in multiple scenarios.

Some example scenarios of when conditional writes can be used include the following:

- You want to write an item using the `PutItem` operation, only if the item with the specific key does not exist.
- You want to update an item using the `UpdateItem` operation, only if the item attribute contains a specific value.
- You want to update an item using the `UpdateItem` operation, only if it has not already been modified by another user.

Conditional writes can be handy in situations where multiple users try to update the same item. Let's look at this diagram to understand the scenario in which Vipul and Bhavin are trying to update the same item in a DynamoDB table:



Let's assume that Vipul tries to update the `Price` attribute to 10 in the `ProductMaster` table:

```
aws dynamodb update-item \
--table-name ProductMaster \
--key '{"Id":{"N":"1"}}' \
--update-expression "SET Price = :newprice" \
--expression-attribute-values file://expression-attribute-
values.json
```

As we explained earlier, the arguments for `--expression-attribute-values` should be stored in a separate JSON file. In this case, arguments are stored in the file named `expression-attribute-values.json` with the following content:

```
{
  ":newprice": {"N": "10"}
}
```

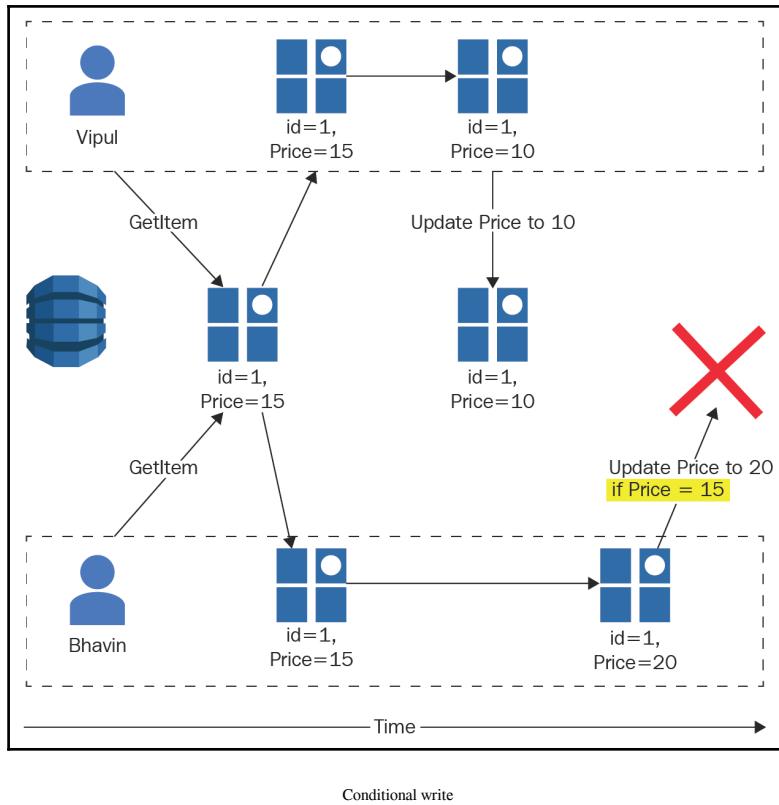
Now, let's consider that Bhavin updates the same item using the `UpdateItem` request and changes the price to 20. For Bhavin, the `--expression-attribute-values` parameter file can contain the following values:

```
{
  ":newprice": {"N": "20"}
}
```

The `UpdateItem` operation initiated by Bhavin succeeds, but it overwrites the change made by Vipul.

For performing a conditional write, you need to specify the condition expression along with `PutItem`, `DeleteItem`, or `UpdateItem`. A condition expression contains a string with attribute names, conditional operators, and built-in functions. The operation executes only if the entire expression evaluates to true; if it does not, the operation fails.

Let's look at the conditional write in the following diagram, which shows how the conditional write prevents Bhavin from overwriting Vipul's changes on the same item. You can compare the previous diagram to the following one, noticing how the condition is highlighted:



Let's look at how you can achieve this using AWS CLI. While updating `Price`, Vipul gives a condition to update `Price` to 10 only if `Price` is 15, as shown here:

```
aws dynamodb update-item \
    --table-name ProductMaster \
    --key '{"Id":{"N":"1"}}' \
    --update-expression "SET Price = :newprice" \
    --condition-expression "Price = :currprice" \
    --expression-attribute-values file://expression-attribute-
values.json
```

The arguments for `--expression-attribute-values` should be stored in a separate JSON file. In this case, the arguments are stored in the file named `expression-attribute-values.json` with the following content:

```
{  
    ":newprice": {"N": "10"},  
    ":currprice": {"N": "15"}  
}
```

Vipul's update succeeds as the condition evaluates to be true. Similarly, Bhavin tries to update the `Price` to 20 with a conditional expression that checks for the current price as 15 before updating it.

For Bhavin, the `--expression-attribute-values` parameter file can contain the following values:

```
{  
    ":newprice": {"N": "20"},  
    ":currprice": {"N": "15"}  
}
```

Since Vipul has already changed `Price` to 10, the condition expression in Bhavin's request evaluates to false and his update fails.

User authentication and access control

For accessing DynamoDB, you need **credentials**. The credentials should have the permission to access the DynamoDB table. This section provides details on how you can use **Identity and Access Management (IAM)** to secure DynamoDB resources.

There are a number of ways in which you can access DynamoDB resources:

- AWS root user account
- IAM user
- IAM role:
 - Identity federation
 - Cross-account access
 - AWS service access
 - Application running on EC2

If you have valid credentials that can authenticate against DynamoDB, you can initiate the request to access, but unless you have permissions associated with your credentials, you cannot perform any operation against DynamoDB. For example, for creating a new DynamoDB table, you need to have the table creation permission.

Before understanding these permissions, let's understand the resources in DynamoDB. Tables are the primary resources in DynamoDB. Apart from tables, there are indexes and streams, which are part of DynamoDB. Indexes and streams are associated with a table and they are sub-resources of a table. DynamoDB maintains unique **Amazon Resource Names (ARNs)** with each of the resources and sub-resources, as shown in the following table:

Resource type	ARN format
Table	arn:aws:dynamodb:region:account-id:table/table-name
Index	arn:aws:dynamodb:region:account-id:table/table-name/index/index-name
Stream	arn:aws:dynamodb:region:account-id:table/table-name/stream/stream-label

Managing policies

Access to any resource is determined by a permission policy. In chapter 3, *Identity and Access Management (IAM)*, managing policy is discussed in detail. This chapter emphasizes IAM with respect to DynamoDB. There are two types of policy: **identity-based policies** and **resource-based policies**. Identity-based policies are attached to an IAM identity and resource-based policies are attached to a resource. This section discusses identity-based policies only, as DynamoDB does not support resource-based policies.

There are three identities in IAM:

- User
- Group
- Role

You can attach a policy to a user or a group and grant them access to DynamoDB resources, such as tables, indexes, and streams. You can attach a policy to a role for granting cross-account permissions. The following example policy allows dynamoDB:ListTables permission for all resources:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {
```

```
        "Sid": "ListTables",
        "Effect": "Allow",
        "Action": [
            "dynamodb>ListTables"
        ],
        "Resource": "*"
    }
]
}
```

Here's one more example of a permission policy. In this example policy, access is granted on three actions, namely `DescribeTable`, `Query`, and `Scan`. As you can see in the previous policy, access is granted to all resources with `*` and, unlike the previous policy, in this policy, access is granted to a specific table using an ARN for the table:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DescribeQueryScanEmployeeTable",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:account-
id:table/employee"
        }
    ]
}
```

DynamoDB API permissions

While setting up a permission policy, you can refer to the following URL, wherein a detailed table is given that lists DynamoDB API operations, associated actions, and the ARN format for the resource.



The table can be found at <http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/api-permissions-reference.html>, and it gives you the complete details of DynamoDB API permissions.

DynamoDB best practices

DynamoDB best practices are as follows:

- Create a primary key that spans multiple partitions. Choose the primary key that has more distinct values. If the number of distinct values is less in a primary key attribute, items may be distributed in a limited number of partitions instead of all available partitions.
- In DynamoDB, each item can have a maximum size of 400 KB; however, in a table, there is no limit on the number of items. To efficiently store large items in a table, use one of the mechanisms, such as a one-to-many table, multiple tables to support varied access patterns, compress large attribute values, store large attribute values in Amazon S3, or break up large attributes across multiple items.
- By default, `Scan` reads the entire table with all items and consumes more throughput. Use `Query` instead of `Scan`, as it is more economical.
- Create LSIs for frequently queried attributes on the table apart from primary key attributes. It improves the query performance.
- If it is required to use the `Scan` operation, design an application to use the `Scan` operation in a way that minimizes the impact on read requests on the table.
- One or more secondary indexes can be created to perform an efficient search and query on the attributes other than the primary and sort keys.
- Rather than `Scan`, use parallel scan to retrieve a large dataset from the table, as it uses multiple work threads in the background at low priority without affecting the production traffic. These background processes are called **sweepers**.
- It is suggested to plan LSIs very carefully, as they can only be defined at the time of creating a table, and later they cannot be deleted throughout the table's life cycle. On top of that, they share primary read and write throughput from the base table.
- A GSI allows for the creation of a secondary index with a different primary key and sort key from the base table. This secondary index can be created and deleted separately at any given time on the base tables. These indexes are maintained automatically with the base table but have their own read/write throughput, so create them wisely.

Summary

- DynamoDB is an easy-to-use and fully managed NoSQL database service.
- A NoSQL database provides a way to store and retrieve data that is in a non-tabular format.
- There are basically four types of NoSQL databases, that is, key-value pair databases, document databases, graph databases, and wide column stores.
- A key-value pair database uses a very simple data model that stores data in a pair of unique keys and the associated value.
- A document database stores data elements in a structure that represents a document-like format, such as JSON, XML, or YAML.
- A graph database is a NoSQL database type that uses graph structures and stores related data in nodes.
- The wide column database is a type of NoSQL database that stores data using a column-oriented model.
- DynamoDB supports key-value and document data models.
- DynamoDB allows you to scale up or down a table's read/write capacity without affecting the uptime and performance of the table.
- DynamoDB can reduce storage usage by automatically deleting the expired items from a table.
- There are basically three core components of a DynamoDB table: tables, items, and attributes.
- An item in a table consists of multiple attributes. An attribute is the basic data element of an item. It is similar to a field or a column in an RDBMS.
- Unlike the case in an RDBMS, attributes in a DynamoDB table item can have sub-attributes.
- It is mandatory to define a primary key in the DynamoDB table.
- There are two types of primary keys, that is, the partition key and sort key.
- A partition key has one attribute.
- A partition key and sort key are two attributes. Together they are called as composite primary key.
- There are two types of secondary indexes, that is, GSI and LSI.
- A GSI consists of a partition key and a sort key, which have to be different than the primary keys defined on the table.
- An LSI uses the same partition key as the table but uses a different sort key.

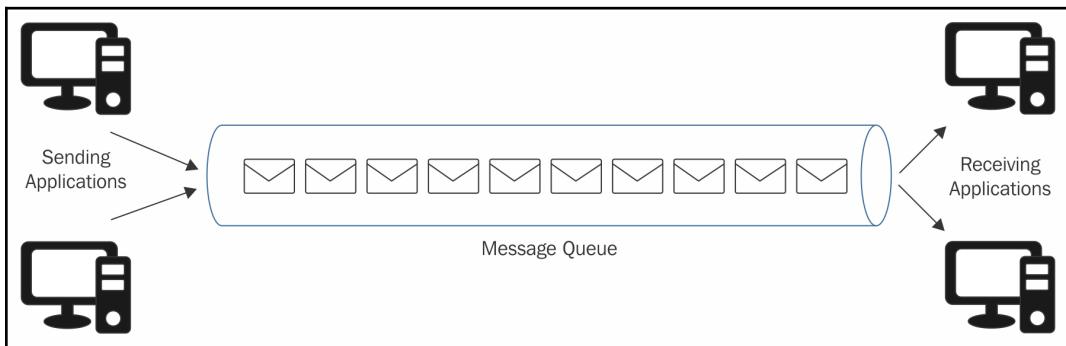
- DynamoDB Streams is an optional feature; when enabled, it captures data modification events whenever a DynamoDB table is changed.
- Whenever you write data to a DynamoDB table, AWS replicates this data across multiple AZs to provide high availability.
- After writing data to a DynamoDB table, you get an HTTP 200 response.
- HTTP 200 (OK) indicates that the data has safely updated to all the replicated copies stored in different AZs.
- AWS provides two types of read consistency models, that is, eventually consistent read and strongly consistent read.
- DynamoDB supports a number of data types for attributes in a table. These data types can be categorized into three parts, that is, scalar types, document types, and set types.
- DynamoDB offers two read/write capacity modes, that is, on-demand mode and provisioned mode.
- DynamoDB provides the Auto Scaling feature for automatically scaling the read and write capacity of a table.
- Amazon DynamoDB supports encryption at rest using KMS.
- One read capacity unit = one strongly consistent read.
- One read capacity unit = two eventually consistent reads.
- One read operation can process an item of up to 4 KB in size.
- One write capacity unit = one write operation of up to 1 KB in size.
- If an item is greater than 1 KB, it requires additional write capacity units.
- If an item is less than 1 KB in size, it still requires one write capacity unit.
- Table partitioning is a mechanism to segregate a large table into smaller, more manageable parts without creating a separate table for each part.
- DynamoDB uses an internal hash function for writing an item to the table.
- Based on the partition key value, the hash function determines the target partition for storing the item.
- The main purpose of a secondary index is to provide an alternate key for query operations.
- DynamoDB allows you to create multiple secondary indexes, which can help you to access the data using different query patterns.
- A GSI is an index that can have a different partition key and sort key as compared to the base table.

- An LSI is an index that has the same partition key as its base table; however, it has a different sort key.
- A GSI's scope spans across all the partitions in a table.
- An LSI's scope is associated with its base table partition.
- Using DynamoDB queries, you can request data from a table or any secondary index that has a composite primary key.
- A DynamoDB Scan operation can perform a search on any attribute of the table.
- You can perform write operations in a DynamoDB table using PutItem, UpdateItem, and DeleteItem.
- DynamoDB can perform write operations based on a condition. This is called conditional writes.

12

Amazon Simple Queue Service (SQS)

Before we look at **Simple Queue Service (SQS)**, let's look at what a **message queue** is. A message queue is a queue of messages exchanged between applications. Messages are data objects that are inserted in the queue by sender applications and received by receiving applications. Receiving applications get the data objects from the queue and process the data received from the queue based on the application requirements. The following diagram describes the message queue in a simple way:



Amazon SQS is a highly reliable, scalable, and distributed message queuing service provided by Amazon. It's a hosted solution provided by Amazon, so you do not need to manage the service infrastructure. Amazon SQS stores the messages in transit as they travel between various applications and micro services. Amazon provides a host of web service APIs, which can be accessed using any programming language supported by AWS SDK.

The purpose of this chapter is to introduce readers to the basic concepts of SQS with respect to the scope of the AWS Certified Developer – Associate exam. From a development perspective, SQS is a wide topic and a full book can be written on SQS. Considering the scope of the exam, this chapter does not intend to teach the reader how to code SQS applications, but focuses more on fundamental aspects of SQS.

The following topics will be covered in this chapter:

- Why use SQS?
- How do queues work?
- Main features of SQS
- Queue attributes
- Operations in a queue
- SQS limits
- Queue monitoring and logging
- SQS security

Why use SQS?

There are many reasons to use SQS. Let's start by looking at a simple use case followed by a brief review of a few more use cases.

Consider a scenario where you have a collaborative news site or application that accepts images from users, optimizes these images to display in multiple devices, and stores them for future retrieval. If the application users are spread across the globe, the application may get a huge number of visitors that upload images. If these images are directly offloaded to the processing server, the server may not be able to handle the traffic.

If you use a scaled-up environment to process the images, you end up incurring more costs. Also, a scaled-up environment may miss processing some of the images if the process gets interrupted or crashes.

In these scenarios, a message queue comes in handy. The application can send all the image processing requests to a message queue. At the other end, the image processing component of the application can read the message queue and process the requests one by one. This way, the application is not flooded with requests and all the images are processed irrespective of the amount of traffic.

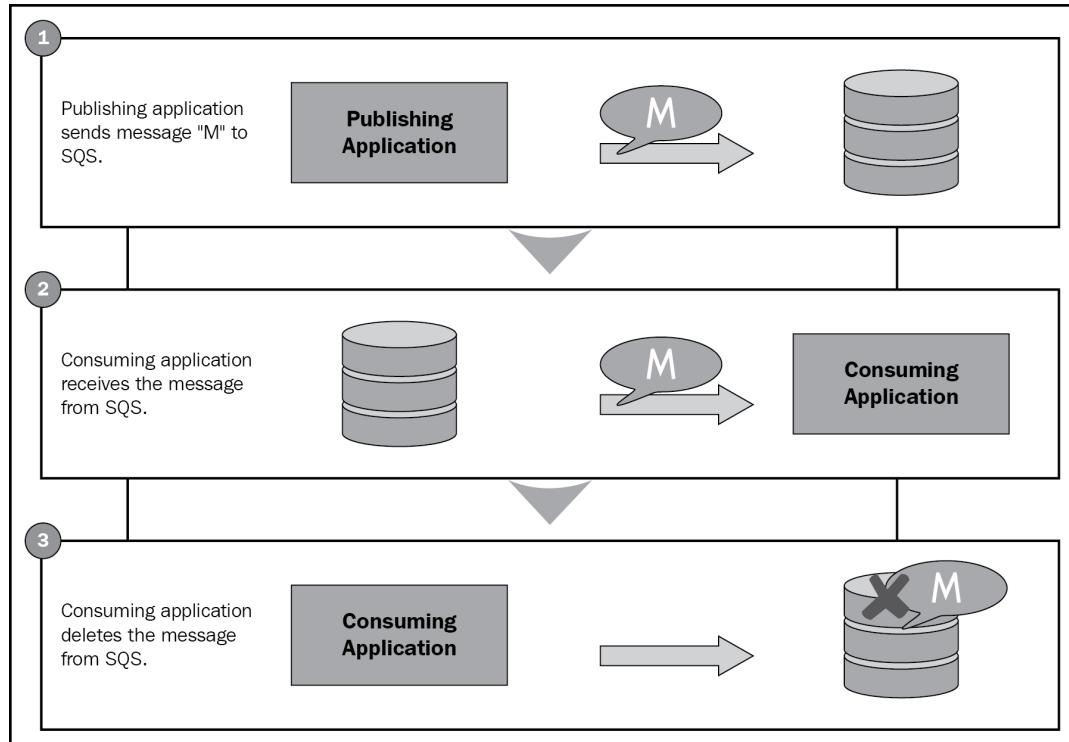
Here are a few more use cases for SQS:

- **Decoupling application processes:** When you start a project, you may not be able to predict what the future needs of the project will be. By segregating the data generation process, the data consumption process, and implementing a message queue in between the processes, you can create a data-based interface. All the processes involved in the application can implement this data-based interface. In this scenario, any process involved in the overall application workflow can be changed without disturbing other processes.
- **Application scalability:** Since the application processes are decoupled with a message queue, you can easily scale up the data generation and data consuming processes to control the rate at which the data is processed. Without changing the underlying code and by simply adding more processing resources, you can scale up the whole process.
- **Guaranteed message delivery:** The use of a message queue ensures that a message is delivered at least once, and is eventually processed by the processing application as long as the process continues reading the queue. Depending on the configuration of the queue, you can also ensure that a message is processed only once. This guarantee is possible with SQS; when a process retrieves the message from the queue, it temporarily removes this message from the queue. When the client informs the queue that it has finished processing the message, SQS deletes this message from the queue. If the client does not respond back to the queue in a specific amount of time, SQS places the message back in the queue. This way, if the message is not processed by a client, it will be available for another client to process.
- **Message order guarantee:** In many scenarios, the order in which the data is processed is very important. SQS provides a mechanism to process the data in a predefined order.
- **Asynchronous data processing:** There are certain requirements where you do not need to process the data immediately. SQS allows you to process the messages asynchronously. That means you can add a message on the queue, but do not need to process it immediately. You can add as many messages as required and process them later.

- **Building resilience in the system:** If some processes in your application environment go down, your entire environment does not go down with it. Since SQS decouples the processes, if a client processing a message goes down, the message is added back to the queue after a specific amount of time. This message is either processed by another client or the same client after it recovers. This way, SQS builds resilience in the system.
- **Bringing elasticity to applications with SQS:** When your application hits an unusually high amount of traffic, it should be able to sustain the load with additional resources. When the traffic subsides, it is wasteful to keep resources on standby for future needs. In this scenario, you can use SQS to build elasticity into your application. Since SQS helps you decouple the processes, you can host data publisher processes, as well as data consumer processes, in separate resources. Data publisher and data consumer resources can be scaled up when there is a high amount of traffic, and the resources can be scaled down when traffic is low.
- **Analyzing data flow and performance of the processes:** When you build a distributed system, it is critical to understand how each of the components performs, and, if there is any delay, the reason for that delay. SQS can easily help identify under-performing resources with the insight on the rate at which each of the resources perform.
- **Building redundancy with SQS:** Sometimes, processes do fail while processing data. In these scenarios, if the data is not persistent, it is lost forever. SQS takes care of this data risk by persisting the data until it's completely processed. SQS uses something called the **put-get-delete** paradigm. It requires a consumer to explicitly state that its data has finished processing the message it pulled from the queue. The message data is kept safe with the queue and gets deleted from the queue only after confirmation that it has been processed.

How do queues work?

As shown in the following diagram, SQS works on the **put-get-delete** paradigm:



Working of SQS

The put-get-delete paradigm works in the following way:

1. The **Publishing Application** pushes a message, **M**, into the queue.
2. The **Consuming Application** pulls the message, **M**, from the queue and processes it.
3. The **Consuming Application** confirms to the SQS queue that processing on the message is completed and deletes this message from the SQS queue.

Main features of SQS

SQS provides a scalable and reliable messaging platform. It enables you to build operational efficiency in your application without any operational overhead. Here are some of the benefits of using SQS:

- **Redundant infrastructure:** Amazon SQS provides redundant infrastructure. With its redundant infrastructure, it ensures that a message is delivered at least once in a standard queue and it ensures that a message is delivered exactly once in a **First In First Out (FIFO)** queue. It provides a concurrent access mechanism and a highly available environment for queue producer and consumer applications.
- **Multiple producers and consumers:** Multiple components of a distributed application can concurrently send and receive messages at the same time. When a client picks up a message from the queue, SQS locks up that message until the client confirms that it has completed processing the message. If the queue does not receive a response from the client for a specific amount of time, the queue unlocks the message and makes it available for other clients to pick up from the queue.
- **Queue-wise configurable settings:** SQS provides options to configure each queue independently. You do not need to have the same configuration for all the queues. For example, queue A may take longer than queue B to process a request. This requirement means you need to configure your queues differently, and SQS allows you to configure queues independent of each other.
- **Variable message size:** SQS supports a maximum message size of 256 KB. If you need a larger size than 256 KB, you can store it in S3 or DynamoDB. SQS holds the pointer to the S3 object in the queue.
- **Queue access control:** SQS allows you to control producers and consumers that can send and receive messages to or from the queue.
- **Delay queue:** SQS enables you to set a delay time in a queue. Delay time ensures that a message inserted in the queue is postponed for the time configured as **delay time** in the queue. This delay time can be set while creating a queue. You can also change the delay time later on; however, any change in the delay time is effective only with the new messages added to the queue.

- **Payment Card Industry (PCI) compliance:** PCI standard mandates that any service that handles payment data must adhere to PCI standards in order to be PCI compliant. SQS supports the handling of credit card payments with the storage and transmission of credit card data. Considering this need, Amazon has built a PCI-compliant service.
- **Health Insurance Portability and Accountability Act (HIPAA) compliance:** SQS supports HIPAA compliance. If you have a **Business Associate Agreement (BAA)** with AWS, it enables you to use SQS for building an HIPAA-compliant application that can store **Protected Health Information (PHI)**.

Types of queues

SQS supports these types of queues:

- **Standard queues**
- **FIFO queues**

Standard queues and FIFO queues

A standard queue generally sends the data in the same order as it receives it; however, on certain occasions, the order may change. Unlike a standard queue, the order in which the data is sent is fixed in FIFO queues.

Let's look at the differences between these two queue types with the help of the following table:

Description	Standard queue	FIFO queue
Availability	It's available in all AWS regions.	It's available only in the US East (North Virginia), US East (Ohio), US West (Oregon), and EU (Ireland) regions.
Throughput	It supports an almost unlimited number of Transactions Per Second (TPS) with each API action.	It supports limited throughput. It can support up to 300 messages per second without any batching. With a batch size of 10 messages, it can support up to 3,000 messages per second.

Order	It generally sends the data in the same order as it receives it; however, on certain occasions, the order may change.	It ensures that the data is sent in FIFO order.
Delivery guarantee	A standard queue guarantees that a message will be delivered at least once; however, a message may occasionally be delivered more than once.	A FIFO queue guarantees that a message will be delivered exactly once. The message remains in the queue until the consumer confirms that the message is processed.
Usage	It is used when the throughput is more important than the order in which the data is processed.	It is used when the order in which the data is processed is more important than the throughput.

Dead Letter Queue (DLQ)

A **Dead Letter Queue (DLQ)** is used by other queues for storing failed messages that are not successfully consumed by consumer processes. You can use a DLQ to isolate unconsumed or failed messages and subsequently troubleshoot these messages to determine the reason for their failures.

Amazon SQS uses a redrive policy to indicate the source queue, and the scenario in which SQS transfers messages from the source queue to the DLQ. If a queue fails to process a message a predefined number of times, that message is moved to the DLQ.

While creating a queue, you can enable a redrive policy and set the DLQ name as well as maximum receive count, after which if the message is still unprocessed, it can be moved to the DLQ.

Queue attributes

SQS uses certain queue attributes that define the behavior of a queue. While creating a queue, you can either create a queue with default attributes or customize these attributes as per your needs.

The following table describes these queue attributes with their acceptable ranges:

Queue attribute	Description	Minimum acceptable range	Maximum acceptable range
Default visibility timeout	The length of time that a message is received from a queue will be invisible to other receiving components.	0 seconds	12 hours
Message retention period	The amount of time that SQS retains a message if it does not get deleted.	1 minute	14 days
Maximum message size	The maximum message size accepted by SQS.	1 KB	256 KB
Delivery delay	The amount of time to delay the first delivery of all messages added to the queue.	0 seconds	15 minutes
Receive message wait time	The maximum amount of time that a long-polling receive call waits for a message to become available before returning an empty response.	0 seconds	20 seconds
Content-based deduplication	The FIFO queue uses more than one strategy to prevent duplicate messages in a queue. If you select this checkbox, SQS generates an SHA-256 hash of the body of the message to generate the content-based message deduplication ID. When an application sends a message using <code>SendMessage</code> for the FIFO queue with a message deduplication ID, it is used for ensuring deduplication of the message.	N/A	N/A

Operations in a queue

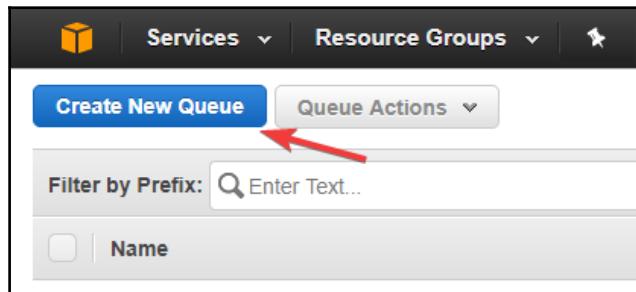
There are various operations that we can perform in the queue. The various operations that we will be performing on a queue are as follows:

- Creating a queue
- Sending a message in a queue
- Viewing/deleting a message from a queue
- Purging a queue
- Deleting a queue
- Subscribing a queue to a topic
- Adding user permissions to a queue

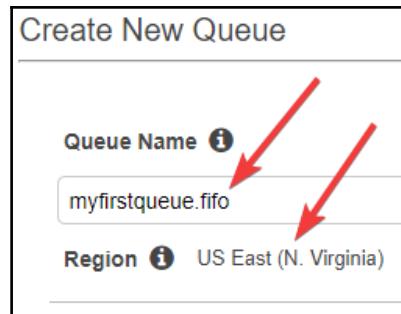
Creating a queue

Since you have developed a basic understanding of what SQS is, let's go through the following steps to create and configure our first SQS queue:

1. Log into the AWS Management Console and navigate to <https://console.aws.amazon.com/sqs/>.
2. Click the **Create New Queue** button, as shown:

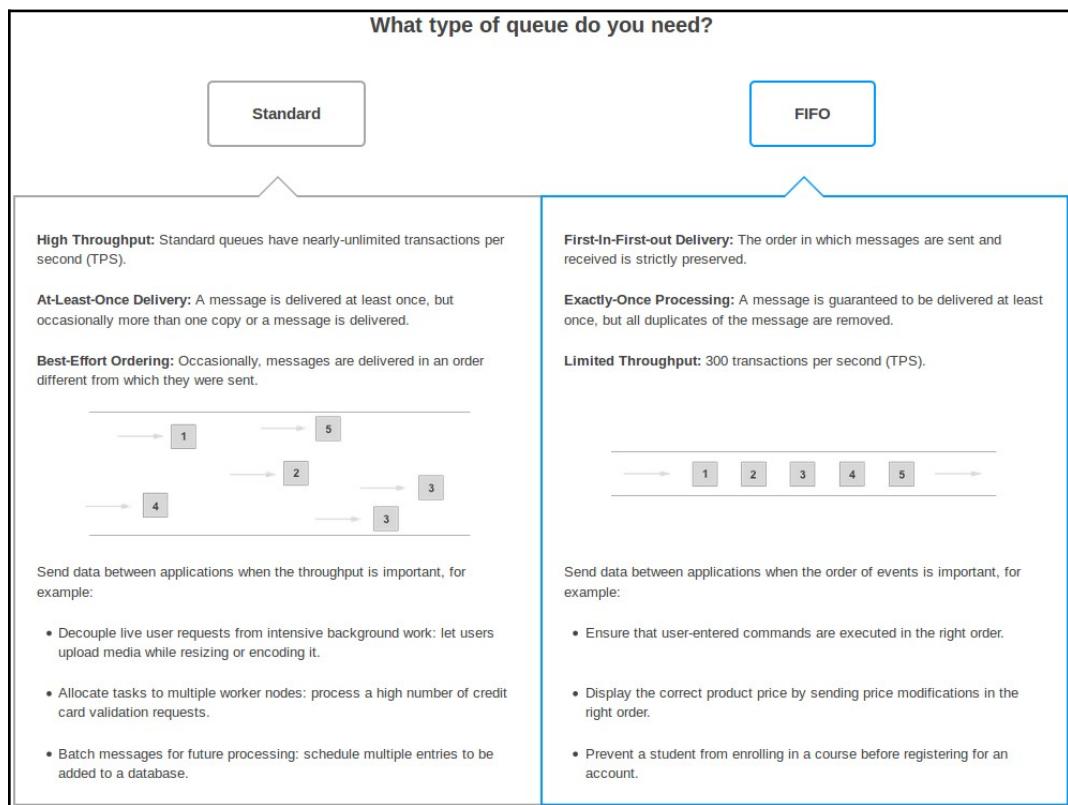


3. On the subsequent page, type the name of the queue as shown, ensuring that you are in the intended region. If required, you can change the region from the top-right corner of the screen. Also, note that **Queue Name** is case sensitive and can have a maximum of up to 80 characters. When you're creating a FIFO queue, **Queue Name** must end with a `.fifo` suffix:



Providing a queue name

4. By default, the SQS wizard has a **Standard** queue select. Depending on your requirements, you can choose **Standard** or **FIFO**, as shown:



Queue types

5. Click on the **Quick-CREATE Queue** button for creating a queue with the default parameters. Alternatively, you can click the **Configure Queue** button for configuring the queue parameters. These queue attributes are described in the previous table. You can enter appropriate values in the queue attributes, as shown:

You can change these default parameters.

Queue Attributes

Default Visibility Timeout	30	seconds	Value must be between 0 seconds and 12 hours.
Message Retention Period	4	days	Value must be between 1 minute and 14 days.
Maximum Message Size	256	KB	Value must be between 1 and 256 KB.
Delivery Delay	0	seconds	Value must be between 0 seconds and 15 minutes.
Receive Message Wait Time	0	seconds	Value must be between 0 and 20 seconds.
Content-Based Deduplication	<input type="checkbox"/>		

Queue attributes

6. On the same screen, you can choose to enable the redrive policy, as shown. This is an optional step and is required only if you want to divert unprocessed messages to a DLQ. If you enable the redrive policy, you need to specify the name of an existing queue that can act as a DLQ. You also need to specify the maximum receive count. If a message is received back in the specified amount of time, it is moved to the DLQ:

Dead Letter Queue Settings

Use Redrive Policy	<input checked="" type="checkbox"/>	mydead-letter-queue	Value must be an existing queue name.
Dead Letter Queue	mydead-letter-queue	Value must be an existing queue name.	
Maximum Receives	5	Value must be between 1 and 1000.	

DLQ settings

7. In this step, you can choose to enable **Server Side Encryption (SSE)**, as shown. If you choose to enable SSE, you can either use the default **Customer Master Key (CMK)** from the AWS Key Management Service (KMS), or you can specify any other existing CMK you have in KMS. If you select any key other than the default CMK, you need to manually specify the ARN key. You also need to specify a **Data Key Reuse Period** that can range between 1 minute and 24 hours. For encrypting or decrypting data, SQS needs a data key, which is provided by KMS. Once a key is obtained from KMS, it can be used by SQS for the time specified in **Data Key Reuse Period** before going back to KMS for a new data key:

Server-Side Encryption (SSE) Settings

Use SSE

AWS KMS Customer Master Key (CMK) (Default) aws/sqs

Description Default master key that protects my SQS messages when no other key is defined

Account 835808778612

Key ARN arn:aws:kms:us-east-1:XXXXXXXXXXXXX:key/XXXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX

Data Key Reuse Period 5 minutes This value must be between 1 minute and 24 hours.

SSE settings

8. After providing all the required input on the screen, you can click the **Create Queue** button. Once the queue is created, you can see the queue in the SQS dashboard, as shown:

Filter by Prefix: <input type="text"/> Enter Text...					
Name	Queue Type	Content-Based Deduplication	Messages Available	Messages in Flight	Created
myfirstqueue fifo	FIFO	Disabled	0	0	2017-09-16 12:22:11 GMT+05:30

SQS dashboard with newly created queue

You can select the queue and see more descriptions on the dashboard. You can also perform other actions on the queue, such as configuring a queue, sending a message, and deleting a queue.

Sending a message in a queue

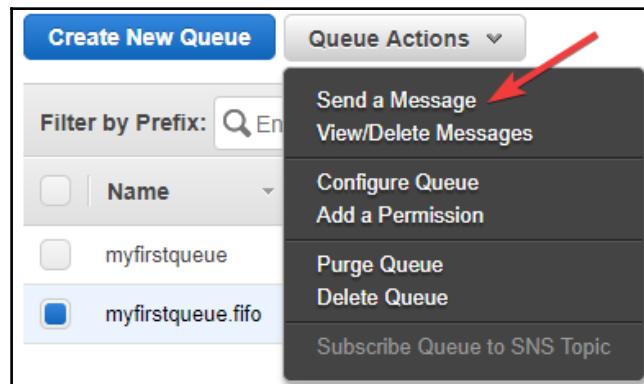
To send a message in a queue, you can perform the following steps:

1. Go to the SQS dashboard and select the queue you want to send the message to, as shown:



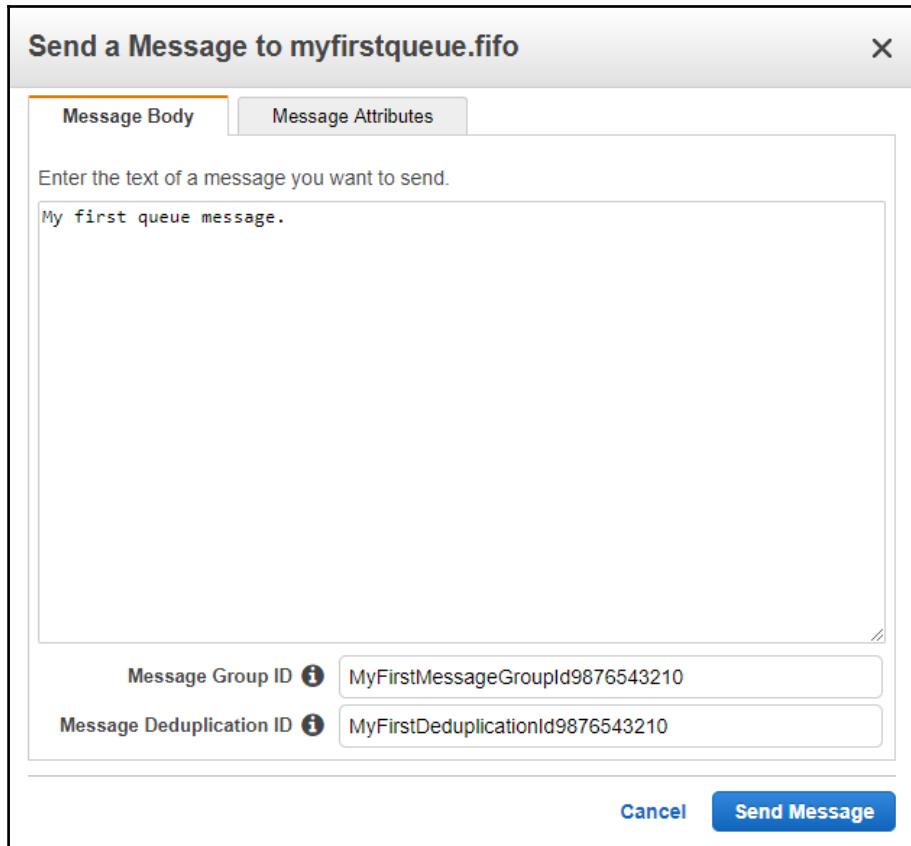
Queue list

2. Click the **Queue Actions** button and select **Send a Message**, as shown:



Queue Actions – Send a Message

3. In the subsequent screen, enter the message that you want to send to the queue. Enter the **Message Group ID** and also specify the **Message Deduplication ID**, as shown. **Message Group ID** is mandatory; it is used for grouping the message. When you specify **Message Group ID**, messages sent to a specific group ID in a FIFO queue are guaranteed to be delivered in the FIFO order. If you have enabled the **Content-Based Deduplication** checkbox while creating the queue, **Message Deduplication ID** is optional:



Send a Message options

Optionally, you can also specify **Message Attributes**, as shown:

The screenshot shows the 'Send a Message to myfirstqueue.fifo' dialog box. At the top, there are two tabs: 'Message Body' (selected) and 'Message Attributes'. A red arrow points to the 'Message Attributes' tab. Below the tabs, there are three fields: 'Name' (address), 'Type' (String), and 'Value' (Prestige Notting Hill...). A note below says 'Enter a string value.' In the bottom left, there's a 'Add Attribute' button with a red arrow pointing to it, and a link 'What is Message Attribute?'. On the right, there's a table for attributes:

Name	Type	Values	X
city	String	Bangalore	X
empname	String	Ankur Sharma	X
pincode	String	560076	X

At the bottom right are 'Cancel' and 'Send Message' buttons, with a red arrow pointing to the 'Send Message' button.

Message Attributes

- Finally, click the **Send Message** button. It displays a confirmation message, as shown:

Your message has been sent and is ready to be received.
Note: It may take up to 60 seconds for the *Messages Available* column to update.

Sent Message Attributes:

Message Identifier:	83fd7112-6ce8-4723-8a9c-3c8f222f5397
MD5 of Body:	634404ebcf889e4e4a8f26097127553
MD5 of Message Attributes:	06c8bc6f7034c7ecd2e1bbcb8d7b52eb
Sequence Number:	18832165418875119616

Message sent confirmation

You can either send another message or close the window after reading the confirmation.

Viewing/deleting a message from a queue

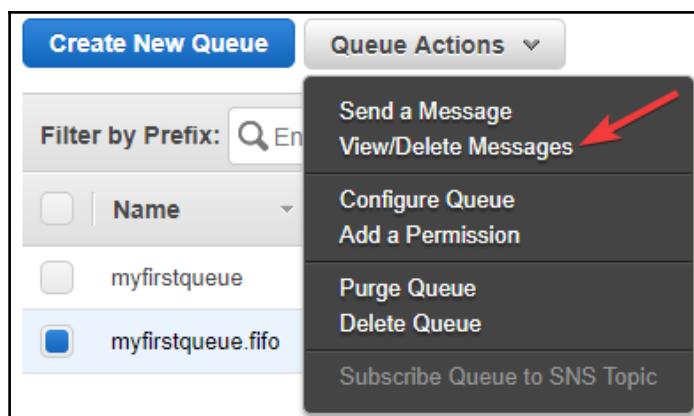
After the message is sent to the queue, you can retrieve it from the queue. While retrieving a message from the queue, you cannot specify which message you want to retrieve from the queue; however, you can specify how many messages you want to retrieve. Here are the steps for viewing a message from the queue:

1. Select a queue from the queue list, as shown:



Queue list

2. Click **Queue Actions** and then **View/Delete Messages**, as follows:



Queue Actions – View/Delete Messages

3. From the subsequent screen, you can click **Start Polling for Messages**, as shown:



Polling for messages

4. In the subsequent screen, you can see up to 10 messages available in the queue, as specified in the previous step. The screen should resemble this:

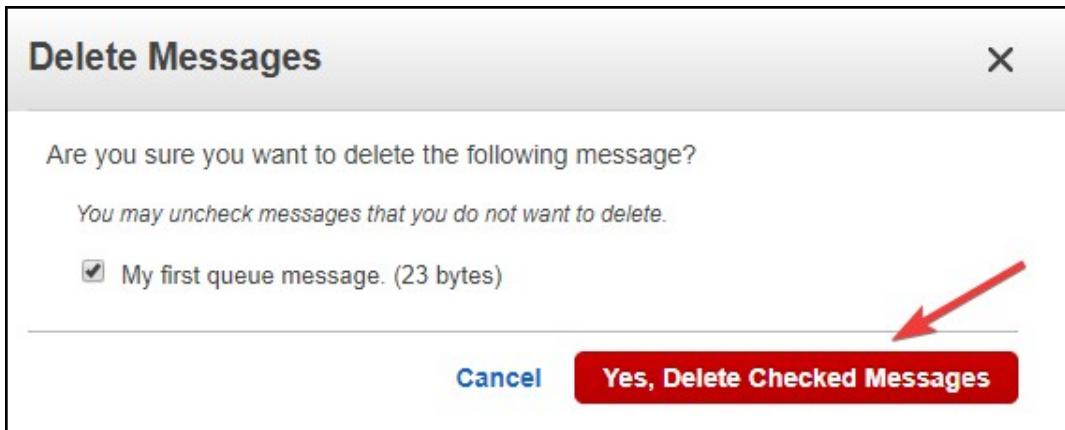
The screenshot shows the same modal window after polling. It displays a table of messages. The first message, "My first queue message.", has its "Delete" checkbox checked (highlighted with a red arrow). At the bottom right of the table area is a red button labeled "Delete 1 Message" (highlighted with a red arrow).

Delete	Body	Message Group ID	Message Deduplication ID	Sequence Number	Size	Sent
<input checked="" type="checkbox"/>	My first queue message.	MyFirstMessageGroupId98...	MyFirstDeduplicationId9876543...	18832165418875119616	23 bytes	2017-09-16 14:25:29 GMT+

100%
Stopped after polling the queue at 0.5 receives/second for 30.7 seconds. Messages shown above are now available to other consumers.
Close Delete 1 Message

Messages in a queue

5. You can select one or more messages from the list that you want to delete, and click **Delete 1 Message**. It displays the **Delete Messages** dialog box, as shown. You can click **Yes, Delete Checked Messages**. This action deletes the selected message:



Delete message confirmation

Purging a queue

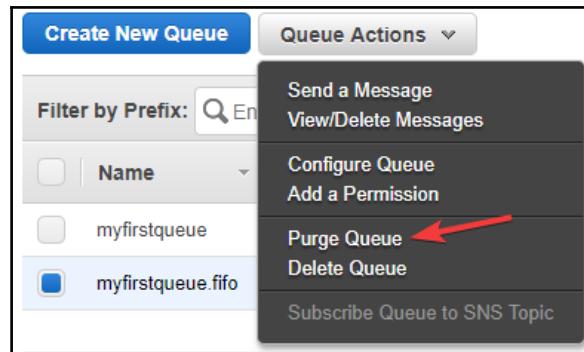
Purging a queue means deleting all the messages from a queue. To purge a queue, you can follow these steps:

1. Select a queue from the queue list, as shown:



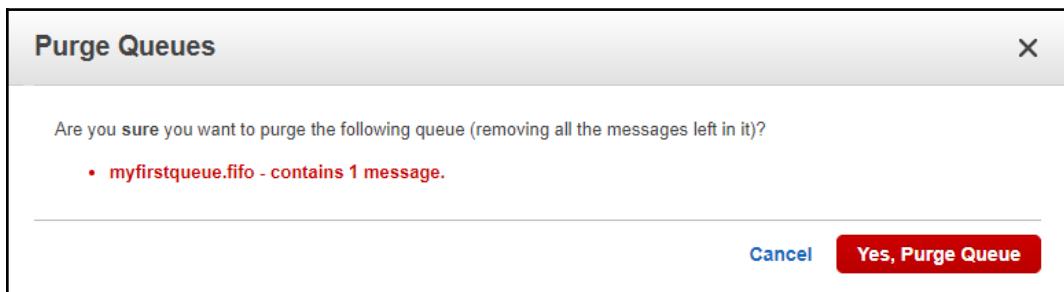
Queue list

2. Click **Queue Actions**, and then **Purge Queue**, as follows:



Queue Actions – Purge Queue

3. The subsequent screen displays a confirmation message with a number of messages that it would purge from the queue, as shown. Clicking the **Yes, Purge Queue** button purges the queue:

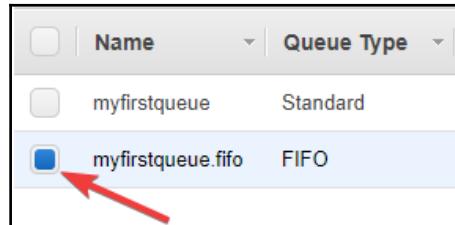


Purge Queues – confirmation

Deleting a queue

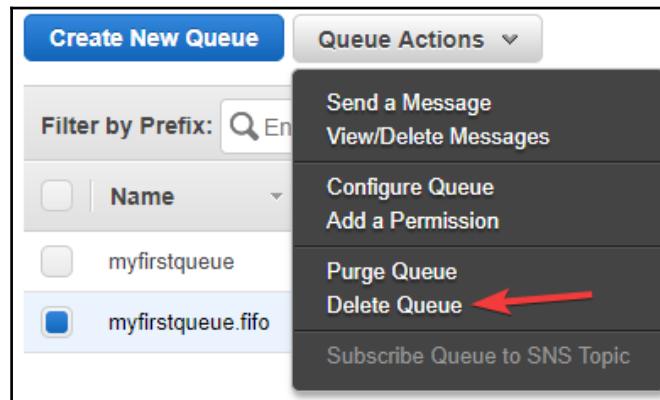
You can perform the following steps to delete a queue:

1. Select a queue from the queue list, as follows:



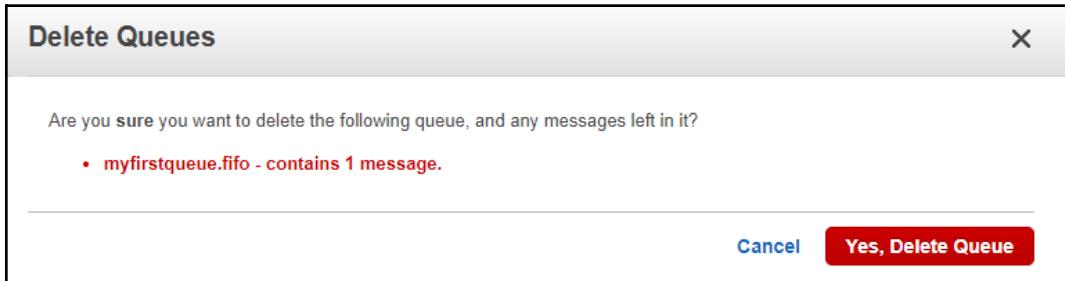
Queue list

2. Click **Queue Actions**, and then **Delete Queue**, as shown:



Queue Actions – Delete Queue

3. The subsequent screen displays a confirmation message with the number of messages the queue contains, as shown. Clicking the **Yes, Delete Queue** button deletes the queue:



Delete Queues – confirmation

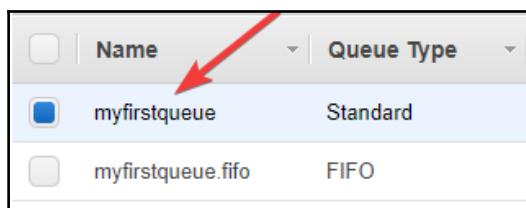
Subscribing a queue to a topic

SQS enables you to subscribe your queues to **Simple Notification Service (SNS)** topics. You can choose from a list of already-available SNS topics and subscribe the queue to that topic. Subscription permission is automatically managed by SQS. When a message is published to a topic in an SNS, that message automatically goes to all the queues subscribed to that topic. For more details on SNS, you can refer to Chapter 13, *Simple Notification Service*.

At present, only standard queues can subscribe to an SNS topic. FIFO queues are presently not supported for topic subscription.

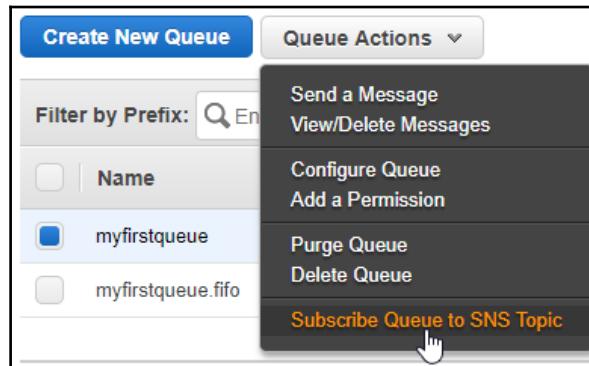
Here are the steps to subscribe a queue to an SNS topic:

1. Select a queue from the queue list, as shown:



Queue list

2. Click **Queue Actions**, and then **Subscribe Queue to SNS Topic**, as shown here:



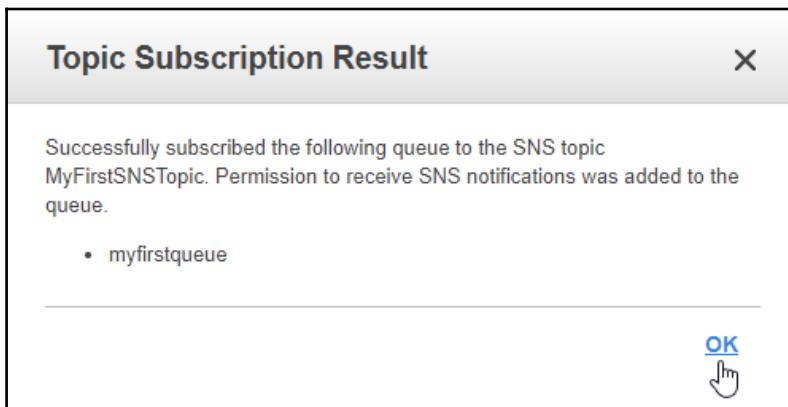
Queue Actions – Subscribe Queue to SNS Topic

3. From the subsequent screen, select the appropriate region where your SNS topic is available and choose a topic from the list. A topic's ARN is automatically populated depending on the topic you select. Alternatively, you can manually type the ARN as required. Finally, you can click the **Subscribe** button, as shown:

The screenshot shows the 'Subscribe to a Topic' dialog box. It has a header 'Subscribe to a Topic' and a close button 'X'. Below the header is a descriptive text: 'Select an SNS Topic from the *Choose a Topic* drop-down or enter a topic's ARN in the *Topic ARN* text box and then press *Subscribe* to allow your queue(s) to receive SNS notifications from the topic and to subscribe your queue(s) to the topic.' There are three input fields: 'Topic Region' with a dropdown menu showing 'US East (N. Virginia)', 'Choose a Topic' with a dropdown menu showing 'MyFirstSNSTopic', and 'Topic ARN' with a text input field containing 'arn:aws:sns:us-east-1:XXXXXXXXXXXX:MyFirstSNSTopic'. Red arrows point from the text descriptions above to each of these three fields. In the bottom right corner, there are 'Cancel' and 'Subscribe' buttons, with a red arrow pointing to the 'Subscribe' button.

Subscribe to a Topic

4. This displays a confirmation dialog box, as shown. Click **OK** and the queue will now be subscribed to the topic:



Topic Subscription Result

Adding user permissions to a queue

SQS allows you to define permissions for your queue. This permission determines the ability of your queue to interact. You can allow or explicitly deny some permissions. The following steps explain how to set permissions for a queue:

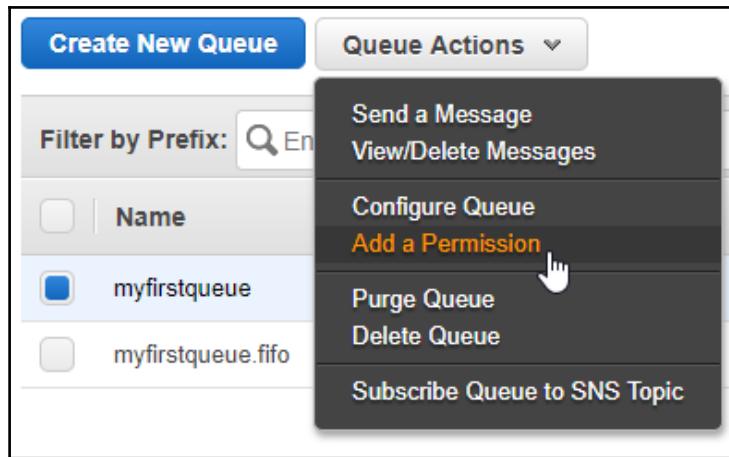
1. Select a queue from the queue list, as shown:

A screenshot of a "Queue list" interface. It shows two rows of queue information. The first row is selected, indicated by a blue square icon next to the name "myfirstqueue". The second row is unselected, indicated by a grey square icon next to the name "myfirstqueue fifo". Both rows have "Standard" and "FIFO" listed under "Queue Type" respectively. A red arrow points to the "myfirstqueue" entry in the first row.

Name	Queue Type
myfirstqueue	Standard
myfirstqueue fifo	FIFO

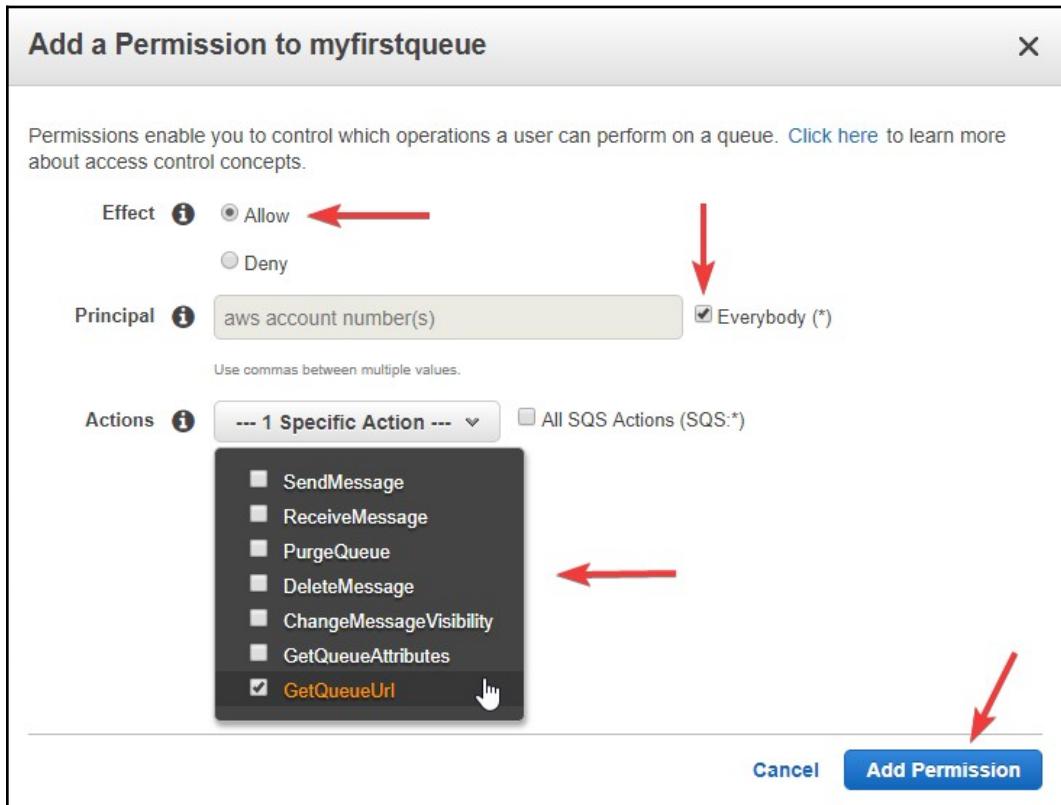
Queue list

2. Click **Queue Actions**, and then **Add a Permission**, as shown:



Queue Actions – Add a Permission

3. In the subsequent screen, for **Effect**, select **Allow** or **Deny** as required. You can specify an account in **Principal**, if you want to assign the permission to a specific person, or select the **Everybody (*)** checkbox to assign the permission to everybody. From the **Actions** tab, you can choose specific actions that you want to assign, or you can select the **All SQS Actions (SQS:*)** checkbox to assign permissions for all actions, as shown:



Adding a permission to a queue

SQS limits

This topic helps us understand the limits with respect to the following:

- Limits related to queues
- Limits related to messages
- Limits related to policies

The following table describes these limits:

Limit	Applies to	Description
Inflight messages per queue	Queues	Standard queue: maximum of 120,000 inflight messages. FIFO queue: maximum of 20,000 inflight messages. SQS returns an <code>OverLimit</code> error if this limit is exceeded.
Queue name	Queues	Queue names can have a maximum of 80 characters. FIFO queue names must end with <code>.fifo</code> . Queue names can have alphanumeric characters, hyphens, and underscores.
Message attributes	Message	A message can have a maximum of 10 metadata attributes.
Message batch	Message	A message batch can have a maximum of 10 messages per batch.
Message content	Message	Message content can include only XML, JSON, and any unformatted text.
Message retention	Message	A queue can retain a message for a minimum of 60 seconds, a maximum of 14 days, and, by default, the message is retained for 4 days.
Message throughput	Message	Standard queue: nearly unlimited TPS. FIFO queue: maximum of 300 messages per second without batch, and 3,000 messages per second with a maximum batch size of 10 messages per operation.
Message size	Message	Minimum size: 1 byte; maximum size: 256 KB.
Message visibility timeout	Message	Maximum 12 hours visibility timeout for a message.
Policy information	Policies	A policy can have a maximum of 8,192 bytes, 20 statements, 50 principals, or 10 conditions.

Queue monitoring and logging

Monitoring SQS queues plays a vital role in an application's life cycle. There are many functions that are dependent on monitoring, using CloudWatch and relevant triggers. For example, you monitor the size of a queue and define a trigger to automatically scale up EC2 instances with an Auto Scaling group. Similarly, you can scale the number of instances serving the consumer process if the size of a queue is smaller. Considering the criticality of monitoring with SQS, CloudWatch and SQS are integrated so that you can easily view and analyze various CloudWatch metrics for SQS queues. Queue metrics can be viewed and analyzed using an SQS console, CloudWatch console, programmatically using APIs, or using the CLI.

Amazon automatically gathers CloudWatch metrics for an SQS queue and pushes it to CloudWatch with an interval of five minutes. Amazon gathers metrics for all active SQS queues. A queue is said to be active for up to six hours if it has messages or if any API action is performed on the queue.

Amazon does not charge you for the SQS metrics collated in CloudWatch. It's provided as part of the SQS queue service without any additional charges. SQS does not support detail with one minute interval monitoring. CloudWatch supports metrics for both types of queues, that is, standard queues as well as FIFO queues.

CloudWatch metrics available for SQS

CloudWatch supports a list of metrics for SQS. Each of these metrics are explained in the following table:

Metrics	Description	Units	Statistics
ApproximateAgeOfOldestMessage	This indicates the approximate age of the oldest message in the queue that is not deleted.	Seconds	Average, Minimum, Maximum, Sum, Data Samples

ApproximateNumberOfMessagesDelayed	This indicates the number of delayed messages in the queue that are not available for reading immediately.	Count	Average, Minimum, Maximum, Sum, Data Samples
ApproximateNumberOfMessagesNotVisible	This displays the number of messages that are in flight.	Count	Average, Minimum, Maximum, Sum, Data Samples
ApproximateNumberOfMessagesVisible	This indicates the number of messages that are available in the queue for retrieval.	Count	Average, Minimum, Maximum, Sum, Data Samples
NumberOfEmptyReceives	This denotes the number of empty responses received by ReceiveMessage API calls.	Count	Average, Minimum, Maximum, Sum, Data Samples
NumberOfMessagesDeleted	This provides the number of messages that are deleted from the queue.	Count	Average, Minimum, Maximum, Sum, Data Samples
NumberOfMessagesReceived	This describes the number of messages that are returned by the ReceiveMessage API calls.	Count	Average, Minimum, Maximum, Sum, Data Samples

NumberOfMessagesSent	This captures the number of messages that are pushed to a queue.	Count	Average, Minimum, Maximum, Sum, Data Samples
SentMessageSize	This calculates the size of messages that are added to a queue.	Bytes	Average, Minimum, Maximum, Sum, Data Samples

Logging SQS API actions

While using SQS, producer and consumer applications perform a number of actions using API calls. It is critical to log all these API calls made by various components of a distributed application. Apart from that, there are many operations performed on SQS queues using SQS consoles; their activity should also be logged. In short, any change or operation performed on any SQS queue using APIs should be logged for auditing and troubleshooting. To cater to this requirement, Amazon has integrated SQS with **CloudTrail**. CloudTrail is a service that captures any API call initiated to perform SQS operations on a queue. Irrespective of the type of SQS queue, CloudTrail records all the activities performed on a queue.

CloudTrail supports the following actions and records a detailed log for these actions:

- AddPermission
- CreateQueue
- DeleteQueue
- PurgeQueue
- RemovePermission
- SetQueueAttributes

Every time any of the preceding actions are performed on a queue, CloudTrail generates a detailed log entry along with the requestor information that initiated the action. With this information, you can verify whether the actions were performed by a root account or an IAM user. It also includes the details if the requested operation was performed with any temporary credentials, a federated user, or any other AWS service. Log files can be stored in an S3 bucket for as long as required. Alternatively, you can define any S3 life cycle rule to archive it to Glacier or delete it.

You can also integrate it with SNS for generating notifications when a new log file is created. It can help you to quickly analyze the log and take any necessary immediate action. AWS also allows you to aggregate SQS log files from more than one AWS region into a single S3 bucket. This makes it easy to keep track of logs and perform an aggregated analysis on the environment.

SQS security

Amazon's SQS is built securely. It requires credentials to initiate any request to SQS queues. Even if you supply credentials while initiating a request to a queue, you may not be able to access it unless you have sufficient permission to access the queues and messages. In this section, let's look at authentication and access control related to SQS queues.

Authentication

AWS allows you to access SQS with any of the following identities:

- Root user
- IAM user
- IAM role
- Federated access
- Cross-account access
- AWS service access
- EC2 applications

Server-Side Encryption (SSE)

Sometimes, it becomes necessary to protect your data using SSE due to some compliance requirement or due to the criticality of the data used in the SQS queue. Amazon provides SSE to protect sensitive data in SQS. SSE helps you to transmit sensitive data in encrypted queues. Amazon uses KMS to manage encryption keys. These keys are used for encrypting the queue.

Messages are encrypted by SSE as soon as the messages are added to the queue. The queue stores these messages in an encrypted format. These messages are decrypted when they are sent to a consumer who is authorized to use it. When using SSE in SQS, requests initiated with the queues must be on the HTTPS protocol.

Summary

- A message queue is a queue of messages exchanged between applications.
- Messages are data objects that are inserted in the queue by sender applications and received by the receiving applications.
- SQS can save an application from getting flooded with messages by storing them into a queue.
- Applications can process the messages without getting interrupted.
- SQS can be used for decoupling application processes.
- SQS provides scalability to an application environment.
- The use of a message queue ensures that a message is delivered at least once, and it is eventually processed by the processing application as long as the process continues reading the queue.
- SQS provides a mechanism to process the data in a predefined order.
- SQS allows you to process the messages asynchronously.
- SQS builds resilience in the system.
- SQS brings elasticity to applications.
- SQS can easily help identify under-performing resources.
- SQS uses something called the put-get-delete paradigm. It requires a consumer to explicitly state that its data has finished processing the message it pulled from the queue. The message data is kept safe with the queue and gets deleted from the queue only after confirmation that it has been processed.
- SQS ensures that a message is delivered at least once in a standard queue and it ensures that a message is delivered exactly once in a FIFO queue.
- When a client picks up a message from the queue, SQS locks up that message until the client confirms that it has completed processing the message or the operation times out and SQS has not receive confirmation from the consumer.
- SQS provides options to configure each queue independently.

- SQS supports a maximum message size of 256.
- SQS allows you to control producers and consumers that can send and receive messages to or from the queue.
- Delay time ensures that a message inserted in the queue is postponed for the time configured as delay time in the queue.
- SQS supports PCI and HIPAA compliance.
- There are two types of SQS queues, that is, standard queues and FIFO queues.
- A standard queue generally sends the data in the same order as it receives it; however, on certain occasions, the order may change.
- The order in which the data is sent is fixed in FIFO queues.
- A DLQ is used by other queues for storing failed messages that are not successfully consumed by consumer processes.

13

Simple Notification Service (SNS)

AWS **Simple Notification Service (SNS)** works based on push technology. It is also called a server push. In this mechanism, the message or transaction is initiated by the publisher or a central server and AWS SNS delivers it to the subscribers. It is the opposite of the pull mechanism. The pull mechanism is also called a client pull, where the client raises a request to fetch or pull data from the server. As a side note, unlike SNS, AWS SQS works on a pull mechanism.

In this chapter, we will cover the following topics:

- Introducing Amazon SNS
- Amazon SNS fanout
- Application and system alerts
- Creating an Amazon SNS topic
- Publishing a message to an SNS topic
- Deleting an SNS topic
- Managing access to an SNS topic
- An architectural overview
- SNS best practices

Introducing Amazon SNS

Amazon SNS is a managed notification service. It works on a push mechanism—the **publisher** raises a request to send a message to the **subscriber**. *Figure 13.1* shows us how it works:

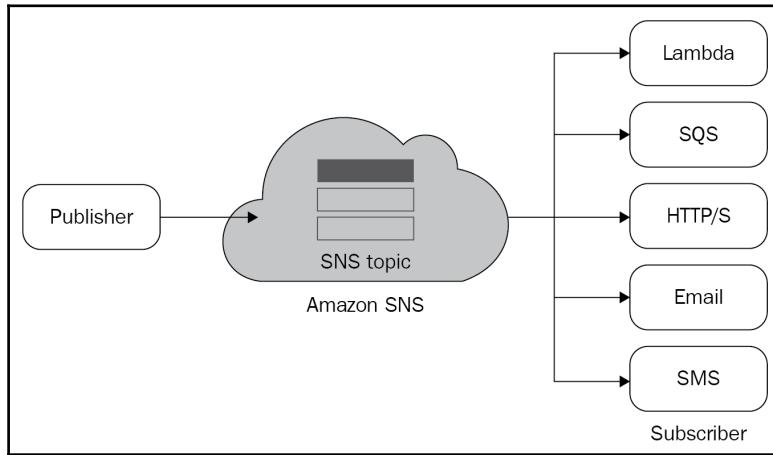


Figure 13.1: Introduction to SNS reference

First, you need to create an Amazon **SNS topic**. An SNS topic acts as an access point between the publisher and subscriber applications. The publisher communicates asynchronously with the subscribers, using SNS. A subscriber can be an entity, such as a Lambda function, SQS, an HTTP or HTTPS endpoint, email, or a mobile device that subscribes to an SNS topic for receiving notifications. To receive notifications, subscribers must specify the protocol (that is, **HTTP**, **HTTPS**, **Email**, **Email-JSON**, **Amazon SQS**, **Application**, **AWS Lambda**, or **SMS**). When a publisher has new information to share with the subscribers, it publishes a message to the topic. Finally, SNS delivers the message/notification to all subscribers.



The topic policy controls who can publish messages and subscribe to a topic.

In enterprise architectures, we often need to send notifications to the subscribers. Some of the following real-time notification use cases can help us to understand how and where they are used:

- When an EC2 instance is under- or over-utilized for a specific time frame, it should send a notification to the system administrators and stack holders. For example, at any given time, when the average CPU usage is above 70% or below 30% for a specific time frame, it sends a notification to the system administrator or autoscale, scale-out or scale-in the number of EC2 instances, as per the configuration.
- Mobile applications installed on a smartphone occasionally send push notifications with various offers and discounts, based on the user's interaction with the mobile application.
- To send a mobile application push notification, the end user may need to install an application on the smartphone. On the other hand, SMS notifications can be sent to any mobile device. It is irrelevant whether the end user has installed your mobile application.

When any new message is published to an SNS topic, it can invoke a subscribed Lambda function. Some of the example use cases are as follows:

- Serverless architecture
- Automated backups and other daily administrative tasks
- Processing uploaded objects in S3 buckets

Amazon SNS fanout

Amazon SNS and Amazon SQS are key services to create loosely coupled, scalable, cloud-based, and serverless applications in the cloud. One of the common architecture concepts is **fanout**. In this concept, several Amazon SQSs act as a subscriber. A publisher sends a message to an SNS topic and it distributes this topic to many SQS queues in parallel. The diagram given in *Figure 13.2* describes the concept of fanout in SQS. For example, a virtual, multinational, e-commerce company introduces hundreds of products for sale in many countries every day. Now, when a new product is saved in a web application, it will send a message to the SNS topic. Immediately, the SNS topic sends notifications in parallel to all subscriber SQS queues:

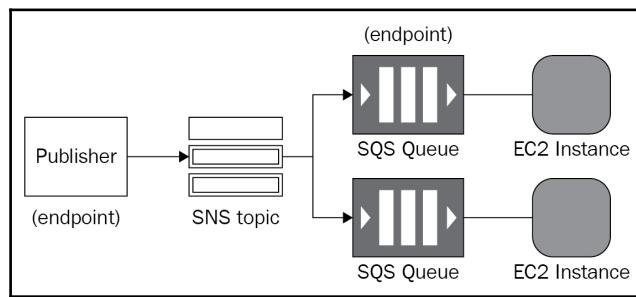


Figure 13.2: Amazon SNS fanout

Let's explore a scenario to better understand this point.

Consider a scenario in which there is a web application hosted in multiple regions for a number of countries. Each web application is hosted on a specific web endpoint, with URLs ending with a country-specific **Top Level Domain (TLD)**, such as **.in** for India, **.uk** for UK, and **.nz** for New Zealand. The global site is hosted on the **.com** domain. When a new product is introduced at the **.com** portal, it sends a push notification to all the subscriber SQS queues. Applications hosted in different countries read the queue on a periodic basis and update the product details. The message fans out from the SNS to multiple SQSs, and finally, to several country-specific web applications.

In certain scenarios, web applications can directly subscribe to the SNS topic with the HTTP or HTTPS endpoint URL; however, that is not a fanout mechanism. It's a direct communication between the SNS and web application endpoint.

Application and system alerts

It is important to monitor AWS resources over various parameters (that is, **CPUUtilization**, **MemoryUtilization**, **NetworkIn**, **NetworkOut**, and so on) to avoid bottlenecks and to deliver consistent web application performance to the end user. As resource consumption crosses the defined threshold, the administrator should get an alert. This alert can be in the form of an SMS and/or email to the system, network, or a database administrator, based on the resource type. Other AWS services also use SNS to send notifications on certain events. For example, an autoscaling group can also optionally inform the administrator upon scale-out and scale-in using SNS notifications. Subsequent points describe mobile device push notifications, push emails, and text messaging.

Mobile device push notifications

Mobile push notifications send a notification directly to mobile applications. User interaction with the mobile application helps applications to understand the subscriber's interests, and accordingly, occasional notifications are sent to their mobile device to update them about offers on products, services, or company news.

Push emails and text messaging

Email and text messages (SMS) are two common ways to convey important messages to an individual or groups of people. Usually, with this method, subscribers get notifications by email or SMS on their subscribed email address or phone number, respectively. This notification contains important messages and URLs to get more information.

Creating an Amazon SNS topic

First, it is essential you create an SNS topic. Then, it is possible for a publisher to publish a message and for subscribers to subscribe to get a notification. New SNS topics can be created by performing the following steps:

1. Sign in to the AWS account with the IAM user who has sufficient privileges to work with Amazon SNS. Once you have successfully logged in to the AWS account, make sure that the appropriate AWS region is selected from the right-hand side top toolbar, as shown in *Figure 13.3*:



Figure 13.3: AWS web console; select desired region

2. From the AWS dashboard, select **Simple Notification Service** from the **Application Integration** services group, as shown in *Figure 13.4*:



Figure 13.4: AWS web console; select SNS

3. In the text box, provide an appropriate (that is, meaningful) name for the topic and select **Next step** from the **SNS dashboard**, as shown in the following screenshot:

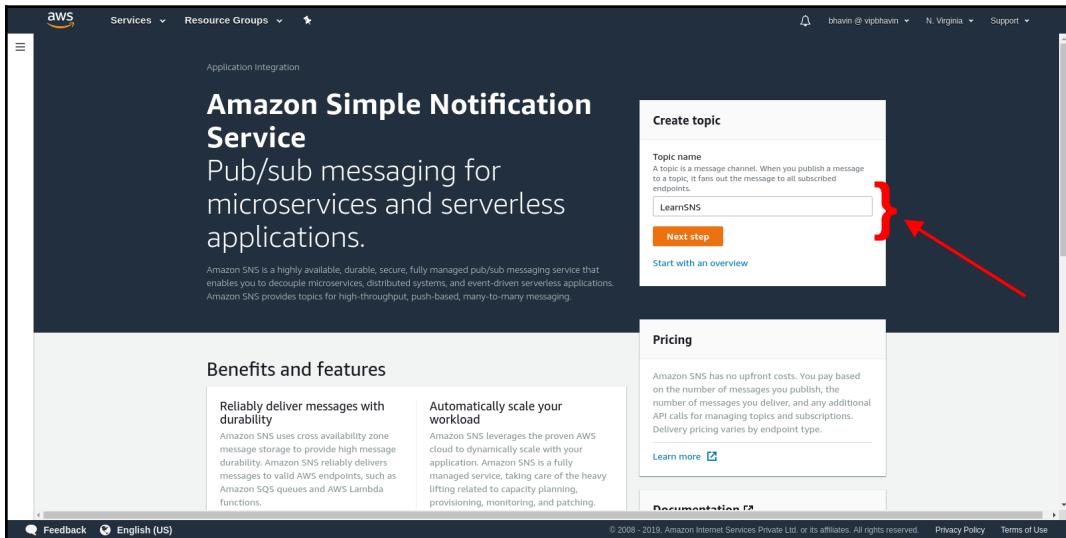


Figure 13.5: SNS dashboard; select Create topic

4. In the **Create new topic** pop-up box, provide the appropriate input and click on **Create topic**, as shown in *Figure 13.6*:

Amazon SNS > Topics > Create topic

Create topic

Details

Name
LearnSNS
Maximum 256 characters. Can include alphanumeric characters, hyphens (-) and underscores (_).

Display name - *optional*
To use this topic with SMS subscriptions, enter a display name. Only the first 10 characters are displayed in an SMS message. [Info](#)
LEARN-SNS
Maximum 100 characters, including hyphens (-) and underscores (_).

► **Encryption - *optional***
Amazon SNS provides in-transit encryption by default. Enabling server-side encryption adds at-rest encryption to your topic.

► **Access policy - *optional***
This policy defines who can access your topic. By default, only the topic owner can publish or subscribe to the topic. [Info](#)

► **Delivery retry policy (HTTP/S) - *optional***
The policy defines how Amazon SNS retries failed deliveries to HTTP/S endpoints. To modify the default settings, expand this section. [Info](#)

► **Delivery status logging - *optional***
These settings configure the logging of message delivery status to CloudWatch Logs. [Info](#)

► **Tags - *optional***
A tag is a metadata label that you can assign to an Amazon SNS topic. Each tag consists of a key and an optional value. You can use tags to search and filter your topics and track your costs. [Learn more](#)

Cancel **Create topic**

Figure 13.6: Provide appropriate topic name and display name

The fields in the preceding screenshot are explained as follows:

- **Topic name:** This can be up to 256 characters in size. Alphanumeric, underscores (_), and hyphens (-) are allowed. For example, LearnSNS.

- **Display name:** This is only mandatory for the SMS protocol. A maximum of 10 characters are allowed. For example, LEARN-SNS.

- Upon the successful creation of the SNS topic, the **Topic** counter changes from **1** to **0** on the AWS SNS dashboard, as shown in *Figure 13.7*:

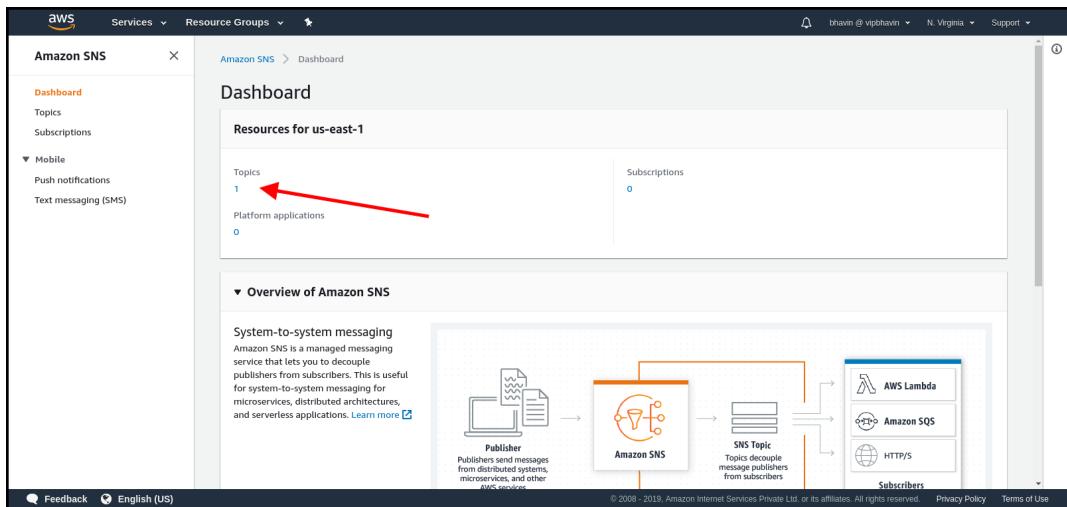


Figure 13.7: SNS dashboard

- To see a complete list of SNS topics in a specific AWS region, select **Topics** from the left-hand side pane on the SNS dashboard, as shown in *Figure 13.8*:

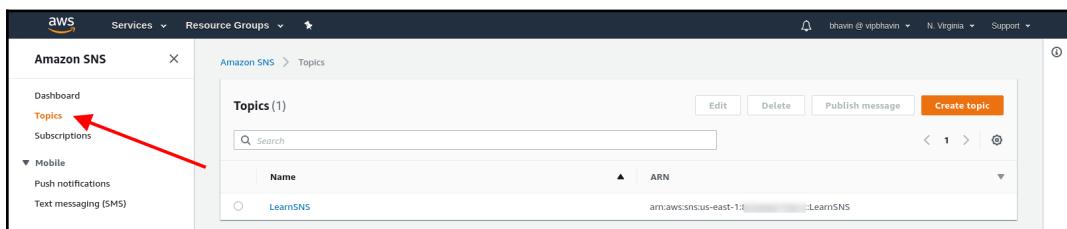


Figure 13.8: SNS dashboard; select topics

Copy the SNS **Topic ARN**, as shown in the preceding screenshot. This ARN is used in the subsequent steps for subscribing to the topic.

Subscribing to an SNS topic

Each SNS topic can have multiple subscribers. Each subscriber may use the same protocols or different protocols. Copy the ARN of the recently-created SNS topic. This step-by-step guide requires you to subscribe to a topic. Subscribers can receive a notification for a desired protocol as and when the publisher sends any message to the same topic. The steps for subscribing to an SNS topic are as follows:

1. Go to the SNS dashboard and select **Subscriptions**, as shown in *Figure 13.9*:

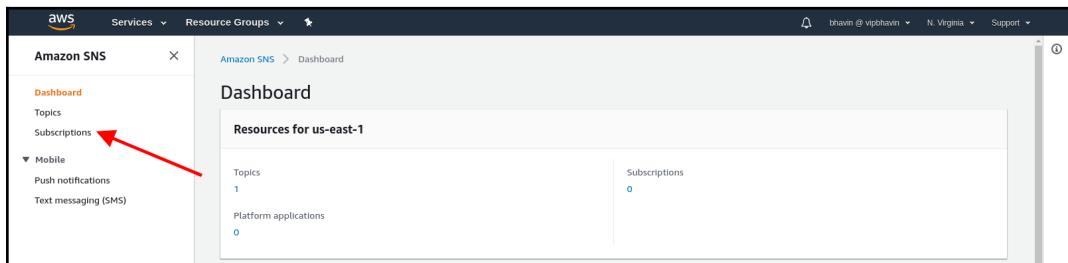


Figure 13.9: SNS dashboard; select subscriptions

2. Click on **Create subscription**, as shown in *Figure 13.10*:

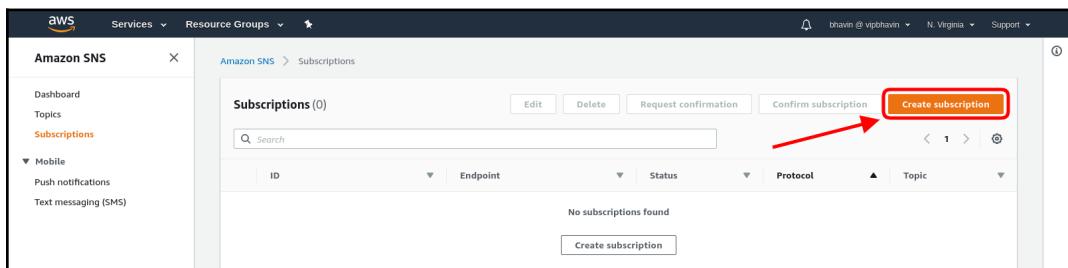


Figure 13.10: Create subscription

3. A popup will appear, as shown in *Figure 13.11*. Provide a valid **Topic ARN**, **Protocol**, and **Endpoint**. Finally, click on **Create subscription**:

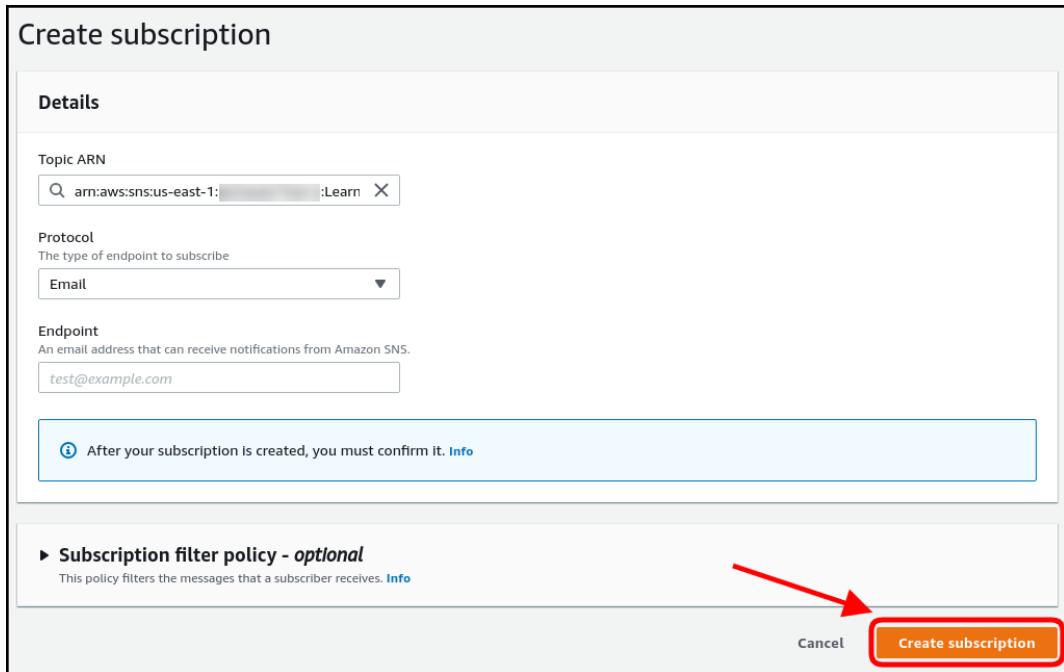


Figure 13.11: Create subscription

The fields in the preceding screenshot are explained as follows:

- **Topic ARN:** Paste the copied ARN of the recently-created SNS topic
- **Protocol:** Select the appropriate protocol (**HTTP**, **HTTPS**, **Email**, **Email-JSON**, **Amazon SQS**, **Application**, **AWS Lambda**, or **SMS**)
- **Endpoint:** According to the protocol, provide an appropriate endpoint (that is, domain name, email address, SQS ARN, application ARN, Lambda function ARN, or phone number, respectively)

Amazon SNS will send an email on a specified endpoint to confirm the subscription. Click the link in the email. It will lead to a web browser that displays a confirmation of a subscription response from Amazon SNS.

Publishing a message to an SNS topic

As soon as the publisher sends a message to a topic, Amazon SNS will try to deliver a notification/message to all the subscribers. The subscriber may have different protocols and individual endpoints. With the help of the following steps, a message can be published over an SNS topic:

1. Go to the SNS dashboard and select **Topics** from the left-hand side pane; it will display a list of topics, as shown in *Figure 13.12*:

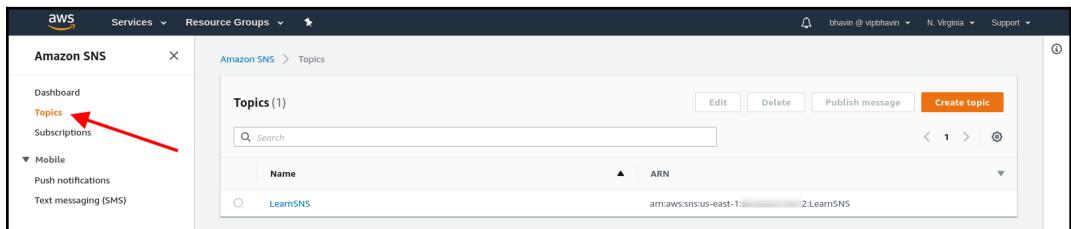


Figure 13.12: SNS Topics dashboard

2. Click on the topic **Name**, as shown in *Figure 13.13*:

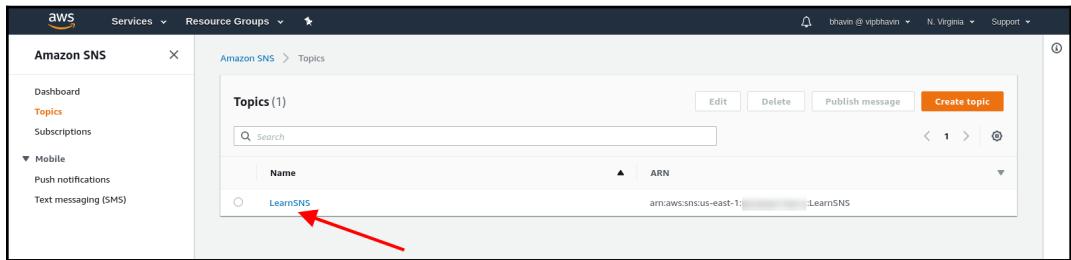


Figure 13.13: Select topic; to get ARN

3. Click on **Publish message**, as shown in *Figure 13.14*:

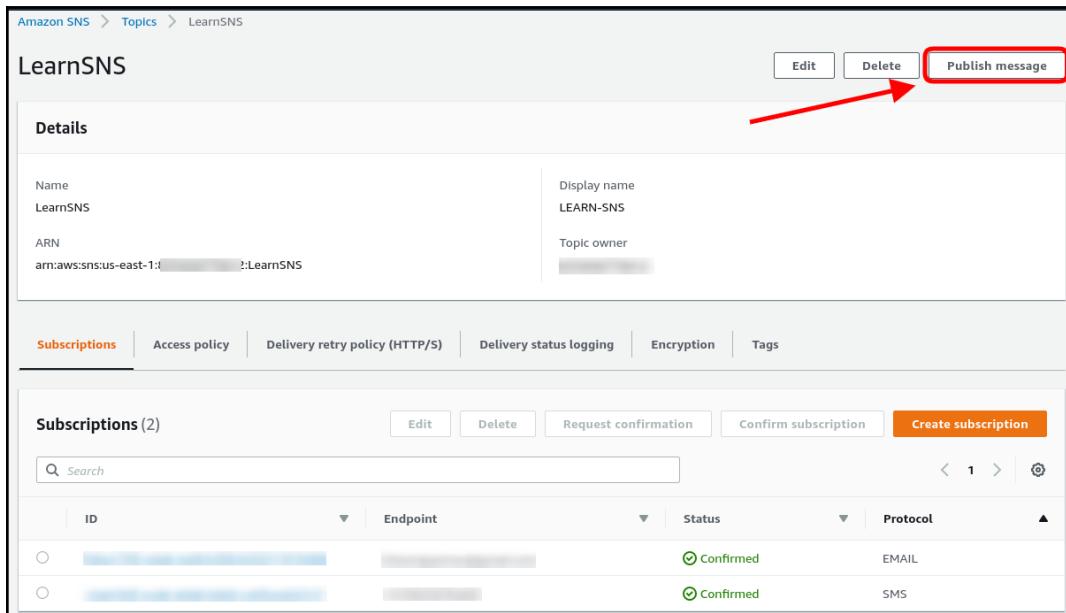


Figure 13.14: SNS topic details

4. To publish a message, provide the details, as shown in *Figure 13.15*:

Amazon SNS > Topics > LearnSNS > Publish message

Publish message to topic

Message details

Topic ARN
arn:aws:sns:us-east-1:::LearnSNS

Subject - optional

Maximum 100 printable ASCII characters

Time to Live (TTL) - optional
This setting applies only to mobile application endpoints. The number of seconds that the push notification service has to deliver the message to the endpoint. [Info](#)

Message body

Message structure

Identical payload for all delivery protocols.
The same payload is sent to endpoints subscribed to the topic, regardless of their delivery protocol.

Custom payload for each delivery protocol.
Different payloads are sent to endpoints subscribed to the topic, based on their delivery protocol.

Message body to send to the endpoint

Message attributes

Message attributes let you provide structured metadata items (such as timestamps, geospatial data, signatures, and identifiers) for the message. [Info](#)

Type	Name	Value
<input type="button" value="Select attribute type"/>	<input type="text" value="Enter attribute name"/>	<input \"value2\"]"="" type="text" value="value or [\" value1\",=""/>
<input type="button" value="Remove"/>		
<input type="button" value="Add another attribute"/>		

Figure 13.15: Publish a message on a topic

The fields in the preceding screenshot are explained as follows:

- **Subject:** This is optional and can be up to 100 printable ASCII characters. In the case of an email notification, it will appear as an email subject line.
- **Time to live (TTL):** This is in seconds and will be the same for all mobile platforms' push notifications. It is an additional capacity to configure TTL, apart from the existing capacity of setting TTL within the SNS message body. It specifies the time to expire metadata about a message. Within a specified time, **Apple Push Notification Service (APNS)** or **Google Cloud Messaging (GCM)** must deliver messages to the endpoint. If the message is not delivered within the specified time frame, then the message will be dropped and no further attempts will take place to deliver the message.
- **Message structure:** This can be either **Identical payload for all delivery protocols** (that is, a raw message) or **Custom payload for each delivery protocol** (that is, a JSON message):
 - **Identical payload for all delivery protocols:** This is the actual plain text message to send to all the subscribers, as shown in the following screenshot:

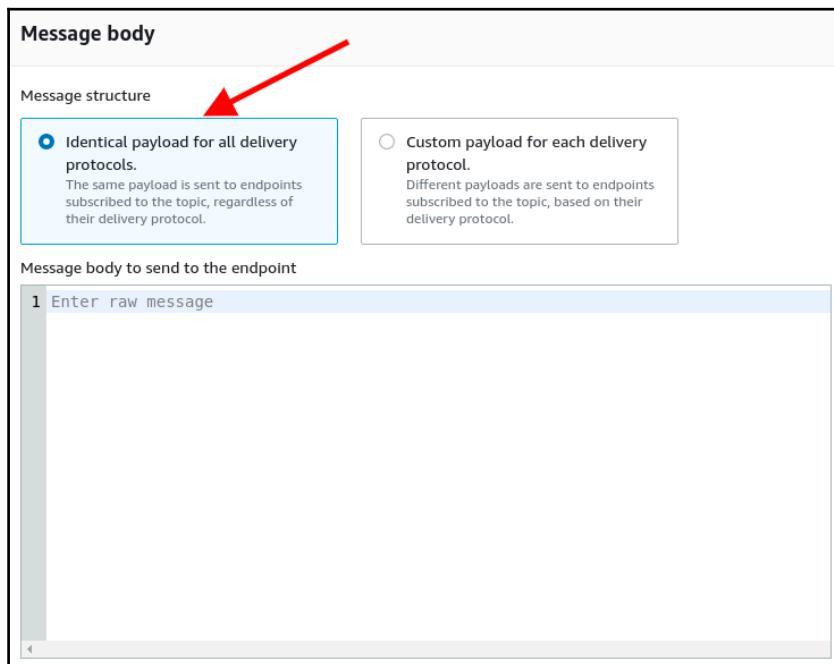


Figure 13.16: Identical payload for all delivery

- **Custom payload for each delivery protocol:** This is the formatted message to customize the message for each of the protocols, as shown in the following screenshot:

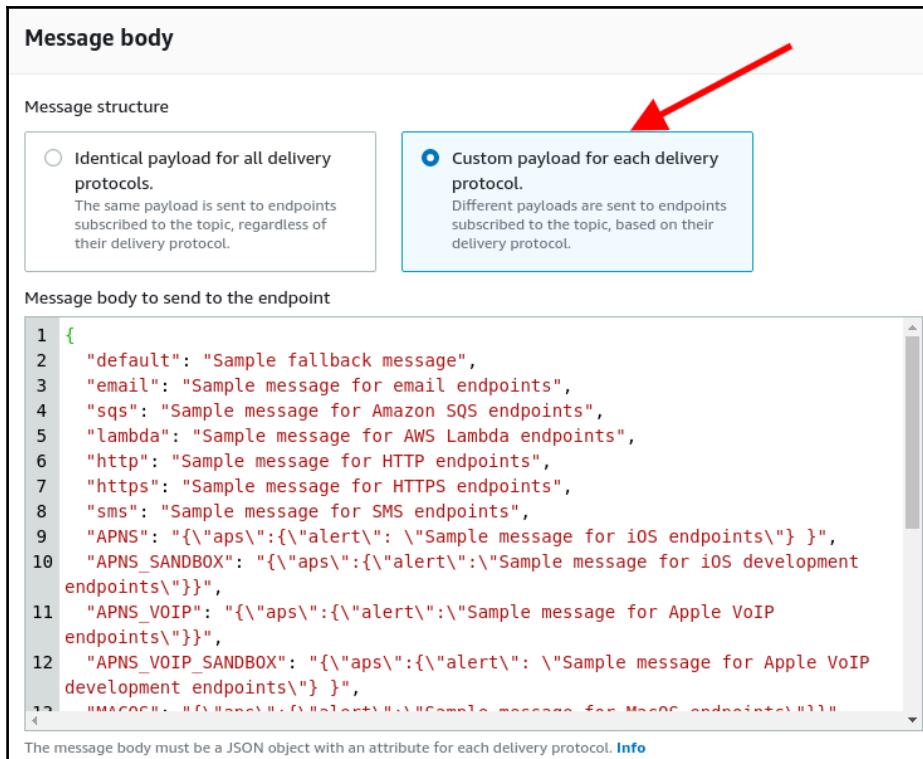


Figure 13.17: Custom payload for each protocol

- **Message body:** Based on the message format type, the actual message will be either plaintext or JSON. SMS messages can have up to 160 ASCII or 70 Unicode characters, while email messages can be up to 256 KB in size.

- **Message attributes:** This makes it possible to configure structured metadata for the message being published, such as timestamps, signatures, and identifiers. To configure such attributes, the data type for the same can be **String**, **String.Array**, **Number**, or **Binary**, as shown in the following screenshot:

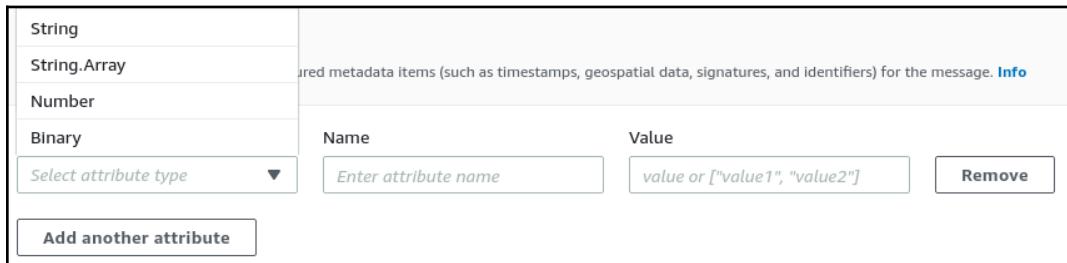


Figure 13.18: Message attribute types drop-down menu

- Finally, the message that you want to publish should look like the one in *Figure 13.17*. To publish a notification that sends an email and SMS to the subscribers, click on the **Publish message** button, as shown in *Figure 13.19*:

Amazon SNS > Topics > LearnSNS > Publish message

Publish message to topic

Message details

Topic ARN

arn:aws:sns:us-east-1:XXXXXXXXXX:LearnSNS

Subject - optional

Test subject line.

Maximum 100 printable ASCII characters

Time to Live (TTL) - optional

This setting applies only to mobile application endpoints. The number of seconds that the push notification service has to deliver the message to the endpoint. [Info](#)

30

Message body

Message structure

Identical payload for all delivery protocols.

The same payload is sent to endpoints subscribed to the topic, regardless of their delivery protocol.

Custom payload for each delivery protocol.

Different payloads are sent to endpoints subscribed to the topic, based on their delivery protocol.

Message body to send to the endpoint

```

1 {
2   "default": "Test notification from Amazon SNS.",
3   "email": "Test notification from Amazon SNS.",
4   "sns": "Test notification from Amazon SNS.",
5   "lambda": "Test notification from Amazon SNS.",
6   "http": "Test notification from Amazon SNS.",
7   "https": "Test notification from Amazon SNS.",
8   "sms": "Test notification from Amazon SNS.",
9   "APNS": "{\"aps\":{\"alert\": \"Test notification from Amazon SNS.\\"}}",
10  "APNS_SANDBOX": "{\"aps\":{\"alert\":\"Test notification from Amazon
SNS.\\"}}",
11  "APNS_VOIP": "{\"aps\":{\"alert\":\"Test notification from Amazon SNS.\\"}}",
12  "APNS_SANDBOX": "{\"aps\":{\"alert\": \"Test notification from Amazon
SNS.\\"}}",
13  "MACOS": "{\"aps\":{\"alert\":\\\"Test notification from Amazon SNS.\\"}}",
14  "MACOS_SANDBOX": \"\\\"Test notification from Amazon SNS.\\"}

```

The message body must be a JSON object with an attribute for each delivery protocol. [Info](#)

Message attributes

Message attributes let you provide structured metadata items (such as timestamps, geospatial data, signatures, and identifiers) for the message. [Info](#)

Type	Name	Value	Remove
String	Signature	LearnSNS	Remove
Add another attribute			

[Cancel](#)

[Publish message](#)

Figure 13.19: Publish a message

Deleting an SNS topic

To delete an SNS topic, you first need to unsubscribe the subscribers, and then you can delete the SNS topic. The following steps explain how to unsubscribe subscribers and delete an SNS topic:

1. Go to the Amazon SNS dashboard and click on **Subscriptions**, as shown in *Figure 13.20*:



Figure 13.20: SNS dashboard

2. Before you can delete a topic, you need to unsubscribe all the subscribers from that topic. Select all the relevant subscribers for the topic and click on **Delete** from the menu, as shown in *Figure 13.21*:

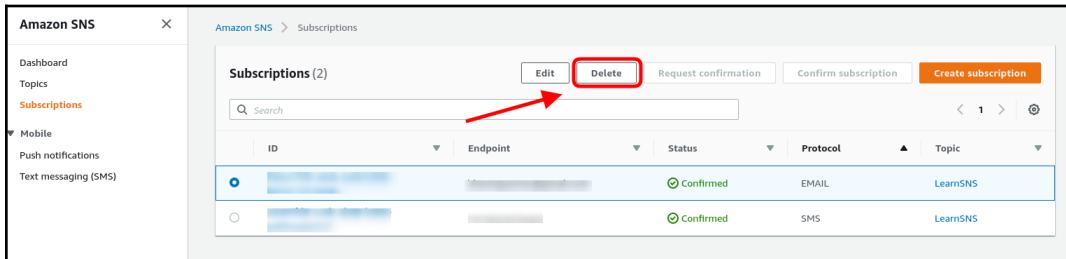


Figure 13.21: Delete subscriptions

3. Before unsubscribing the selected subscribers, click on **Delete** in the confirmation dialog box, as shown in *Figure 13.22*:

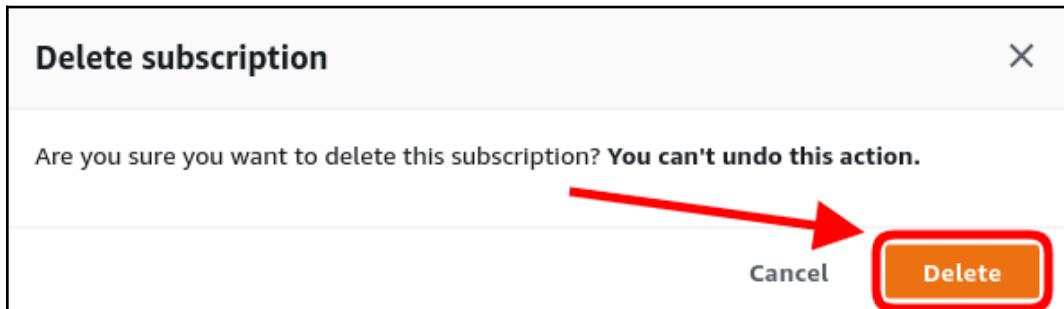


Figure 13.22: Popup; delete subscriptions confirmation



All the subscriptions that are pending for confirmation cannot be deleted manually. They are automatically deleted after three days if the confirmation is not received for those subscriptions.

4. Select **Topics** from the left-hand side pane, as shown in *Figure 13.23*:

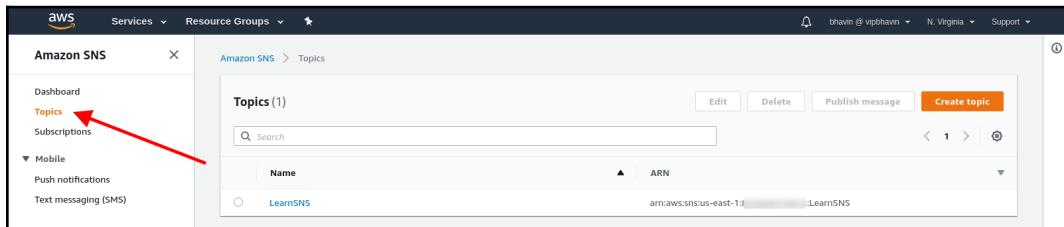


Figure 13.23: SNS subscriptions dashboard; select topics

5. Select a desired SNS topic to delete and click on **Delete** from the menu, as shown in *Figure 13.24*:

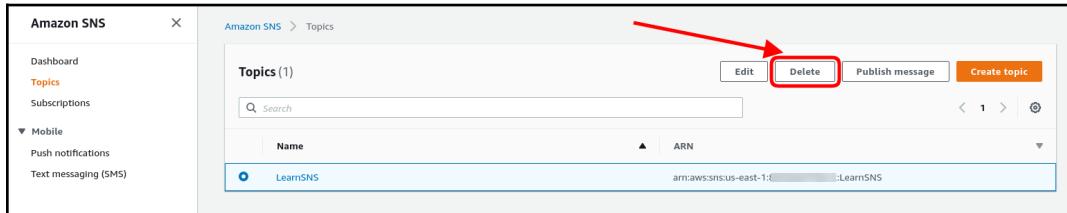


Figure 13.24: SNS topics dashboard; select desired topic to delete

Managing access to Amazon SNS topics

Amazon SNS supports multiple protocols, such as **HTTP**, **HTTPS**, **Email**, **Email-JSON**, **Amazon SQS**, **Application**, **AWS Lambda**, and **SMS**. SNS subscribers can receive the message or notification over one of the supported protocols. Apart from the protocols, SNS also provides the **topic policy**, which can be used to control who can subscribe to or publish to a topic. The subsequent section will describe when to use the topic policy for access control over an SNS topic.

When to use access control

The access control policy helps to define the way to control access to an SNS topic. There can be a number of scenarios where you may need to use the access control policy for an SNS topic. Here are some examples:

- You can use the access control policy when you want to allow an IAM user to publish a message to one or more SNS topics. This IAM user can be in the same AWS account, or a different one.
- SNS topics allow subscribers to use multiple supported protocols. With the help of the access control policy, subscribers can be restricted to use one or more specific protocols. For example, on an SNS topic, you can allow the subscriber to use only email and HTTPS.
- You can also define the access control policy to restrict an SNS topic to only publish a message to an SQS queue.

Key concepts

Understanding the following key concepts is essential to effectively writing the access policy:

- **Permission:** A permission is used for allowing or disallowing access to a specific resource. A permission can be either **allow** or **deny**.
- **Statement:** A statement describes a single permission written in an access policy language, such as JSON. One or more statements are part of a policy.
- **Policy:** A policy is a JSON document; this includes one or more statements. *Figure 13.25* helps to illustrate the concept of single and multiple statements in a policy:

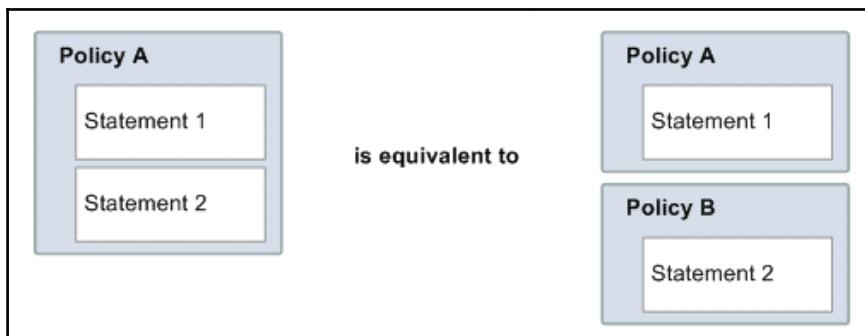


Figure 13.25: Policy and statements

For example, one statement allows Jack to subscribe to `TestTopic` using email protocol only. Another statement restricts Adam to publishing messages to `TestTopic`. As shown in the preceding diagram, these two statements can be placed in a single policy, or they can be placed separately in each policy.

- **Issuer:** In general, an issuer is a resource owner. A person who writes the access policy on a resource is called the **issuer**. For example, Bob has created an SNS topic named `TestTopic`. He has full privileges to `TestTopic`. AWS doesn't allow other IAM users to write an access policy on it. Bob can write an access policy to specify who can publish or subscribe to `TestTopic`, and with which protocols.

- **Principal:** When an issuer writes an access policy, the person to whom the privilege is granted/restricted in the policy is called a principal. It can be an actual identity, such as a username, or an IP address, such as a CIDR range. A principal can be anyone, just by mentioning *. For example, in the statement, Jack will subscribe to TestTopic using email protocol only, Jack is a principal.
- **Action:** An action specifies the activity that a principal can perform on a resource. One or more actions can be specified in a policy.
- **Resource:** This is an object (that is, SNS topic) that a principal is requesting to access. For example, in the statement, Jack will subscribe to TestTopic using email protocol only, TestTopic is a resource.
- **Conditions and keys:** Using conditions, you can apply specific restrictions on the permission. For example, we can write a statement in a policy that will allow Jack to subscribe to a topic; however, it adds a condition that allows Jack to subscribe to the topic only using email protocol.
A key is a specific characteristic, such as the date and time. It acts as a base for an access restriction. Keys and conditions are used in a pair, in order to define restrictions. For example, when the issuer wants the principal to deny access to a resource before January 1, 2018, then the condition to be used is DateLessThan, the key should be aws:CurrentTime, and the value is set to 2018-01-01T00:00:00Z.
- **Requester:** The person or entity sends the access request to the resource, called a requester. For example, the requester asks the AWS service: *Will you allow me to do B to C, where D applies?*
- **Evaluation:** An evaluation is a process to conclude whether to allow or deny a requester, based on the applicable policies. This topic is elaborated in the *Accessing request evaluation logic* section.
- **Effect:** Possible values for an effect can be deny or allow. At the time of the policy evaluation, it helps to decide whether the requester can perform an action against configured conditions.
- **Default deny:** When a policy statement is evaluated, if a permission is not explicitly allowed to perform an action, the statement considers it as deny, by default. In short, if any permission is not explicitly allowed in the policy statement, the user is denied the permission for these actions, which are not defined in the policy.

- **Allow:** Policy evaluation is `allow` when a policy statement has `effect=allow` and the defined conditions are met; a requester can perform an action on the resource.
- **Explicit deny:** Policy evaluation is `deny`; when a policy statement has `effect=deny` and the defined conditions are met; a requester cannot perform an action on the resource.

Architectural overview

Figure 13.26 can help you to understand the architectural overview at a high level. It illustrates this from the beginning, from when the resource is created until the `allow` or `deny` permission is evaluated:

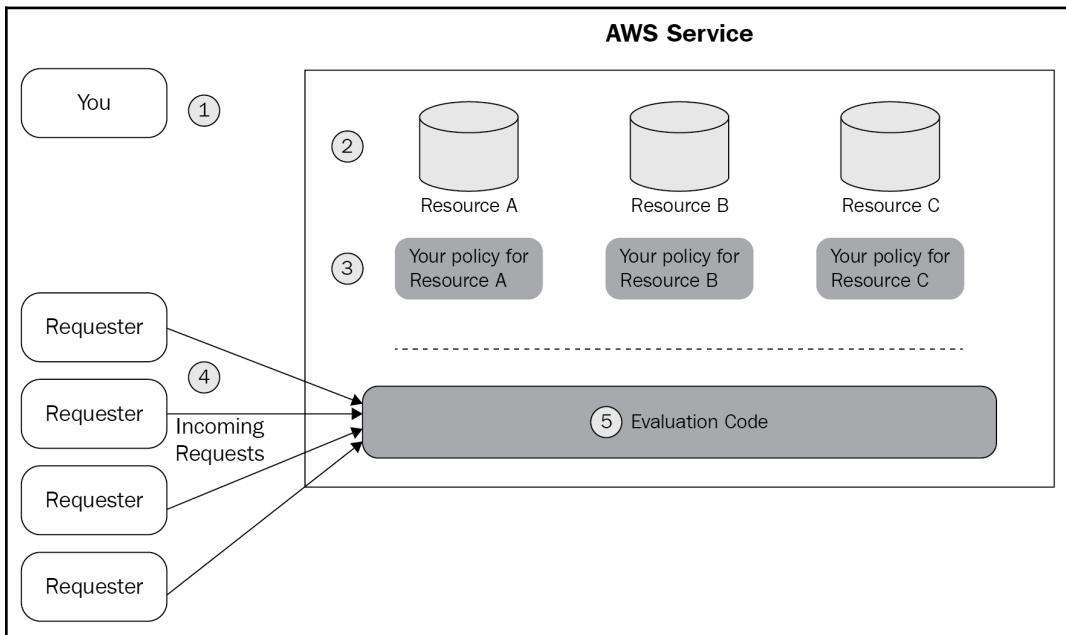


Figure 13.26: Architectural overview (policy evaluation)

Each of the points given in *Figure 13.24* is explained as follows. Readers should refer to the preceding diagram and co-relate the point numbers given in the diagram:

1. A user creates an AWS resource. For example, Bob creates SNS topics. Bob is the owner for SNS topics.
2. Topics are created within AWS SNS.
3. An owner, also called an issuer, creates an access policy. Usually, one policy with one or more statements is created, rather than multiple policies, as it is easy to manage.
4. Requests are incoming from the requesters to AWS SNS. Requesters can be subscribers or publishers.
5. All incoming requests to access AWS resources (that is, in this case, SNS topics) are evaluated against applicable policies, and it is determined whether the requester can access the resource. Evaluation is carried out by the access policy language evaluation code.

Accessing request evaluation logic

Whenever any request to access an AWS resource is initiated, policy evaluation logic evaluates the related policies to determine whether to allow an incoming request, or deny it. Basic policy evaluation rules are given here:

- First and foremost, policy evaluation logic applies to the default deny rule. That means that apart from the resource owner, all other requests are denied if no explicit `allow` permission is specified in the policy.
- Explicit allow statements or a policy override the default deny permission. As a result, the request gets access.
- Explicit deny statements or a policy override the explicit allow statement.

The following flowchart illustrates in detail how a decision is made on whether to allow or deny:

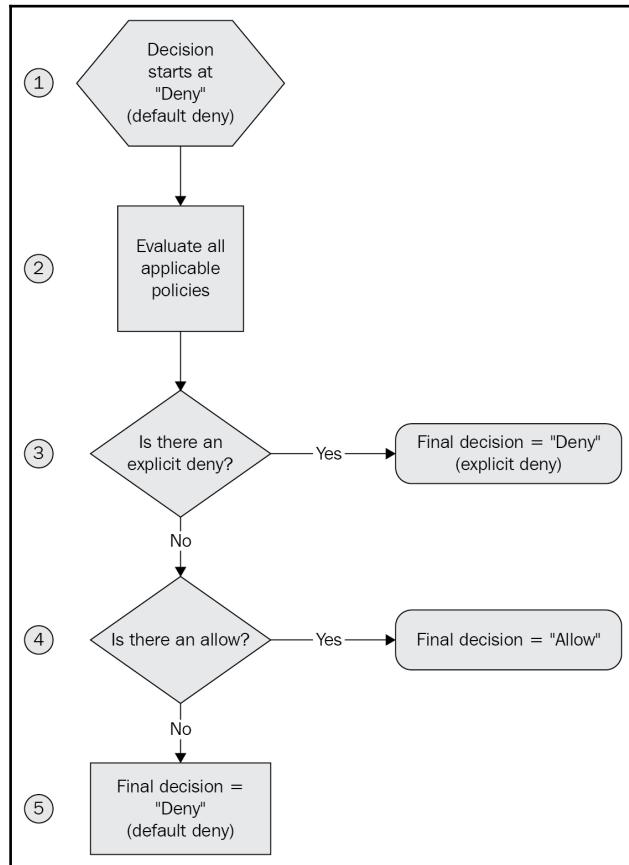


Figure 13.27: Policy evaluation flowchart

The policy evaluation flowchart is explained as follows:

1. By default, the **default deny** rule applies. Any request apart from the owner is denied.
2. AWS evaluates a policy based on the evaluation logic, relevant resources, principal, and conditions specified in the policies. If there is more than one policy associated with a resource, the order of the policy evaluation is not important. Any policy can be evaluated first, irrespective of the order in which they are associated with the resource.

3. During the evaluation of policies, if any policy has an explicit deny, the final decision is to deny the request.
4. If there is no explicit deny specified in the policy and there is an explicit allow available in the policy, the final decision is to allow the request.
5. If there is no explicit allow or deny policy, the **default deny** rule is applied.

Invoking the Lambda function using SNS notifications

An AWS Lambda function can be invoked with Amazon SNS notifications. When a publisher publishes a message to an SNS topic and a Lambda function is subscribed to the same SNS topic, that Lambda function is invoked with the payload of a published message. When a publisher publishes a message to an SNS topic, SNS provides the message delivery status to the publisher, stating that the message is sent to the Lambda function. The payload message acts as an input parameter for the Lambda function. The function can process the message (payload) as needed.

A prerequisite is to have an SNS topic and a Lambda function.

An SNS topic can be configured to execute a Lambda function with the help of the following steps:

1. Go to the SNS dashboard and select **Topics** from the left-hand side panel; it will display a list of topics, as shown in *Figure 13.28*:

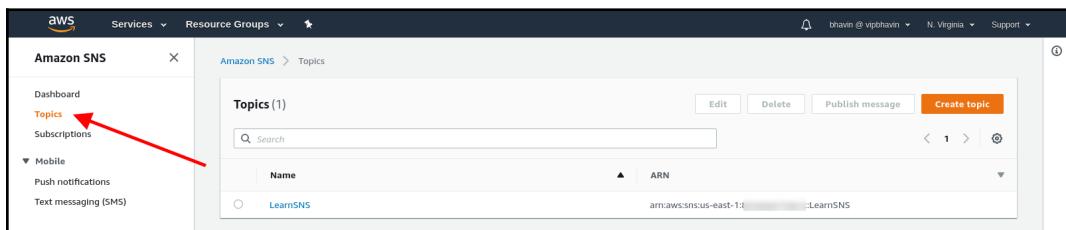


Figure 13.28: SNS topics dashboard

2. Click on the topic **Name** for a topic to subscribe a Lambda function, as shown in the following screenshot:

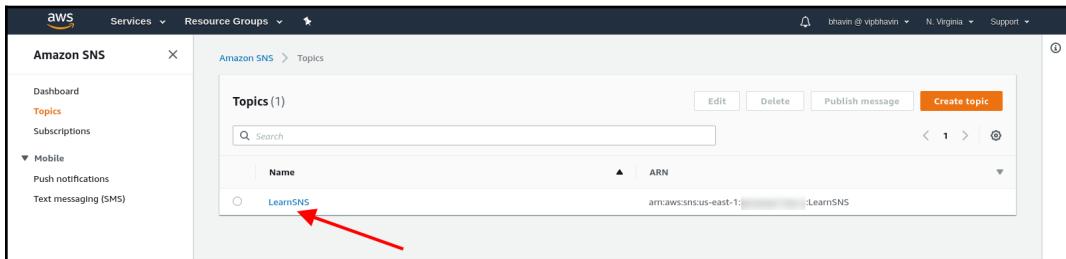


Figure 13.29: SNS topics dashboard

3. Click on **Create subscription**, as shown in *Figure 13.30*:

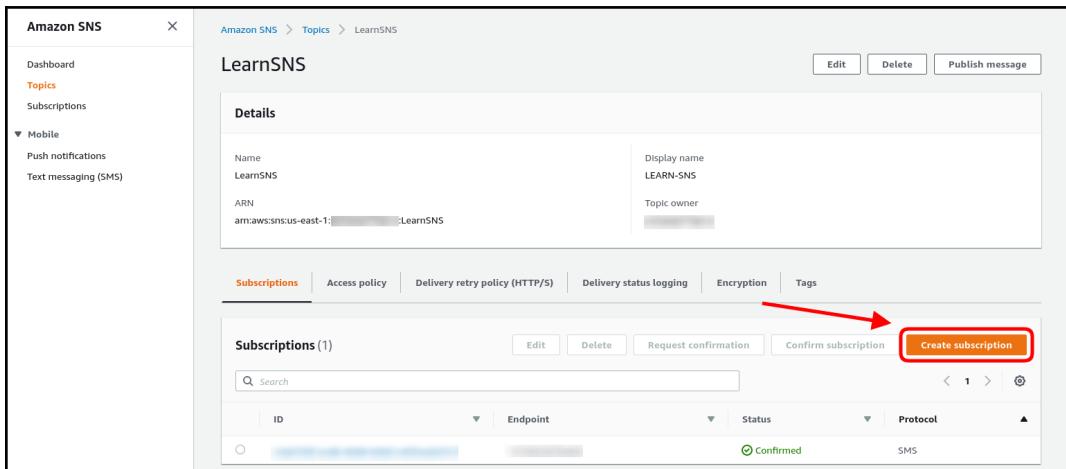


Figure 13.30: SNS topics dashboard

4. This will come up with a popup, as shown in *Figure 13.31*. Select the **Protocol** as **AWS Lambda** and provide a Lambda function endpoint. It also allows you to customize whether to trigger a specific alias or a version of an alias. Finally, click on **Create subscription**:

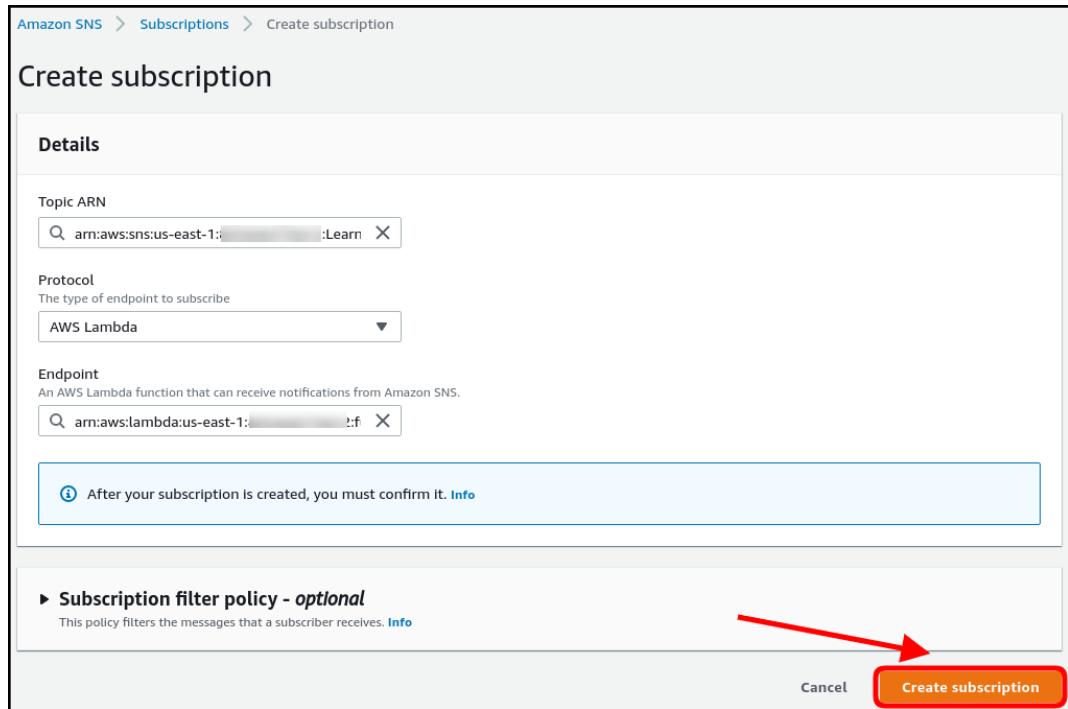


Figure 13.31: Create subscription

Sending Amazon SNS messages to Amazon SQS queues

The Amazon SNS topic's publisher can send a notification to an Amazon SQS queue. It is essential that the SQS queue is subscribed to a topic. A prerequisite is to have an SNS topic and an SQS queue.

Consider the following steps:

1. Go to the SQS dashboard, as shown in *Figure 13.32*:

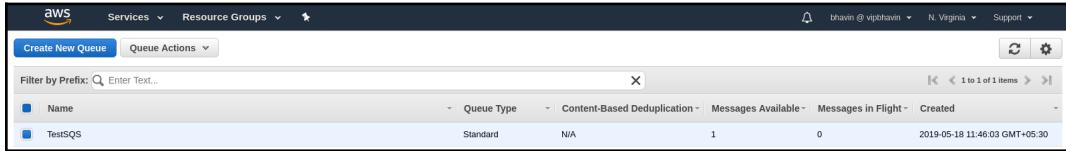


Figure 13.32: Amazon SQS dashboard

2. Select the queue and click on the **Queue Actions** drop-down menu. Select **Subscribe Queue to SNS Topic**, as shown in *Figure 13.33*:

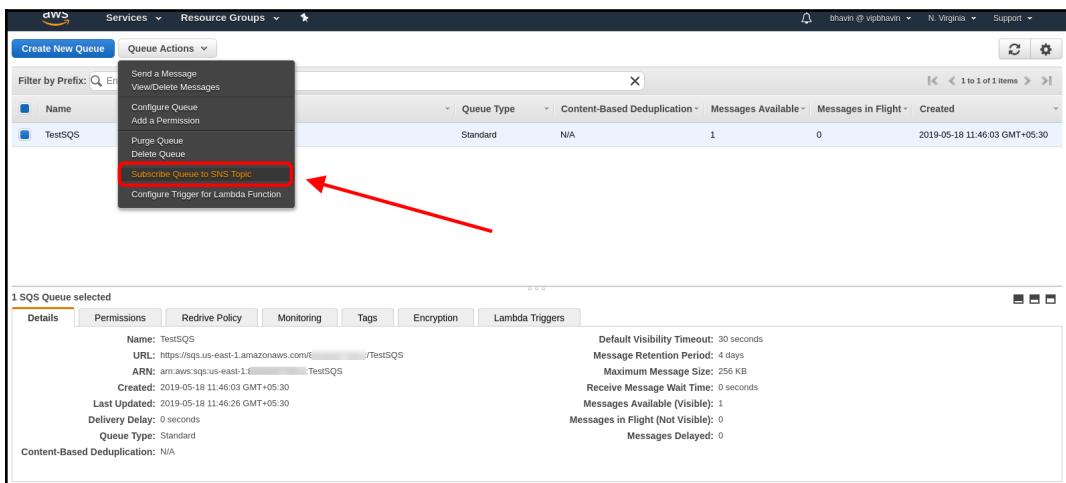


Figure 13.33: Subscribe queue to SNS topic

3. In the popup, select the appropriate AWS region where the SNS topic is created, choose the appropriate SNS topic to subscribe to the selected SQS queue, and click on **Subscribe**, as shown in *Figure 13.34*:

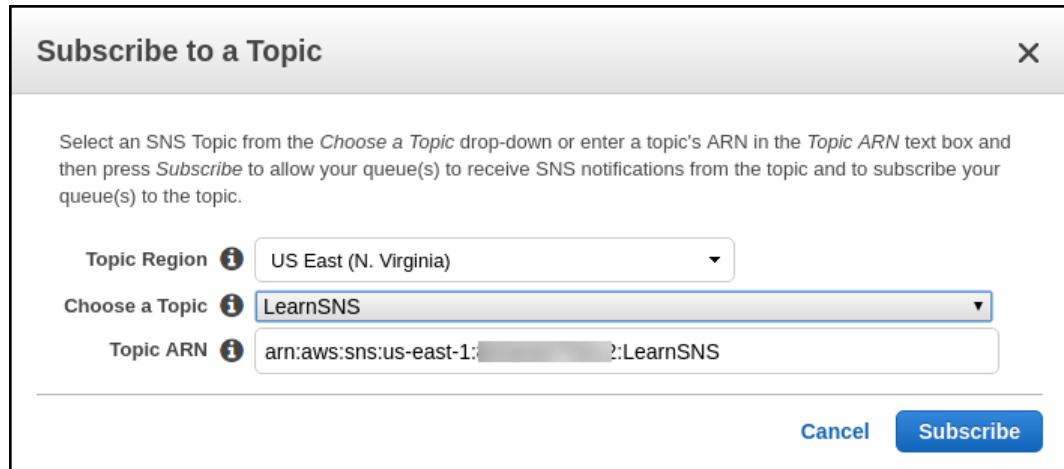


Figure 13.34: Subscribe to a topic

In the preceding popup, typing an ARN helps an SNS topic in another AWS account. Upon successful subscription of an SQS queue to an SNS topic, a popup will appear, as shown in *Figure 13.35*:

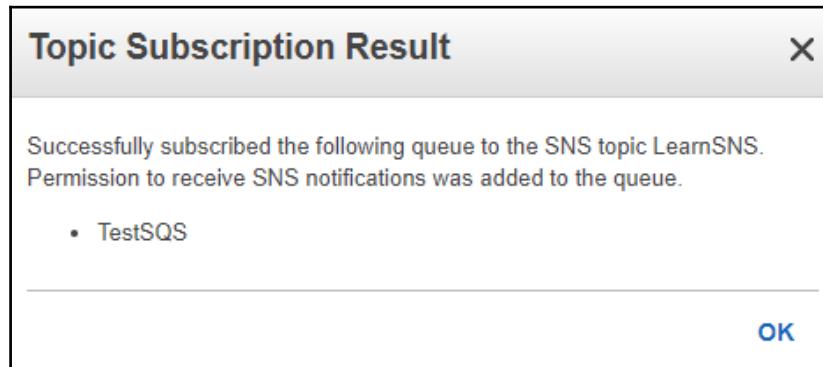


Figure 13.35: Subscription result

4. To verify whether the SQS queue has been successfully subscribed to a desired SNS topic, select the SNS topic from **SNS dashboard | Topics** and click on **ARN** to see the details. It is visible that the SQS topic has been subscribed, as shown in *Figure 13.36*:

The screenshot shows the 'Amazon SNS' interface with the 'Topics' section selected. A topic named 'LearnSNS' is displayed. The 'Details' section shows the topic's name ('LearnSNS') and display name ('LEARN-SNS'). The 'ARN' field contains the value 'arn:aws:sns:us-east-1:i...:LearnSNS'. The 'Subscriptions' tab is active, showing a table with two entries:

ID	Endpoint	Status	Protocol
[REDACTED]	[REDACTED]	Confirmed	SMS
[REDACTED]	arn:aws:sqs:us-east-1:[REDACTED]:TestSQS	Confirmed	SQS

Figure 13.36: SNS topic dashboard

5. Now, when the publisher publishes a message, the SQS queue will get a notification. Visible messages at the SQS queue will be increased. To see a visible message, go to the SQS dashboard, select the queue, and the available messages will be seen in the lower pane, as shown in *Figure 13.37*:

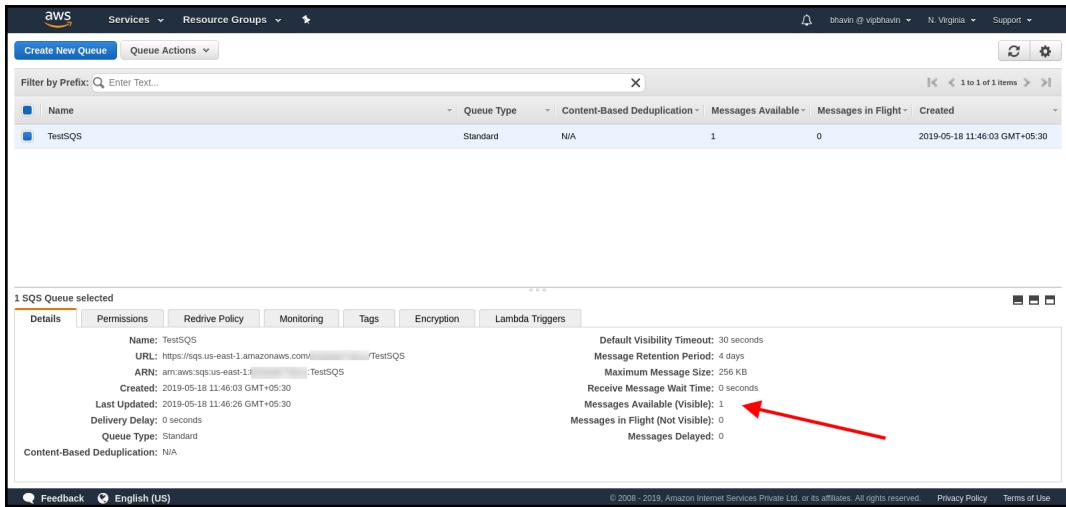


Figure 13.37: SQS dashboard

The Amazon SQS queue will receive messages in a JSON format. Here is a sample message:

```
{
  "Type" : "Notification",
  "MessageId" : "153697k2i-5672-9k8g-6fp8-159hn86d4h97",
  "TopicArn" : "arn:aws:sns:us-west-2:123456789012:TestTopic",
  "Subject" : "Testing message published to a subscribed queue",
  "Message" : "Hello world!",
  "Timestamp" : "2019-05-02T05:08:40.901Z",
  "SignatureVersion" : "1", "Signature" :
  "EXAMPLEd4f5h7a3f5VO0FFbh75fk197JLRfySEoWz4uZHSj6ycK4ph71Zmdv0
  NtJ4dC/E19FOG
  p3VuvchpaTraNHWhhq/OsN1HVz20zxmF9b88R878hqqfKB5woZZmz87HiM6CYD
  To3l7LMwFT4VU 7ELtyaBBafhPTg905GhsKkg=",
  "SigningCertURL" :
  "https://sns.us-west-2.amazonaws.com/SimpleNotificationService
  -f3ecfb7224c7 233fe7bb5f59f96de52f.pem",
  "UnsubscribeURL" :
  "https://sns.us-west-2.amazonaws.com/?Action=Unsubscribe&Subsc
  riptionArn=ar n:aws:sns:us-
  west-2:123456789012:MyTopic:c7fe3a54-ab0e-4ec2-88e0-
  db410a0f2bee"
}
```

Monitoring SNS with CloudWatch

Amazon SNS and Amazon CloudWatch are both integrated. Every SNS topic publishes standard metrics and dimensions in a CloudWatch. The CloudWatch metrics for each SNS topic can be viewed by performing the following steps:

1. Go to the CloudWatch dashboard with the IAM user who has sufficient privileges. Click on **Metrics**, as shown in *Figure 13.38*:

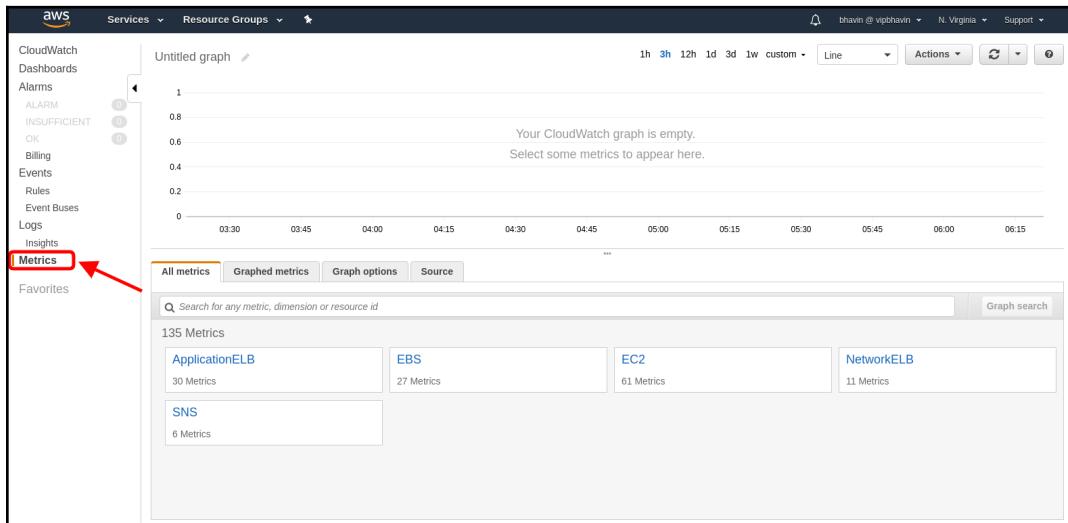


Figure 13.38: CloudWatch metrics dashboard

2. Select SNS to explore all the metrics, as shown in *Figure 13.39*:

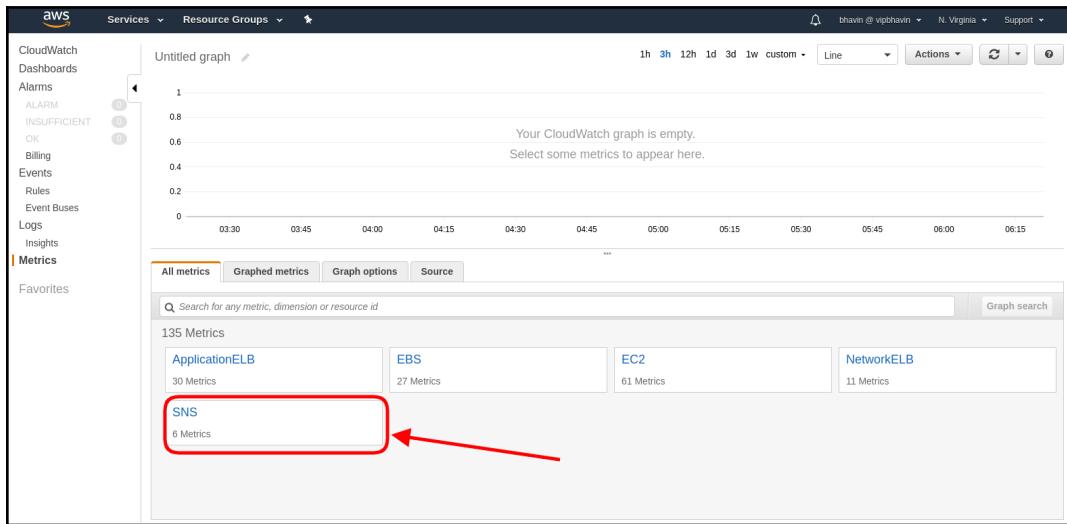


Figure 13.39: CloudWatch metrics dashboard

3. Click on **Topic Metrics** to explore the SNS topic metrics, as shown in *Figure 13.40*:

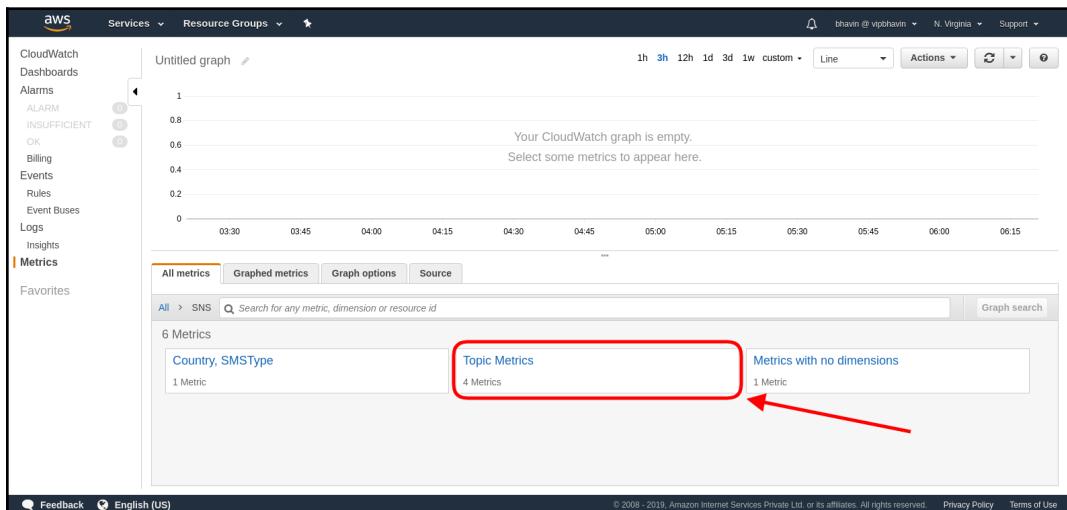


Figure 13.40: CloudWatch metrics dashboard

4. Select the appropriate metric from the lower pane to see the activity graph on the upper pane, as shown in *Figure 13.41*:

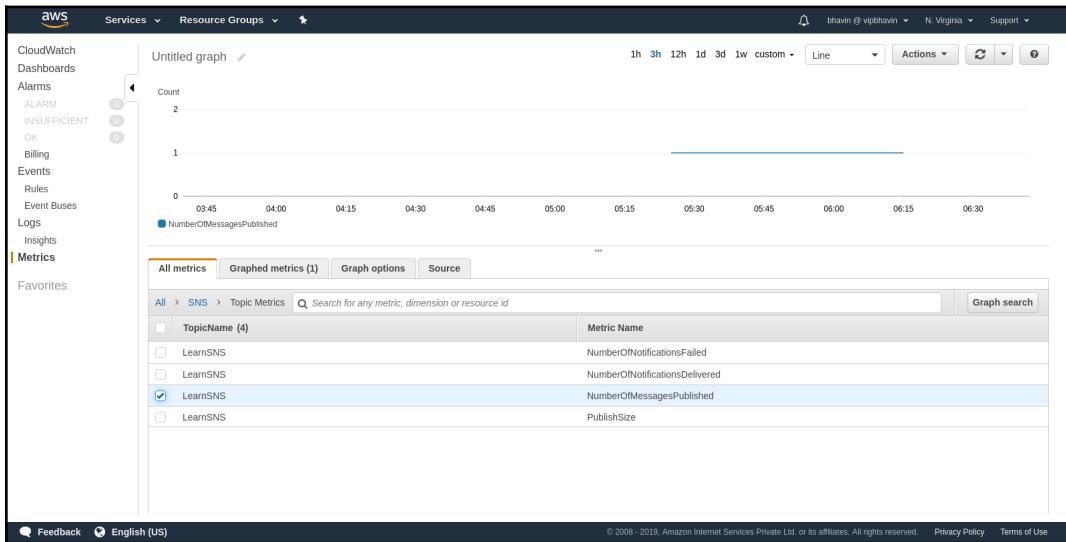


Figure 13.41: CloudWatch metrics dashboard

It is a best practice to configure alarms on critical metrics, such as `NumberOfNotificationsFailed`.

SNS best practices

Some SNS best practices are as follows:

- It is recommended to configure alerts on various SNS standard metrics to observe performance parameters, such as success rates, failure rates, and deliveries to SQS.
- Configure an access policy to control who can publish a message to, and receive a notification from, an SNS topic.
- In order to delete an SNS topic, ensure that all the subscriptions for that topic are deleted first.
- Use Amazon SNS and SQS services to build loosely-coupled applications or a serverless architecture.

Summary

- AWS SNS works based on push technology.
- An SNS topic acts as an access point between the publisher and subscriber applications.
- When several Amazon SQSs act as a subscriber, a publisher sends a message to an SNS topic and it distributes this topic to many SQS queues in parallel. This concept is called fanout.
- SNS can send mobile push notifications directly to mobile applications.
- SNS can push email and text messages to subscriber applications or users.
- Each SNS topic can have multiple subscribers.
- Each subscriber may use the same, or different, protocols.
- Subscribers can receive a notification over a desired protocol as and when the publisher sends any message to the same topic.
- SNS provides various protocols for communication (HTTP, HTTPS, email, email-JSON, Amazon SQS, application, AWS Lambda, or SMS).
- As soon as the publisher sends a message to an SNS topic, it tries to deliver a notification/message to all the subscribers.
- SNS provides a topic policy, which can be used to control who can subscribe to or publish to a topic. You can use an access control policy when you want to allow an IAM user to publish a message to one or more SNS topics.
- Whenever any request to access an AWS resource is initiated, policy evaluation logic evaluates the related policies to determine whether to allow or deny an incoming request.
- An AWS Lambda function can be invoked with Amazon SNS notifications.
- The Amazon SNS topic's publisher can send a notification to an Amazon SQS queue.
- Amazon SNS and Amazon CloudWatch are both integrated. Every SNS topic publishes standard metrics and dimensions in a CloudWatch.

14

AWS Simple Workflow Service (SWF)

Amazon's **Simple Workflow Service (SWF)** is a workflow-management service that helps build applications that can handle work through distributed components. Using SWF, you can define a number of tasks that can be executed in a predefined sequence. You can build scalable, resilient, and truly distributed applications using Amazon SWF. You can schedule tasks and define dependencies and concurrency depending on the logical workflow of the application. This chapter introduces you to workflows, workflow history, actors, tasks, domains, object identifiers, task lists, workflow execution closure, life cycle, polling for tasks, execution, access key, and secret key, SWF endpoints, and managing access with **Identity and Access Management (IAM)**.

The purpose of this chapter is to introduce the reader to the basic concepts of SWF, with respect to the scope of the AWS Certified Developer – Associate exam. From a development perspective, SWF is a wide topic and a whole book could be written on it. Considering the scope of the exam, this chapter does not intend to teach the reader how to code SWF applications, but focuses more on the fundamental aspects of SWF.

The following topics will be covered in this chapter:

- When to use Amazon SWF
- SWF endpoints
- Managing access with IAM

When to use Amazon SWF

Here are some scenarios where SWF can be used:

- When you have multiple tasks that need to be coordinated and executed in a specific sequence based on some dependency or in parallel
- When you have multiple application components and need to dispatch tasks to these application components
- When you have a distributed application and you need to coordinate and process tasks in a distributed application environment
- When you need to execute ordered application steps
- When you need to manage the application state during distributed execution
- When you need to reliably execute periodic tasks and audit the execution
- When you need to asynchronously execute event-driven tasks

Here are some SWF use cases:

- Media processing
- Customer-order-processing workflow
- Web application backend
- Business-process workflow
- Analytics pipelines

Now that we know what SWF is and what it can do, let's look at some basic concepts of SWF, such as workflows, workflow history, actors, tasks, domains, object identifiers, task lists, workflow execution closure, execution life cycle, and polling for tasks.

Workflow

A **workflow** is a mechanism to execute a number of distributed application tasks in an asynchronous way. With workflow, you can manage multiple activities asynchronously using more than one computing resource. The execution of the workflow tasks can be sequential and parallel as needed. While creating a workflow, you need to determine the tasks to be executed in the workflow. SWF recognizes these tasks as activities. You can define the coordination logic in the workflow that determines the order in which the activities are executed.

Example workflow

As shown in the following diagram, a customer-order-processing workflow can be implemented using SWF:

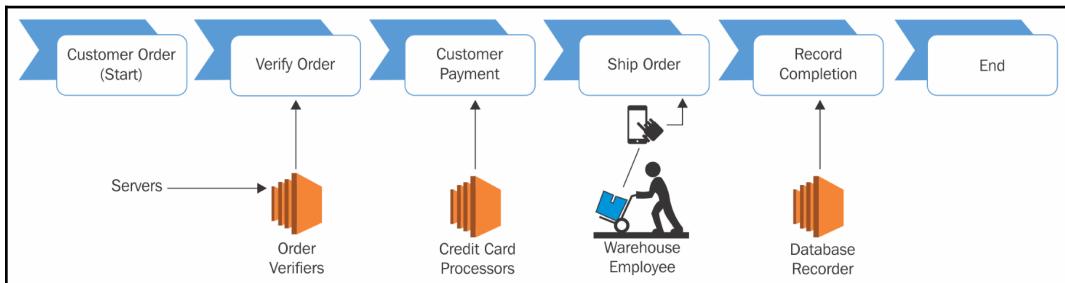


Figure 14.1: SWF customer-order-processing workflow

The steps for SWF customer-order-processing workflow are as follows:

1. The customer places an order.
2. The order is verified by the **Order Verifiers** component of the application, which is hosted on a separate environment.
3. The customer is charged by the **Credit Card Processors** component of the application.
4. The **Warehouse Employee** processes the order and ships it.
5. The order detail is updated in the database by the **Database Recorder**.
6. The workflow ends.

Workflow history

SWF keeps the history or execution progress of any workflow in the workflow history. Once the execution of a workflow starts, SWF keeps a detailed history of each and every step of the workflow. Whenever the workflow execution state changes, such as when a new activity is scheduled in the workflow or an activity is completed, it is represented as an event in the workflow history. It records events that change the state of the workflow, such as when an activity is scheduled or completed or a task execution times out. It does not record any event that does not change the state of the workflow.

How workflow history helps

Workflow history can be helpful in a number of ways. The following list describes some of the ways in which workflow history can be helpful:

- It stores all the details about the workflow execution and thus eliminates the need for the application to maintain the state.
- It provides the current status of each of the activities scheduled along with its results. SWF executes the next steps based on this information.
- It provides an audit trail, which can be used to monitor and verify the workflow execution:

```
Order0001
  Start Workflow Execution
    Schedule Verify Order
      Start Verify Order Activity
      Complete Verify Order Activity
    Schedule Customer Payment
      Start Charge Customer Payment Activity
      Complete Customer Payment Activity
    Schedule Ship Order
      Start Ship Order Activity
```

Actors

In simple terms, an **actor** is a program or an entity that performs different types of activities in a workflow. An actor can be any of the following:

- Workflow starter
- Decider
- Activity worker

Actors can interact with SWF using APIs. Actors can be developed in any programming language. The next diagram shows the SWF architecture along with its actors:

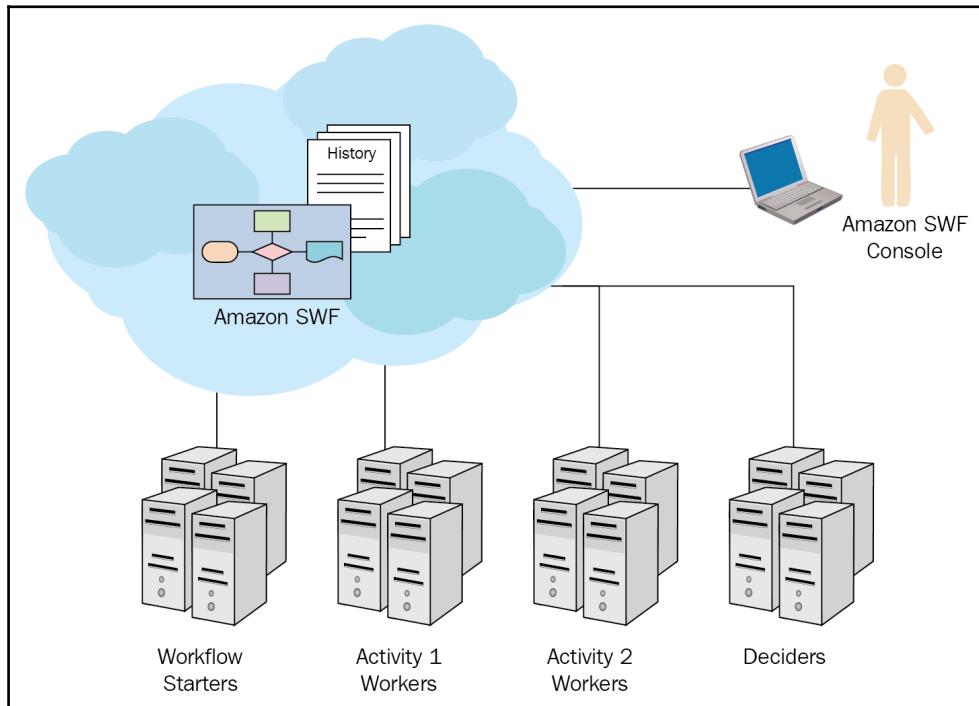


Figure 14.2: Amazon SWF architecture

Workflow starter

A **workflow starter** is a program or an application that starts the execution of a workflow. In the customer-order-processing example, the workflow starter can be a shopping site where the customer orders an item. It can also be a mobile application from which the customer places an order.

Decider

A **decider** is a program or an application that decides the coordination logic of a workflow. It decides on the order of execution, concurrency, and scheduling of the tasks as per the application logic. Whenever there is any change in the workflow execution, for example, an activity is completing, the underlined client polls for the tasks for making decisions and it passes them to the programmatic entity, called the **decider**. The decider receives the decision tasks along with the workflow history. The job of the decider is to analyze the execution history and decide which step should be executed next. Once the decider takes the decision, it communicates this decision back to SWF. Interactions between workers and the decider are facilitated by Amazon SWF. It provides uniform views on the advancement of the tasks and allows initiating new tasks in a continuous manner. The tasks are stored by SWF and assigned to workers as and when they are ready. SWF monitors the tasks and ensures that a task is assigned only once and is never duplicated.



Remember, a decision is a data type in SWF, and represents the next actions.

Activity worker

An **activity worker** is a program or an application that receives tasks from SWF, executes the tasks, and returns the result to SWF. Activity tasks are tasks that are identified by you in your application.

For using an activity task, you need to register the activity task in an SWF console or programmatically by using the `RegisterActivityType` action.

All the activity workers are registered in SWF polls for new activity tasks. SWF assigns activity tasks to a worker. There may be some tasks that can be performed by specific activity workers only. Once the activity worker receives a task from SWF, it starts the execution of the task and reports it to SWF on completion along with the results.

Subsequently, it polls the SWF for the next task. This entire process of polling for a task and executing it goes on until the entire workflow execution is completed.

Tasks

The work assignments that SWF provides to activity workers and deciders are called **tasks**. There are basically three types of tasks in SWF—an **activity task**, a **Lambda task**, and a **decision task**. Let's look at these task types:

- **Activity task:** An activity task describes the actions to be performed by an activity worker. The action depends upon the function of the activity worker. For example, an activity worker may be asked to check the inventory for a specific product or it may be asked to initiate a credit card transaction. The task contains all the details that an activity worker requires to perform the actions.
- **Lambda task:** A Lambda task and an activity task are similar. As the name suggests, it executes a Lambda function instead of an SWF activity.
- **Decision task:** A decider determines the next activity in a workflow based on the decision task. It tells the decider that the workflow execution status has changed. The current workflow history is carried along with the decision task.

SWF can schedule a decision task as and when the workflow starts and the status of the workflow changes, that is, activity task scheduled, activity task completed, and so on.

SWF domains

Domains in SWF are a mechanism to scope SWF resources, such as workflows, activity types, and workflow execution. All the resources are scoped to a domain. Domains isolate one set of types, executions, and task lists from other ones within an AWS account. When you work with SWF, you need to first define a domain. All the other resources are defined within a domain. You can define multiple domains in SWF. Similarly, one domain can have multiple workflows; however, workflows defined in different domains cannot interact with one another.

While registering a domain in SWF, you need to define the workflow-history-retention period. SWF maintains the history of a workflow for the time specified in the workflow-history-retention period even after the execution of the workflow is completed.

Object identifiers

Object identifiers are a way of uniquely identifying SWF objects. The following list describes how different types of objects are identified in SWF:

- **Workflow type:** A registered workflow type is distinguished by its domain, workflow name, and workflow version. You can specify the workflow type in a call to `RegisterWorkflowType`.
- **Activity type:** A registered activity type is distinguished by its domain, activity name, and activity version. You can specify the activity types in the call to `RegisterActivityType`.
- **Decision tasks and activity tasks:** SWF uses a unique task token to identify decision tasks and activity tasks. It generates a task token and returns it with other task information when `PollForDecisionTask` or `PollForActivityTask` are called. Mostly, the token is used by the process that is assigned to the task, but the token can also be passed on to other processes. Subsequently, the process with the token can report completion or failure of the task.

Task lists

Task lists are a mechanism to organize different tasks related to a workflow. Task lists can be thought of as dynamic queues. When scheduling a task in SWF, you can specify a task list. The task list works in a similar way to a queue. When polling SWF for tasks, you can specify the task list from where the task can be fetched.

Task lists offer a way to route tasks to worker processes based on the requirement of your application workflow. You don't need to explicitly create a task list; it is automatically created when a task is scheduled, if the task list is not already there. SWF maintains a separate task list for activity tasks and decision tasks. A task belongs to only one task list; it is not shared among multiple task lists. Just like activities and workflows, task lists also have a restricted scope. The scope of a task list is restricted to a specific AWS region and a specific SWF domain.

Workflow-execution closure

When a **workflow execution** is started, it changes an open state. An open workflow execution can be closed as one of the following:

- Completed
- Canceled
- Failed
- Timed out

Open workflow execution can be closed by a decider process, by an administrator, or by SWF. As and when the activities of the workflow finish, the decider process identifies and marks the workflow execution as completed. The decider uses the `RespondDecisionTaskCompleted` action and forwards the `CompleteWorkflowExecution` decision to SWF. Similarly, a decider process can also close the workflow execution as cancelled or failed. The decider process uses the `RespondDecisionTaskCompleted` action and forwards the `CancelWorkflowExecution` decision to SWF. Whenever a task enters a state that is outside the purview of normal completion, a decider should fail that workflow execution. To fail the workflow execution, the decider uses the `RespondDecisionTaskCompleted` action and forwards the `FailWorkflowExecution` decision to SWF.

Workflow executions are continuously monitored by SWF to confirm that the workflow execution does not exceed the timeout limit specified by the user in the workflow settings. As and when a workflow exceeds timeout, SWF closes the workflow.

At times, certain workflows run for too long and the history grows too large. The decider may close the execution of this workflow and continue running it as a new workflow execution. For this scenario, a decider uses the `RespondDecisionTaskCompleted` action and forwards the `ContinueAsNewWorkflowExecution` decision to SWF.

Last, but not least, a user can terminate a workflow execution directly from the SWF console. You can also terminate the execution programmatically with the `TerminateWorkflowExecution` API.

When you initiate termination of a workflow through the console or API, it automatically forces closure of the running workflow execution based on the selected workflow from the console or given `domain`, `runId`, and `workflowId` in an API call.

SWF can terminate a workflow if it exceeds any service-defined limits. It can also terminate a child workflow if the parent workflow is terminated and the child policy associated with the workflow is defined to terminate the child workflows.

Life cycle of a workflow execution

SWF starts communicating with actors from the start to the completion of a workflow and allocates respective activities and decision tasks to these actors. The following diagram describes the life cycle of a customer-order-processing workflow:

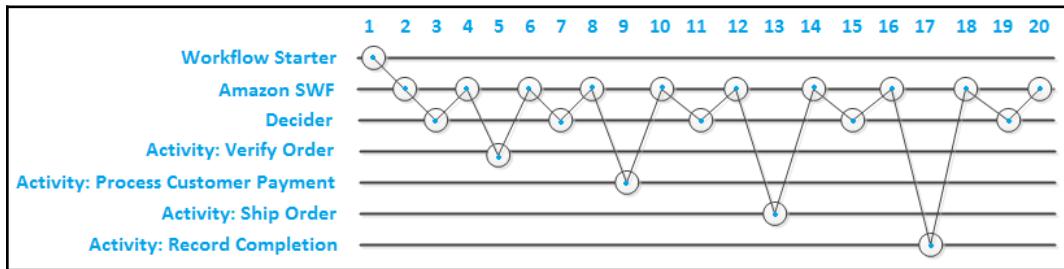


Figure 14.3: Workflow-execution life cycle

Here are the steps shown in the preceding diagram:

1. The workflow starter starts the workflow execution by calling the `StartWorkflowExecution` action with order information.
2. SWF gets the request to start the workflow execution, sends it back with a `WorkflowExecutionStarted` event, and schedules a decision task that raises a `DecisionTaskScheduled` event.
3. A process, configured as a decider in the workflow, polls for a decision task using the `PollForDecisionTask` action, and receives the decision task from SWF along with the task history. The decider uses coordination logic to ensure that the execution has not already run. After verification, it schedules the verify-order activity using the `ScheduleActivityTask` decision and returns the decision detail to SWF with the `RespondDecisionTaskCompleted` action.

4. SWF gets the decision from the decider, schedules the verify order activity task, and raises the `ActivityTaskScheduled` event. After scheduling the task, SWF waits for the task until it gets completed or times out.
5. An activity worker polls for the tasks using the `PollForActivityTask` action and receives the verify order activity task. After receiving the task, an activity worker performs their task and responds to SWF with the result using the `RespondActivityTaskCompleted` action.
6. SWF receives the result of the verify-order activity shared by the activity worker and raises the `ActivityTaskCompleted` event. It adds the result to the workflow history. At the end of the step, it schedules a decision task and raises the `DecisionTaskScheduled` event.
7. The decider polls for the decision task using the `PollForDecisionTask` action and receives the decision task from SWF, along with the task history. The decider uses coordination logic to ensure that the execution has not already run. After verification, it schedules a process customer payment activity using the `ScheduleActivityTask` decision and returns the decision detail to SWF with the `RespondDecisionTaskCompleted` action.
8. SWF gets the decision from the decider through the `DecisionTaskCompleted` event, schedules the process customer payment activity task, and raises the `ActivityTaskScheduled` event. After scheduling the task, SWF waits for the task until it gets completed or times out.
9. An activity worker can perform process customer payment polls for the task using the `PollForActivityTask` action and receives the process customer payment activity task. After receiving the task, the activity worker performs the task and responds to SWF with the result using the `RespondActivityTaskCompleted` action.
10. SWF receives the result of process customer payment activity shared by the activity worker and raises the `ActivityTaskCompleted` event. It adds the result to the workflow history. At the end of the step, it schedules a decision task and raises the `DecisionTaskScheduled` event.
11. The decider polls for the decision task using the `PollForDecisionTask` action and receives the decision task from SWF along with the task history. The decider then uses the coordination logic to ensure that the execution has not already run. After verification, it schedules the shipping of the order activity using the `ScheduleActivityTask` decision and returns the decision detail to SWF with the `RespondDecisionTaskCompleted` action.

12. SWF gets the decision from the decider through the `DecisionTaskCompleted` event, schedules the ship order activity task, and raises the `ActivityTaskScheduled` event. After scheduling the task, SWF waits for the task until it completes or times out and raises a timeout event.
13. An activity worker can perform the ship order polls for the task using the `PollForActivityTask` action and receives the activity task. After receiving the task, the activity worker performs the task and responds to SWF with the result using the `RespondActivityTaskCompleted` action.
14. SWF receives the result of the ship order activity shared by the activity worker through the `ActivityTaskCompleted` event. It adds the result to the workflow history. At the end of the step, it schedules a decision task and raises a `DecisionTaskScheduled` event.
15. The decider polls for the decision task using the `PollForDecisionTask` action and receives the decision task from SWF along with the task history. The decider uses coordination logic to ensure that the execution has not already run. After verification, it schedules a record completion activity using the `ScheduleActivityTask` decision and returns the decision detail to SWF using the `RespondDecisionTaskCompleted` action.
16. SWF gets the decision from the decider through the `DecisionTaskCompleted` event, schedules the record completion activity task, and raises the `ActivityTaskScheduled` event. After scheduling the task, SWF waits for the task until it is completed or it times out and raises a timeout event.
17. An activity worker that can perform the record-completion task polls for the task using the `PollForActivityTask` action and receives the activity task. After receiving the task, the activity worker performs the task and responds to SWF with the result using the `RespondActivityTaskCompleted` action.
18. SWF receives the result of the record completion activity shared by the activity worker through the `ActivityTaskCompleted` event. It adds the result to the workflow history. At the end of the step, it schedules a decision task and raises the `DecisionTaskScheduled` event.

19. The decider polls for the decision task using the `PollForDecisionTask` action and receives the decision task from SWF along with the task history. The decider uses coordination logic and decides to close the workflow execution. It returns the `CompleteWorkflowExecution` decision to SWF with the `RespondDecisionTaskCompleted` action along with any result.
20. SWF closes the workflow execution, archiving the history for any future reference, and raises the `WorkflowExecutionCompleted` event.

Polling for tasks

Deciders and activity workers interact with SWF using **long polling**. They regularly send messages to SWF indicating that they are ready to receive a task from a predefined task list.

If there is a task already available to assign, SWF responds with the task immediately. If the task is not available, SWF keeps the TCP connection alive for up to 60 seconds. If a task becomes available in these 60 seconds, it responds with the task. If there is no task available within these 60 seconds, SWF responds with an empty response and the connection is closed. In cases where the decider or activity worker receives an empty response, they should poll for the task again.

Long polling is suitable when there is a high volume of tasks available for processing. It is recommended that you keep deciders and activity workers behind a firewall.

SWF endpoints

Amazon provides **SWF endpoints** in multiple regions. These endpoints are provided to reduce latency while accessing the service and storing or retrieving data from AWS. SWF endpoints are independent of one another. Your SWF domains, workflows, and activities registered in a region are isolated from the other regions and they do not share data or attributes with one another. For example, you can register a domain named `SWF-Mydomain-1` in multiple regions. Even though the domain name remains the same, they are distinct domains specific to respective regions. A domain registered in `us-east-1` cannot share any data or attributes with a domain registered in `us-west-1`.

SWF endpoints available in different AWS regions are shown in the table found at http://docs.aws.amazon.com/general/latest/gr/rande.html#swf_region.

Managing access with IAM

You can manage controlled access to SWF resources using IAM. Using IAM, you can create users in your AWS account and provide them with respective permissions. Each IAM user has a separate set of IAM keys. These IAM keys provide users with access to respective resources on AWS. An IAM policy can be attached to a user that controls what resources a user can access. Using IAM policies, you can control access at a granular level, such as allowing or denying access to a specific set of SWF domains.

SWF uses the following principles for access control:

- Access to various SWF resources is controlled only on the basis of IAM policies.
- IAM uses the denying-by-default policy, which means if you do not explicitly allow any access, then, by default, access is denied.
- You need to attach IAM policies to the actors of the workflow to control access to the SWF resources.
- You can specify resource permissions only for domains.
- You can use conditions in the permission to further restrict the permission in a policy.

We will have a look at some of the examples of the IAM policy.

SWF – IAM policy examples

The following is a simple policy that allows all SWF actions on all the domains in the account:

```
{  
    "Version": "2012-10-17",  
    "Statement" : [ {  
        "Effect" : "Allow",  
        "Action" : "swf:*",  
        "Resource" : "arn:aws:swf:*:123456789012:/domain/*"  
    } ]  
}
```

The following policy allows all SWF actions, but restricts access to a specific domain in the account:

```
{  
    "Version": "2012-10-17",  
    "Statement": [ {  
        "Effect" : "Allow",  
        "Action" : "swf:*",  
        "Resource" : "arn:aws:swf:*:123456789012:/domain/mydomain1"  
    } ]  
}
```

The following policy allows all SWF actions in two specific domains: mydomain1 and mydomain2:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect" : "Allow",  
            "Action" : "swf:*",  
            "Resource" : "arn:aws:swf:*:123456789012:/domain/mydomain1"  
        }, {  
            "Effect" : "Allow",  
            "Action" : "swf:*",  
            "Resource" : "arn:aws:swf:*:123456789012:/domain/mydomain2"  
        }  
    ]  
}
```

The following policy allows access to the `StartWorkflowExecution` action on the mydomain1 domain, and specifically to version1 of myworkflow1:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect" : "Allow",  
            "Action" : "swf:StartWorkflowExecution",  
            "Resource" : "arn:aws:swf:*:123456789012:/domain/mydomain1",  
            "Condition": {  
                "StringEquals" : {  
                    "swf:workflowType.name" : "myworkflow1",  
                    "swf:workflowType.version" : "version1"  
                }  
            }  
        }]  
}
```

Summary

- Amazon's SWF is a workflow-management service that helps build applications that can handle work through distributed components.
- A workflow is a mechanism to execute a number of distributed application tasks in an asynchronous way.
- SWF keeps the history or execution progress of any workflow in the workflow history.
- In SWF, an actor is a program or an entity that performs different types of activities in a workflow. An actor can be workflow starter, decider, or activity worker.
- A workflow starter is a program or an application that starts the execution of a workflow.
- A decider is a program or an application that decides the coordination logic of a workflow.
- An activity worker is a program or an application that receives tasks from SWF, executes the tasks, and returns the result to SWF.
- The work assignments that SWF provides to activity workers and deciders are called tasks.
- There are basically three types of task in SWF: activity task, Lambda task, and decision task.
- An activity task describes the actions to be performed by an activity worker.
- A Lambda task executes a Lambda function instead of an SWF activity.
- A decider determines the next activity in a workflow based on the decision task.
- Domains in SWF are a mechanism to scope SWF resources, such as workflows, activity types, and workflow execution.
- Object identifiers are a way of uniquely identifying SWF objects.
- A registered workflow type is distinguished by its domain, workflow name, and workflow version.
- A registered activity type is distinguished by its domain, activity name, and activity version.
- SWF uses a unique task token to identify decision tasks and activity tasks.

- Task lists are a mechanism for organizing different tasks related to a workflow.
- An open workflow execution can be closed as completed, canceled, failed, and timed out.
- SWF starts communicating with actors from the start to the completion of a workflow and allocates respective activities and decision tasks to these actors.
- Amazon provides SWF endpoints in multiple regions. These endpoints are provided to reduce latency while accessing the service and storing or retrieving data from AWS.

15

CloudFormation Overview

AWS infrastructure can be created and customized using the AWS dashboard (GUI), CLI, or API. These methods may be able to build an infrastructure quickly as a one-off; however, over a long period of time, if used to create a whole or partial infrastructure repeatedly in a different region to build **Disaster Recovery (DR)**, or in a subsidiary AWS account, then those methods would be costly, not only in terms of time and money, but also in terms of management, modification, and maintenance. It is a case of reinventing the wheel every time and it is error-prone. To resolve this issue, Amazon provides the CloudFormation service.

AWS CloudFormation allows you to create and customize the AWS infrastructure using code. It also enables you to create your infrastructure as code. This program or code is known as a **template** in AWS CloudFormation. These templates are also referred to as **CloudFormation Templates (CFTs)**. For fulfilling various tasks, you may write one or more CFTs. Each CFT can be written in one of the supported scripting languages (such as JSON or YAML). You can use these CFTs to recreate the same infrastructure in a different region or in a different AWS account. With little or no changes in the template using runtime parameters, the infrastructure gets ready in different regions or in different AWS accounts.

AWS does not charge you for using the CloudFormation service. You only pay for the chargeable resources that you create when using it. For example, you can create a web application infrastructure using CFT that includes a VPC, a few EC2 instances, and a few RDS instances. In this case, the AWS account incurs charges only for the EC2 instances and RDS instances based on its configuration, such as instance size and attached EBS volumes, as these are chargeable services. However, for custom VPC and CloudFormation services, there are no charges applied, as both of these services are free.

The following topics will be covered in this chapter:

- Understanding templates
- Understanding a stack
- Template structure
- A sample CloudFormation template
- CloudFormer
- Rolling updates for auto scaling groups
- CloudFormation best practices



If the AWS account is eligible for the free tier, the advantage will be leveraged in the monthly AWS billing, that is, if AWS resources are created using templates or by any other possible method.

Understanding templates

AWS CFT describes all AWS resources and their properties in JSON or YAML format. Templates can be written using any text editor. It is recommended that you give relevant and meaningful filenames to each template. Template extensions can be .json, .yaml, or .txt. When these templates are executed, the defined AWS resources are created in the respective AWS account. You can either upload the template to an S3 bucket and specify the template URL or you can upload the template file using the browse button in the template creation wizard. Even if you upload the template file using the browse button in the template creation wizard, it is internally stored in S3. The following diagram helps us to understand this:

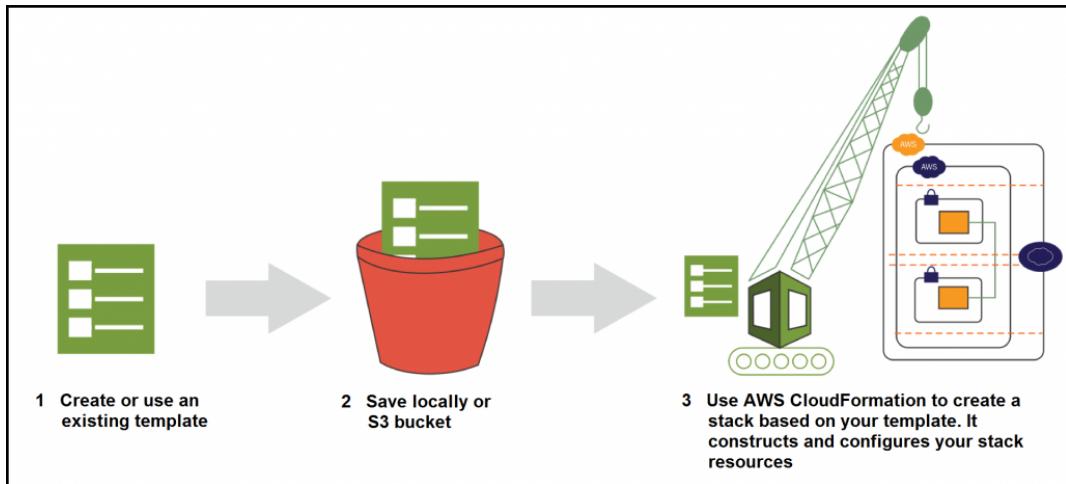


Figure 15.1: The AWS CloudFormation flow



While creating a stack, if the template path is pointing to the local machine, then it will automatically upload the CloudFormation template to the AWS S3 bucket in the relevant region. In each region, AWS CloudFormation will create its own bucket.

While writing a template, it is not necessary for you to identify AWS resource dependencies. CloudFormation automatically identifies the resource dependencies and creates them sequentially. For example, when a template is written to create a custom VPC and EC2 instance, it first creates a custom VPC. Then, an EC2 instance is created in the same VPC only after the VPC is available. Additionally, templates can be used to create simple or complex AWS infrastructures.

Generally, it is recommended that you write a template for each layer of the architecture; for example, separate templates for networking components, database servers, and web servers. As a result, the required downtime during maintenance and its impact on the business can be minimized. In a single template, multiple AWS resources can be specified. As the enterprise requirement changes with time, these templates can be modified accordingly. These modified templates can be stored in a version control repository such as Git to maintain the history of the CFTs.

AWS CFT creation is not just restricted to a text editor using JSON or YAML code. In fact, it can also be created using a GUI tool, such as AWS CloudFormation Designer. To design your own templates with AWS CloudFormation Designer, you can refer to <https://console.aws.amazon.com/cloudformation/designer>. For more details on CloudFormation Designer, you can refer to <http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/working-with-templates-cfn-designer.html>.



To use AWS CloudFormation Designer, you may need to log in to your AWS account.

Understanding a stack

A stack is created upon the successful execution of a template in CloudFormation. Executing a template creates a defined set of AWS resources. A group of these AWS resources defined in CloudFormation is called a **stack**. During template execution, if CloudFormation is unable to create any resource, the whole stack creation fails. When a CloudFormation execution fails, it rolls back all of the execution steps and deletes any resources created during the process. CloudFormation execution may fail due to several reasons, including insufficient privileges. Due to limited IAM privileges, if the rollback process is unable to delete the created resources, then the incomplete stack remains in the AWS account until it is deleted by an IAM user with sufficient privileges to delete the stack.



At the time of creating a stack from the template, AWS CloudFormation only checks for syntax error in JSON/YAML notation. It does not check whether the IAM user executing the template has sufficient privileges to complete the template execution or not. Additionally, it does not check whether any resource creation may violate AWS soft limits for the resources in the account.

Stack helps to efficiently manage several AWS resources as a single unit. The property of each resource created inside a stack can also be modified manually, but it is a best practice to modify stack resource properties by modifying the CFTs only. With the help of the update stack option, modifications can be carried out in an existing stack.

An existing CloudFormation stack can be updated by submitting a modified version of the original stack template, or by giving different input parameter values during the execution. During the re-execution of the template, AWS CloudFormation compares the updated template with the original template and creates a change set. The change set includes the changes required to update the stack. You can review the proposed changes and execute it to update the stack or you can opt to create a new change set. *Figure 15.2* summarizes the workflow for updating a stack:

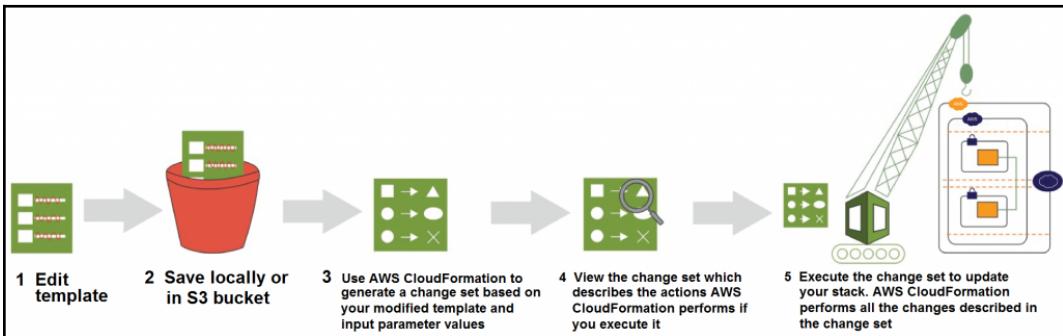


Figure 15.2: The workflow for updating an AWS CloudFormation stack

AWS CloudFormation stacks can be easily deleted. When the stack is deleted, all of the AWS resources created during stack creation are also deleted. While deleting a stack, there may be a situation when partial AWS resources are required to be retained for future use. With the help of `DeletionPolicy`, these resources can be retained. For example, while deleting a stack, you may want to delete an EC2 instance but retain the EBS volume attached to the instance. You can control this behavior using `DeletionPolicy`.

`DeletionPolicy` can have the three following attributes:

- **Delete:** When you specify this attribute in `DeletionPolicy`, CloudFormation deletes the associated resource on stack deletion.
- **Retain:** When you specify this attribute in `DeletionPolicy`, CloudFormation retains the associated resource even after the stack deletion.
- **Snapshot:** When you specify this attribute in `DeletionPolicy`, CloudFormation creates a snapshot of the associated resource before deleting the stack.

The following is an example snippet of how you can specify the `DeletionPolicy` attribute in a CFT. In this example, we are creating a **General Purpose SSD (gp2)** volume of size 200 GB and configuring the `DeletionPolicy` attribute with the `Retain` value . As we know, when we define the `DeletionPolicy` attribute as `Retain` for a resource, the resource is retained even after the CloudFormation stack is deleted:

```
NewEBSVolume" : {
  "Type" : "AWS::EC2::Volume",
  "Properties" : {
    "Size" : "200",
    "VolumeType" : "gp2",
    "Encrypted" : "false",
    "AvailabilityZone" : { "Fn::GetAtt" : [ "Ec2Instance",
      "AvailabilityZone"
    ] },
    "Tags" : [ {
      "Key" : "Name", "Value" : "DataVolume"
    } ]
  },
  "DeletionPolicy" : "Retain"
}
```

The template structure

The following code block helps us to understand the basic AWS CFT structure in JSON and YAML format. For a basic AWS CloudFormation template structure in JSON and YAML, you can refer to <https://docs.aws.amazon.com/AWScloudFormation/latest/UserGuide/template-anatomy.html>:

- **The JSON structure:** The JSON structure is as follows:

```
{
  "AWSTemplateFormatVersion" : "version date",
  "Description" : "JSON string",
  "Metadata" : {
    template metadata
  },
  "Parameters" : { set of parameters
  },
  "Mappings" : { set of mappings
  },
  "Conditions" : {
    set of conditions
  },
  "Transform" : { set of transforms
}
```

```
},
"Resources" : { set of resources
},
"Outputs" : {
set of outputs
}
}
```

- **The YAML structure:** The YAML structure is as follows:

```
---
AWSTemplateFormatVersion: "version date"
Description: String
Metadata: template metadata
Parameters:
set of parameters
Mappings:
set of mappings
Conditions:
set of conditions
Transform:
set of transforms
Resources:
set of resources
Outputs:
set of outputs
```



Having a basic understanding of any one of these two data interchange formats, JSON or YAML, is advantageous for writing a quick, efficient, and effective AWS CFT.

A CFT is divided into nine sections. Out of these nine sections, the Resources section is the only required section to successfully execute an AWS CFT. Each Resources section must have at least one resource definition to create with essential properties. For example, when creating an EC2 instance, the AMI ID and instance type are essential parameters. We will look at the usage of each of these CFT sections.

AWSTemplateFormatVersion

The AWSTemplateFormatVersion section is optional. It identifies the capabilities of the template. The latest and currently supported version is 2010-09-09, and it must be defined as a literal string (that is, enclosed in double quotes). In any case, if this section is not specified, then, by default, the latest template format version is assumed. This template version is different from the API or **Web Services Description Language (WSDL)** version:

- The example in JSON is as follows:

```
"AWSTemplateFormatVersion" : "2010-09-09"
```

- The example in YAML is as follows:

```
AWSTemplateFormatVersion: "2010-09-09"
```

Description

The Description section is optional in a template. It makes it possible to write meaningful comments between 0 and 1,024 bytes for CFTs. This will help other developers to understand the purpose of the template. For example, in the future, when changes need to be carried out in the existing infrastructure, it would be very important for architects and developers to understand the purpose of the template. The Description section should always be next to the AWSTemplateFormatVersion section:

- The example in JSON is as follows:

```
"Description" : "Provide meaningful description about the template."
```

- The example in YAML is as follows:

```
Description: > Provide meaningful description about the template.
```

Metadata

The `Metadata` section is an optional section in a template. It can be used to write additional details in the form of JSON or YAML objects. It supports metadata keys, which enable us to retrieve the defined configuration or settings in the `Resources` section. These keys are defined in the `Metadata` section. While writing this book, the CFT `Metadata` section supports the following three metadata keys:

```
AWS::CloudFormation::Init
```

This defines the configuration for the `cfn-init` helper scripts in the EC2 instance. These helper scripts are executed only once while creating a new EC2 instance to install or configure applications on those EC2 instances. Then, in the future, when these EC2 instances are restarted, it doesn't execute this script:

```
AWS::CloudFormation::Interface
```

This helps us to define the grouping and ordering for the input parameters. Input parameters help to accept values (that is, at template runtime) for resource properties. It helps us to make stack creation and modification dynamic:

```
AWS::CloudFormation::Designer
```

A designer metadata key is automatically added to the CFT when it is created using AWS CloudFormation Designer. It usually contains information about various AWS resources and how they were laid down on a GUI designer.

Parameters

The `Parameters` section is optional in a template. It can be used to pass values into the template to be customized while creating a stack. It will allow us to create a stack each time with different values at runtime without changing any code in the CFT. Suppose that a template has been written to create a three-tier architecture and, based on the infrastructure environment (such as test, development, preproduction, or production), the user may want to change the EC2 or RDS instance type. With the help of the `Parameters` section, these values can be customized at runtime; stack creation will ask for a parameter to input values for these resource properties.

Optionally, it is also possible to provide default values for each parameter. So, when the user does not provide any input, it will take that default value. There is also an option to place validation in a parameter to allow the input of only relevant information. There is no need to write separate CFTs for each environment, such as test, development, preproduction, or production. CFT have also recently introduced an option called **dynamic reference to specify template values**. It is possible to store and manage values in other services. CFT dynamically retrieves these values for the specified reference as and when necessary during the stack building and change set operations. The following code blocks are the basic syntax for the parameters:

- The syntax in JSON is as follows:

```
"Parameters" : { "ParameterLogicalID" : {  
    "Type" : "DataType", "ParameterProperty" : "value"  
}  
}
```

- The syntax in YAML is as follows:

```
Parameters: ParameterLogicalID:  
  Type: DataType ParameterProperty: value
```

A maximum of 60 parameters can be defined in an individual CFT. It is essential to declare a unique logical name (that is, `ParameterLogicalID`) for each parameter within the template. The parameter data type can be the `String`, `Number`, `CommaDelimitedList`, or AWS-specific type.

AWS-specific parameters

At runtime, when creating a stack from a template, you may need to get values from the AWS environment. For example, if an AWS architect or developer has written a CFT for a specific time, as and when required, one of the team members needs to run this template to create a stack. While running the template, it needs to identify the AWS region in which it is executed. Based on the AWS region, the template uses a suitable key-value pair for an EC2 instance. AWS uses a number of pseudo-parameters, which are populated from the resources that are currently available in the specific region. For example, the number of AZs may vary from region to region, or the list of EC2 key-value pairs may vary from region to region and from AWS account to account. For example, when a template that uses these pseudo-parameters is executed to create a stack, it enables an administrator to select the desired AZs for creating the EC2 instances.



Most AWS-specific parameters return multiple values in a drop-down list, apart from `AWS::EC2::Image::Id`.

The following table helps us to understand various pseudo-parameters and their usage:

AWS-specific parameters	Description
<code>AWS::EC2::AvailabilityZone::Name</code>	This provides a list of AZs in the current region. Only one value from a drop-down list can be selected, for example, <code>us-west-2a</code> .
<code>AWS::EC2::Image::Id</code>	This provides an Amazon EC2 image ID. It provides a textbox to enter a valid AMI ID, for example, <code>ami- ff5467egf</code> .
<code>AWS::EC2::Instance::Id</code>	This provides a list of Amazon EC2 instance IDs in the current region. Only one value from a drop-down list can be selected, for example, <code>i-0862FF253e23cfas2</code> .
<code>AWS::EC2::KeyPair::KeyName</code>	This provides a list of key pairs in a region. Only one value from a drop-down list can be selected, for example, <code>test-key</code> .
<code>AWS::EC2::SecurityGroup::GroupName</code>	This provides a list of security groups existing in a region. Only one value from a drop-down list can be selected, for example, <code>launch-wizard-1</code> .
<code>AWS::EC2::SecurityGroup::Id</code>	This provides a list of security groups along with their IDs that exist in a region. Only one value from a drop-down list can be selected, for example, <code>launch-wizard-1 (sg-28db6c88)</code> .
<code>AWS::EC2::Subnet::Id</code>	This provides a list of subnet IDs along with its CIDR range in a region. Multiple values from a drop-down list can be selected, for example, <code>subnet-31hj765a (172.31.48.0/20)</code> .
<code>AWS::EC2::Volume::Id</code>	This provides a list of EBS volume IDs along with names that are available in a region. Only one value from a drop-down list can be selected, for example, <code>vol-010968da1fwd265d8 (TestVol)</code> .
<code>AWS::EC2::VPC::Id</code>	This provides a list of VPCs along with IDs, CIDR range, and names available in a region. Only one value from a drop-down list can be selected, for example, <code>vpc-6c5fe40a (172.31.0.0/16) (DefaultVPC)</code> .

AWS::Route53::HostedZone::Id	This provides a list of hosted zones, along with their domain names and IDs, in an AWS Route 53 service. Only one value from a drop-down list can be selected, for example, testdomain.text (A5EF6W8F6S8FF).
List<AWS::EC2::AvailabilityZone::Name>	This provides a list of AZs in the current region and multiple AZs can be selected from the drop-down menu, for example, us-east-1a us-east-1c.
List<AWS::EC2::Image::Id>	This provides a list of Amazon EC2 AMI IDs. For this parameter type, the AWS console will not show a drop-down list; it will show a text box.
List<AWS::EC2::Instance::Id>	This provides a list of existing EC2 instances in a region along with the instance ID and name. Multiple EC2 instances can be selected, for example, i-0862FF253e23cfas2 (Test1) i-0862FF253e23xfax2 (Test2).
List<AWS::EC2::SecurityGroup::GroupName>	This lists the security groups only with the names available in the AWS Region. Multiple security groups can be selected from the drop-down menu, for example, launch-wizard-1 launch-wizard-2.
List<AWS::EC2::SecurityGroup::Id>	This lists the security groups with names and IDs available in the AWS Region. Multiple security groups can be selected from the drop-down menu, for example, launch-wizard-1 (sg-28db6c88) launch-wizard-2 (sg-56hs9g45).
List<AWS::EC2::Subnet::Id>	This lists all of the subnets along with IDs, CIDR range, and names. Multiple subnets can be selected from a drop-down menu, for example, subnet-5d9d652 (172.31.16.0/20) (DefaultSubnet-1C) subnet-a6sd325s (172.31.32.0/20) (DefaultSubnet-1E).
List<AWS::EC2::Volume::Id>	This lists all of the volumes along with their IDs and names in the region. Multiple values can be selected from a drop-down menu, for example, vol-369a8sd6689sd4587 (Vol1) vol-98df536rt9as3thc3 (Vol2).
List<AWS::EC2::VPC::Id>	This lists all of the VPCs in the region along with IDs, CIDR range, and names. Multiple VPCs can be selected from a drop-down menu, for example, vpc-65d3f69s (192.192.192.0/24) (Custom) vpc-69sd32rt (172.31.0.0/16) (DefaultVPC).

List<AWS::Route53::HostedZone::Id>	This lists all of the hosted zones available in the region in AWS Route 53. Multiple hosted zones can be selected from a drop-down menu.
------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------



All AWS-specified parameters starting with a list are allowed to select multiple values from a drop-down menu, except
 List<AWS::EC2::Image::Id>.

Optionally, one or more properties can be defined for each parameter. This helps to make the user interface much more natural at the time of creating a stack from the template, for example, by providing a drop-down list for valid values and showing * for each character when sensitive information is entered, such as a password. The list of various allowed parameter properties are defined as follows:

Properties	Description
AllowedPattern	Required: No This specifies a regular expression for a String type, to validate the manually entered string by the user at the time of creating the stack, for example, accepting the username or password.
AllowedValues	Required: No This restricts the user to enter only valid values that are specified in the list of values in an array. For example, the EC2 instance type can be only t2.micro, m3.large, or m4.large.
ConstraintDescription	Required: No When user input does not match with the constraint message to be displayed, it can be defined here; for example, the username or password can only contain [A-Z, a-z, 0-9].
Default	Required: No This specifies a value to be used when the user has not provided any value as input.
Description	Required: No This describes the parameters; up to 4,000 characters can be written.
MaxLength	Required: No This defines the maximum number of characters to be allowed for String type parameters. It can be defined by specifying the integer value.
MaxValue	Required: No This defines the largest value allowed for Number types. It can be defined by specifying the integer value.
MinLength	Required: No This defines the minimum number of characters to be entered for String type parameters. It can be defined by specifying the integer value.

MinValue	Required: No This defines the smallest value allowed for Number types. It can be defined by specifying the integer value.
NoEcho	Required: No By enabling this property, sensitive information can be masked by *.
Type	Required: Yes This defines the data type for the parameter (data type). It can be defined as String, Number, List<Number>, or CommaDelimitedList.

A list of the allowed properties in the parameter sections

The following example demonstrates how to parameterize the EC2 instance type at the time of creating a stack from a CFT. The drop-down list allows you to choose any one EC2 instance type from t2.micro, m1.small, or m1.large. The default value is t2.micro:

- This is the example in JSON:

```
"Parameters" : { "InstanceTypeParameter" : {
    "Type" : "String",
    "Default" : "t2.micro",
    "AllowedValues" : ["t2.micro", "m1.small", "m1.large"],
    "Description" : "Enter t2.micro, m1.small, or m1.large.
Default is
t2.micro."
  }
}
```

- This is the example in YAML:

```
Parameters: InstanceTypeParameter:
Type: String Default: t2.micro AllowedValues:
- micro
- small
-- large
Description: Enter t2.micro, m1.small, or m1.large. Default is
t2.micro.
```

More more information, you can refer to <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/parameters-section-structure.html>.

In general, a function is available to be used only in a specific programming language; its implementation is handled specifically by the compiler and it is called an **intrinsic function**. AWS CloudFormation provides several built-in functions to help to manage the stack, such as Fn::Base64, the Condition functions, Fn::FindInMap, Fn::GetAtt, Fn::GetAZs, Fn::ImportValue, Fn::Join, Fn::Select, Fn::Split, Fn::Sub, and Ref. These functions can only be used in specific parts of a template, such as the resource properties in the Resources section, the Output section, the metadata attributes, and the UpdatePolicy attributes:

Intrinsic function	Description
Fn::Base64	This function converts the parameterized string into the Base64 representation. Ideally, it helps to pass encoded data to the Amazon EC2 instance using the <code>UserData</code> property.
Fn::Cidr	By splitting apart a defined CIDR block into a list of small allocations, this function performs subnet calculations. These small allocations are then used for address allocation.
Conditions Functions	Conditional functions can be helpful while creating a CloudFormation stack based on some conditions. While creating a CloudFormation stack, there may be a need to build a few resources based on certain conditions. For example, if the target environment is <i>Staging</i> , the EC2 instance type should be <code>micro</code> , and if the target environment is <i>Production</i> , the EC2 instance type should be <code>large</code> . The set of conditional intrinsic functions, such as Fn::If, Fn::Equals, and Fn::Not, can refer to other conditions and values from the Parameters and Mappings sections of a template to conditionally create or update a stack.
Fn::FindInMap	This helps to find appropriate values corresponding to a two-level map declared in the <code>Mappings</code> section. For example, in your CFT, you can write a mapping to use an appropriate AMI in a respective region. Depending on the region, CloudFormation can take an appropriate AMI ID for creating an EC2 instance.
Fn::GetAtt	There is a very common need to refer to some attributes of other resources for creating a new AWS resource. For example, a template needs an ELB endpoint to create a CNAME in a Route 53. For this scenario, the Fn::GetAtt intrinsic function can be used to get the value of an attribute from another resource created in the same template.

Fn::GetAZs	This is used for fetching a list of AZs in a region where the function is executed.
Fn::ImportValue	This is recommended to create multilayered CloudFormation stacks. In a multilayered CloudFormation stack, it may be necessary to refer to the resources created in another stack. The Fn::ImportValue function returns the value of an output exported by another stack.
Fn::Join	This intrinsic function helps to concatenate a set of values into a single value.
Fn::Select	This intrinsic function simply returns a single object from a list of objects.
Fn::Split	This intrinsic function is opposite to the Fn::Join function. This function splits a given string into a list of string values based on a given delimiter.
Fn::Sub	This intrinsic function substitutes a variable in an input string with specified values.
FN::Transform	This intrinsic function specifies a macro for performing custom processing on part of a stack template, such as find and replace, or any complex validation and manipulation.
Ref	This intrinsic function returns the value of the specified parameter or resource referenced by this function.

While writing a template, a customized value for a parameter can be retrieved using the Ref intrinsic function, as shown in the following code examples:

- This is the example in JSON:

```
"Ec2Instance" : {
  "Type" : "AWS::EC2::Instance", "Properties" : {
    "InstanceType" : { "Ref" : "InstanceTypeParameter" },
    "ImageId" : "ami-2f726546"
  }
}
```

- This is the example in YAML:

```
Ec2Instance:
  Type: AWS::EC2::Instance
  Properties:
    InstanceType:
      Ref: InstanceTypeParameter
    ImageId: ami-2f726546
```

For more information, you can refer to <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/parameters-section-structure.html>.

Other examples of parameter properties, DBPort and DBPwd, are created using parameter properties such as Default, Description, MinValue, MaxValue, NoEcho, Description, MinLength, MaxLength, and AllowedPattern, respectively:

- This is the example in JSON:

```
"Parameters" : { "DBPort" : {  
    "Default" : "3306",  
    "Description" : "TCP/IP port for the database", "Type" :  
    "Number",  
    "MinValue" : "1150",  
    "MaxValue" : "65535"  
},  
    "DBPwd" : {  
    "NoEcho" : "true",  
    "Description" : "The database admin account password", "Type"  
    : "String",  
    "MinLength" : "1",  
    "MaxLength" : "41", "AllowedPattern" : "[a-zA-Z0-9]*"  
}  
}
```

- This is the example in YAML:

```
Parameters: DBPort:  
Default: 3306  
Description: TCP/IP port for the database Type: Number  
MinValue: 1150  
MaxValue: 65535 DBPwd:  
NoEcho: true  
Description: The database admin account password Type: String  
MinLength: 1  
MaxLength: 41  
AllowedPattern: "[a-zA-Z0-9]*"
```

For more information, you can refer to <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/parameters-section-structure.html>.

Pseudo-parameters are predefined by AWS CloudFormation and do not need to be predefined before being used in the template. They are used in the same way that custom-defined parameters are used, with the help of the `Ref` function. The list of pseudo-parameters that are currently available are presented in the following table:

Pseudo parameter	Description
<code>AWS::AccountId</code>	This returns a 12-digit AWS account ID.
<code>AWS::NotificationARNs</code>	This returns ARNs for each resource in the current stack. To fetch a single ARN from the list, the <code>Fn::Select</code> intrinsic function can be used.
<code>AWS::NoValue</code>	When you specify this parameter as the return value in the <code>Fn::If</code> intrinsic function, it removes the corresponding resource property.
<code>AWS::Partition</code>	AWS regions are divided into three partitions, as follows: <ul style="list-style-type: none">• <code>aws</code>: The public partition has more than 15 regions.• <code>aws-cn</code>: AWS China has two regions.• <code>aws-us-gov</code>: AWS GovCloud has two regions, US-West and US-East. With the help of the <code>Partition</code> pseudo-parameter, it is possible to determine what services are available in a region, or what regions a service is available in. While accessing partition pseudo-parameters, it is possible to specify the region.
<code>AWS::Region</code>	This returns the name of the AWS Region where the resource is being created.
<code>AWS::StackId</code>	This returns the stack ID as specified with the <code>aws cloudformation create-stack</code> command.
<code>AWS::StackName</code>	This returns the stack name as specified with the <code>aws cloudformation create-stack</code> command.

A list of pseudo-parameters

It is recommended that you thoroughly understand the template structure and every element of the template in order to start using CloudFormation templates for automating the deployment process.

Mappings

The `Mappings` section is optional in a template. This section matches a key to a corresponding set of named values. For example, the AMI ID for Amazon Linux is `ami-22ce4934` in Northern Virginia and `ami-9e247efe` in Northern California. With the help of the `Mappings` section, we can have a smart template based on the region where it is running and it will take the right AMI ID to launch an EC2 instance.



In the `Mappings` sections, values from parameters, pseudo-parameters, or intrinsic functions cannot be used.

The `Mappings` section begins with `mappings` as a key name. It is required to have keys and values; both must be literal strings. The following syntax in JSON and YAML helps us to understand this. Mappings in the top line indicate the beginning of a section as a key and the string. The `Mapping01` string indicates the variable or parameter to observe for mapping a value. Usually, the output of a pseudo- or AWS-specific parameter is stored in this variable or parameter. The values for `Key01`, `Key02`, and `Key03` could be different in that variable or parameter:

- This is the syntax in JSON:

```
"Mappings" : {  
    "Mapping01" : {  
        "Key01" : {  
            "Name" : "Value01"  
        },  
        "Key02" : {  
            "Name" : "Value02"  
        },  
        "Key03" : {  
            "Name" : "Value03"  
        }  
    }  
}
```

- This is the syntax in YAML:

```
Mappings: Mapping01:  
  Key01:  
    Name: Value01  
  Key02:  
    Name: Value02  
  Key03:  
    Name: Value03
```

For more information, you can refer to <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/mappings-section-structure.html>.

The output of the `AWS::Region` pseudo-variable is stored in a `RegionMap` variable; if it returns `us-east-1`, then it should use the appropriate EC2 AMI available in that region. For example, here, `ami-6411e20d` and `32` are used as key values. These values can be anything meaningful to the project or enterprise. Here, we have used a `32`-bit key; a `64`-bit key will also be suitable to use the AMI. You can use an intrinsic function, such as `Fn::FindInMap`, to automatically populate a suitable value from the `Mappings` section:

- This is the example in JSON:

```
{  
  "AWSTemplateFormatVersion" : "2010-09-09",  
  "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" }  
    }  
  },  
  "Resources" : { "myEC2Instance" : {  
    "Type" : "AWS::EC2::Instance", "Properties" : {  
      "ImageId" : { "Fn::FindInMap" : [ "RegionMap", { "Ref" :  
        "AWS::Region" }, "32" ] },  
      "InstanceType" : "m1.small"  
    }  
  }  
}
```

- This is the example in YAML:

```
AWSTemplateFormatVersion: "2010-09-09" 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"  
"32": "ami-9c03a89d"  
"64": "ami-a003a8a1" Resources:  
myEC2Instance:  
Type: "AWS::EC2::Instance" Properties:  
ImageId: !FindInMap [RegionMap, !Ref "AWS::Region", 32]  
InstanceType: m1.small
```

For more information, you can refer to <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/mappings-section-structure.html>.

If it is a test or production environment in the us-east-1 region, then use ami-8ff710e2 and ami-f5f41398, respectively. Additionally, when the region is us-west-2, for the test and production environments, use ami-eff1028f and ami-d0f506b0, respectively.

Conditions

Conditions can be used in a template for reusing the same template again and again, based on a scenario. The template behaves differently based on the conditions satisfied. For example, in the Parameter section, when the environment type is selected as test, then the EC2 instance is created with basic capabilities, such as a small volume size. Similarly, when the environment is selected as production, the EC2 instance is created with a higher configuration, such as larger EBS volumes and larger instance size.

An example template for the CloudFormation condition can be found at <http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/conditions-section-structure.html>.



Conditions in a template can be modified only when resources are added, modified, or deleted.

In order to create a resource based on a condition, it is essential to specify the statement in at least three different sections in a template, that is, the Resource or Output sections, the Parameter section, and the Conditions section.

In the `Parameter` section, you can define the input value that evaluates whether the input conditions are true or false. In the `Conditions` section, you can specify the condition using an intrinsic function to determine whether to create an associated resource or not.

Finally, in the `Resources` and `Output` sections, associate conditions with the resources in the `Resources` section or the output in the `Outputs` section, which should be created conditionally. Use the condition key and the condition's logical ID to associate it with a resource or output. To conditionally specify a property, use conditional functions such as the `Fn::And`, `Fn::Equals`, `Fn::If`, `Fn::Not`, or `Fn::Or` functions. The syntax in JSON and YAML is as follows:

- This is the syntax in JSON:

```
"Conditions" : {  
    "Logical ID" : {Intrinsic function}  
}
```

- This is the syntax in YAML:

```
Conditions:  
  Logical ID:  
    Intrinsic function
```

Here is an example of the CloudFormation conditions:

```
{  
  "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" },  
      "eu-west-1" : { "AMI" : "ami-24506250", "TestAz" : "eu-west-1a" },  
      "sa-east-1" : { "AMI" : "ami-3e3be423", "TestAz" : "sa-east-1a" },  
      "ap-southeast-1" : { "AMI" : "ami-74dda626", "TestAz" : "ap-  
southeast-1a"  
    },  
      "ap-southeast-2" : { "AMI" : "ami-b3990e89", "TestAz" : "ap-  
southeast-2a"  
    },  
      "ap-northeast-1" : { "AMI" : "ami-dcfa4edd", "TestAz" : "ap-  
northeast-1a"  
    }  
  }  
},
```

```
"Parameters" : { "EnvType" : {
  "Description" : "Environment type.", "Default" : "test",
  "Type" : "String",
  "AllowedValues" : ["prod", "test"], "ConstraintDescription" : "must
specify prod or 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" : {
    "InstanceId" : { "Ref" : "EC2Instance" },
    "VolumeId" : { "Ref" : "NewVolume" }, "Device" : "/dev/sdh"
  }
},
"NewVolume" : {
  "Type" : "AWS::EC2::Volume", "Condition" : "CreateProdResources",
  "Properties" : {
    "Size" : "100",
    "AvailabilityZone" : { "Fn::GetAtt" : [ "EC2Instance",
      "AvailabilityZone"
    ] }
  }
},
"Outputs" : {
  "VolumeId" : {
    "Value" : { "Ref" : "NewVolume" }, "Condition" : "CreateProdResources"
  }
}
}
```

The preceding code demonstrates the use of `Conditions` in the CloudFormation template. You can see in the code that it encompasses two environment types, that is, `prod` and `test`. Users can select an environment type as a parameter while running the CFT. If a user selects the `prod` environment, then CFT generates an additional volume of size 100 GB.

For more information on conditions, you can refer to <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/conditions-section-structure.html>.

Transform

The `Transform` section is optional in a template. This section carries statements to condense and simplify template authoring. For example, in hundreds of lines of CFT, a resource declaration of multiple lines can be replaced by a single line. These statements are declarative statements and tell AWS CloudFormation how to process the template. It uses simple and declarative language with a powerful macro system. All transform functions are resolved again and again on every change set created for a stack. At the time of writing this book, the following transform declarative statements are supported:

`AWS::Serverless`

This is a specific version of an **AWS Serverless Application Model (SAM)**. It helps us to deploy an AWS Lambda-based application; it is also referred to as a serverless application and is composed of AWS Lambda functions triggered by events. The second declarative statement is as follows:

`AWS::Include`

This helps us to include a separate template snippet, for example, in order to perform a common repetitive task as a separate CFT. In an enterprise application, it may be required to create a web server for various projects. With the help of this transform, it is possible to call the web server template in the main template for that particular project.

Resources

The only section required to run any AWS CFT is the Resources section with at least one resource to create and include in the stacks. One template can have only one resource, where multiple resources separated by a comma can be specified. When the stack is created from a template, all of the specified resources will be grouped logically in the same stack. The syntax for the Resources section is as follows:

- This is the syntax in JSON:

```
"Resources" : {
  "Logical ID" : {
    "Type" : "Resource type", "Properties" : {
      Set of properties
    }
  }
}
```

- This is the syntax in YAML:

```
Resources: Logical ID:
Type: Resource type Properties:
Set of properties
```

The Resources fields are explained in the following table:

Resources fields	Description
Logical ID	Each logical ID must be unique within each template and can contain only alphanumerics (A-Za-Z0-9). It is recommended that these IDs are given meaningful and relevant logical names. The logical ID of one resource can be used to perform further tasks on the same resource. For example, a logical ID of an EC2 instance can be used to add an extra EBS volume in the same template.
Type	The resource type specifies the type of AWS resource that is being created, for example, an AWS EC2, EBS, VPC, or a subnet. A detailed list of resource types can be obtained at https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-template-resource-type-ref.html
Properties	Resource properties configure the characteristics of the AWS resource that is being created. For various AWS resources, some properties are essential to define and the rest can be optional. For example, in the case of creating an EC2 instance, <code>ImageId</code> is one of the parameters that must be specified.

Resource fields

Outputs

The `Outputs` section is optional in a template. This section can be used to declare values to be used in another template, return a response (to describe a stack call), or to view the AWS CloudFormation console, for example, to display a public or private DNS name to access an EC2 instance.

In the following syntax, we can see that it begins with the `Outputs` key name. In a single template, a maximum of 60 outputs can be declared:

- This is the syntax in JSON:

```
"Outputs" : {  
    "Logical ID" : {  
        "Description" : "Information about the value", "Value" :  
        "Value to return",  
        "Export" : {  
            "Name" : "Value to export"  
        }  
    }  
}
```

- This is the syntax in YAML:

```
Outputs: Logical ID:  
Description: Information about the value  
Value: Value to return  
Export:  
Name: Value to export
```

For more information on the `outputs` section, you can refer to <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/outputs-section-structure.html>.

The following table describes each `Outputs` field:

Output fields	Description
Logical ID	This is required, must be unique for each output within the template, and can be only alphanumeric (A-Za-z0-9).
Description (optional)	This is an optional String type and is, at most, 4 K in length.
Value (required)	This is required and it can have literals, parameter references, pseudo-parameters, a mapping value, or intrinsic functions.

Export (optional)	<p>This is optional and can be declared to export a resource to be used in another stack, in other words, for a cross-stack reference. These are a few important points to remember when exporting a resource:</p> <ul style="list-style-type: none">With an AWS account, export names must be unique within a region.Cross-stack references can be created for use within one region only. The <code>Fn::ImportValue</code> intrinsic function can only import an exported value in some stacks.Stacks can't be deleted when one or more resource is cross-referenced by another stack.It is also not possible to modify or remove an output value referenced by another stack.
-------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Descriptions of each of the output fields

The following code example helps us to understand the `Outputs` section. It is called `StackVPC` and returns the ID of a VPC, then exports the value for a cross-stack reference with the name `VPCID` appended to the stack's name:

- This is the example in JSON:

```
"Outputs" : {  
    "StackVPC" : {  
        "Description" : "The ID of the VPC", "Value" : { "Ref" :  
            "MyVPC" },  
        "Export" : {  
            "Name" : { "Fn::Sub" : "${AWS::StackName}-VPCID" }  
        }  
    }  
}
```

- This is the example in YAML:

```
Outputs: StackVPC:  
Description: The ID of the VPC Value: !Ref MyVPC  
Export:  
Name: !Sub "${AWS::StackName}-VPCID"
```

For more information, you can refer to <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/outputs-section-structure.html>.

A sample CloudFormation template

The following reference URLs provide various ready-to-use CFTs to match the general needs of an enterprise. These templates can be used directly or be modified as per the actual business need. Once templates are written, the partial code can be referred, or copied and pasted into another template for quickly creating new templates.

The following are important reference URLs for sample CFTs:

- For CloudFormation sample templates, region-wise, refer to <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-sample-templates.html>.
- For an AWS CFT solution for various AWS services, refer to <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/sample-templates-services-us-west-2.html>.

CloudFormer

CloudFormer can automatically generate a CFT from existing AWS resources in your AWS account. It stores the CFT in a target S3 bucket specified by you. Unlike writing a template from scratch, CloudFormer performs a reverse-engineering task and makes your life easier by generating a template from existing AWS resources in your account. This template can be used as it is for DR, or you can use them for customizing your infrastructure based on your needs. At the time of writing this book, CloudFormer is still in beta version.



More about CloudFormer can be found at <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-using-cloudformer.html>.

Rolling updates for auto scaling groups

AWS CloudFormation provides you with a mechanism to control how an auto scaling group updates your resources using the `UpdatePolicy` attribute. If you do not configure your settings correctly, a rolling update on an auto scaling group may be performed unexpectedly. You can address this scenario by using the `AutoScalingRollingUpdate` policy, which supports a number of options to configure your template.

Here is an example of the updated policy for rolling updates, which can be found in official AWS documentation at <https://aws.amazon.com/premiumsupport/knowledge-center/auto-scaling-group-rolling-updates/>. You can also refer to a very good article on rolling updates with CloudFormation at <https://cloudonaut.io/rolling-update-with-aws-cloudformation/>.

CloudFormation best practices

CloudFormation best practices are as follows:

- Always give meaningful and relevant names to AWS CloudFormation templates and resources.
- Make sure the resources used by a CloudFormation template exist in the region where it is being executed to create a stack, for example, resources such as an EC2 key pair. It can be also be created dynamically using templates, but, if it is hardcoded, make sure it exists in the relevant region.
- Write a template and create a stack for each layer, for example, a separate stack for web servers, application servers, and networks. It will help us to minimize downtime and efficiently manage and maintain infrastructures.
- Use a cross-stack reference. This will help us to integrate resources from multiple templates into one template, especially when a separate stack is created for each layer.
- It is best practice to provide essential IAM privileges to the IAM user executing a CloudFormation template to create a stack. This may involve creating or manipulating various AWS resources. Make sure that sufficient permissions are granted.

- At the time of creating a stack, AWS CloudFormation only validates the syntax. It doesn't check for required IAM privileges or soft limits for resources that are being created. Make sure that executing a template to create a stack doesn't attempt to cross a soft limit. If so, ask AWS to extend it.
- Reuse the whole or part of the template as and when required with adequate modifications to meet business requirements.
- Use a nested stack to perform common template patterns.
- It is advised not to embed credentials or sensitive information in any template. Parameters, constraints, AWS-specific parameters, and properties can be used effectively to use the same template dynamically and to avoid invalid user input.
- A set of Python helper scripts is maintained and periodically updated by AWS for installing software and starting services on an Amazon EC2 instance. It is recommended that you always use the latest helper scripts.
- Before creating a stack, validate the template syntax (JSON/YAML).
- Stack resource updates, deletion, or modification should be carried out by modifying a template rather than directly performing the action.

Summary

- CloudFormation can be used to create **Infrastructure-as-Code (IaC)**.
- AWS does not charge you for using the CloudFormation service.
- AWS CFTs describes all AWS resources and their properties in JSON or YAML format.
- You can either upload a CloudFormation template created locally or you can point to an S3 URL to create a CloudFormation stack.
- Executing a CFT creates a defined set of AWS resources. A group of these AWS resources defined in CloudFormation is called a stack.
- While creating a CloudFormation stack, if the template path is pointing to the local machine, then it will automatically upload the CloudFormation template to the AWS S3 bucket in the relevant region.
- CloudFormation automatically identifies the resource dependencies and creates them sequentially.
- Generally, it is recommended that you write a template for each layer of architecture, that is, the web layer and the database layer.

- You can also create a CloudFormation Template using AWS CloudFormation Designer.
- At the time of creating a stack from the template, AWS CloudFormation only checks for the syntax error in JSON/YAML notation.
- CloudFormation does not check whether the IAM user executing the template has sufficient privileges to complete the template execution or not.
- CloudFormation does not check whether any resource creation may violate AWS soft limits for the resources in the account.
- An existing CloudFormation stack can be updated by submitting a modified version of the original stack template, or by giving different input parameter values during the execution.
- CloudFormation `DeletionPolicy` can be used to configure what happens to the underlying resources when the stack is deleted.
- CloudFormation `DeletionPolicy` can have three attributes, that is, `delete`, `retain`, and `snapshot`.
- When you specify the `retain` attribute in `DeletionPolicy`, CloudFormation retains the associated resource even after the stack deletion.
- The `Resource` section is mandatory in a CFT.
- The `Metadata` section is an optional section in a CFT.
- The `Parameters` section is optional in a CFT. It can be used to pass values into the template to be customized while creating a stack.
- Pseudo-parameters are predefined by AWS CloudFormation and do not need to be defined before they are used in the template.
- The mappings section in a CFT matches a key to a corresponding set of named values.
- Conditions can be used in a template for reusing the same template again and again, based on a scenario.
- CloudFormer can automatically generate a CFT from existing AWS resources in your AWS account.
- AWS CloudFormation provides you with a mechanism to control how an auto scaling group updates your resources, using the `UpdatePolicy` attribute.

16

Understanding Elastic Beanstalk

Traditionally, when a web application is deployed on AWS, it may require investing energy in choosing proper AWS services, such as EC2, ELB, and autoscaling. We may need to create and configure an AWS resource from scratch to host a web application. It could be a troublesome undertaking for designers to assemble the framework, arrange the operating system, install the necessary dependencies, and then deploy the web services. AWS Elastic Beanstalk wipes out the need to physically build a framework for the designer and makes it accessible for them to rapidly send and deal with a web application on AWS of any scale.

Developers only need to upload the code. All other aspects, such as capacity provisioning, building, and configuring AWS resources, such as EC2 instances, ELB, autoscaling, and application health monitoring, will be handled by Elastic Beanstalk. Developers do get full access to each of the characterized AWS assets, controlling a web application to clean the setup. In this chapter, we will cover the following topics:

- Introduction to Elastic Beanstalk
- Elastic Beanstalk components
- Architectural concepts
- Elastic Beanstalk-supported platforms
- Creating a web application source bundle
- Getting started using Elastic Beanstalk
- The version life cycle
- Deploying web applications to Elastic Beanstalk environments
- Monitoring the web application environment
- Elastic Beanstalk best practices

Introduction to Elastic Beanstalk

AWS Elastic Beanstalk is an orchestration service provided by AWS. It can deploy applications that orchestrate a number of AWS services, such as EC2 instances, S3 buckets and objects, CloudWatch, SNS, ELB, and autoscaling. At the time of writing, Elastic Beanstalk supports web applications developed in Java, PHP, .NET, Node.js, Python, Docker, Ruby, and Go. It also supports web servers, such as Apache, NGINX, Passenger, and IIS. An easy way to start working with AWS Elastic Beanstalk is through the AWS web console. AWS also supports CLIs, APIs, and SDKs to work with AWS Elastic Beanstalk. There are no additional charges for using AWS Elastic Beanstalk; charges only apply for using the underlying resources, such as EC2, ELB, and autoscaling.

Most of the deployment and infrastructure tasks, such as uploading a newer version of a web application and changing the size of the Amazon EC2 instances, can be done directly from the Elastic Beanstalk web console. The following diagram helps us to understand the application deployment life cycle in Elastic Beanstalk:

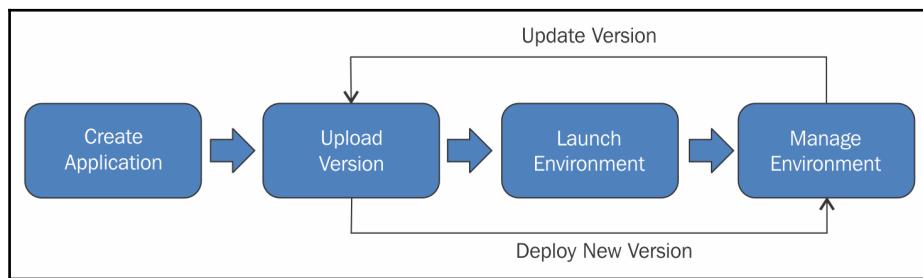


Figure 16.1: Application deployment and life cycle management on AWS Elastic Beanstalk

The preceding diagram, at a very high level, describes the way web applications are deployed on AWS Elastic Beanstalk. Initially, a web application is developed with a preferred programming platform on a developer's machine. Once it is developed, the source code is converted into a source bundle (for example, in Java), and the source bundle is converted into a .war file. This is an initial version of a web application. Once the initial version of the source bundle has been uploaded, Elastic Beanstalk automatically launches and configures the underlying infrastructure for running the source bundle. Over time, as business requirements change, it is also possible to upload a newer version of a web application.

The main purpose of AWS Elastic Beanstalk is to set developers free from creating and configuring AWS resources (that is, infrastructure) and allow them to focus solely on application development. If they are comfortable with creating and configuring AWS resources to host a web application, they can use CloudFormation to write templates and create a stack.

Elastic Beanstalk components

The following lists various Elastic Beanstalk components that work together to make it possible to deploy and manage custom applications easily in the AWS cloud:

- **Application:** This is a logical collection of Elastic Beanstalk components, including the environment, versions, and environment configuration. For easy understanding, it can be imagined as a folder.
- **Application version:** This refers to a specific source code version for a web application. It points to an Amazon S3 object containing deployment code, such as a Java .war file. The application version is part of an application. Each application can have multiple versions. Generally, applications run with the latest code version. At times, multiple versions of an application may run simultaneously for catering to users in a different location or for testing purposes.
- **Environment:** There are two types of environment:
 - The web server environment, to listen and process HTTP(S) requests
 - The worker environment, to process a background task that listens for messages on an Amazon SQS queue

At a given point in time, there is only a single application version by each environment. Creating an environment automatically creates underlying resources to run a specific application version.

- **Environment configuration:** This is a set of parameters and settings that define the way in which an environment and its associated resources behave. Elastic Beanstalk will automatically apply changes from the environment configuration to the existing resources. If required, it may delete existing resources and create new resources to match the environment configuration change.
- **Configuration template:** This is a starting point for creating unique environment configurations. The CLI or API can be used to create and modify the environment configuration.

Elastic Beanstalk environment tiers

In this section, let's understand some of the architectural concepts of Elastic Beanstalk. As you understood from the core concepts, an environment is an essential component used to deploy a web application. Creating a new environment requires selecting the appropriate environment tier, platform, and environment type. Broadly speaking, environment tiers are divided into two environments:

- **The web server environment:** This tier hosts a web application and handles HTTP(S) requests.
- **The worker environment:** This tier hosts a web application and handles long-running or scheduled background processing tasks.

A detailed description of these environment tiers will be given in the following subsections.

The web server environment tier

The following diagram helps us to understand the workings of the web server environment tier:

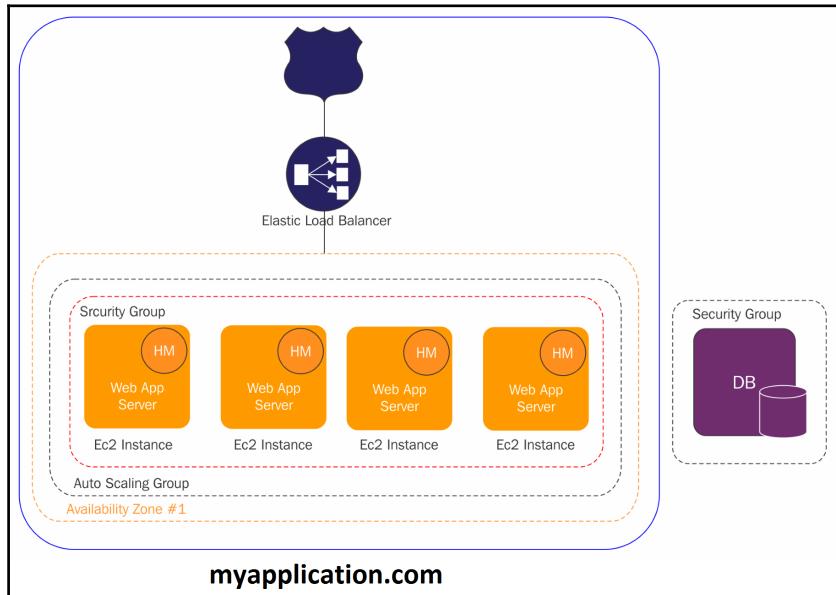


Figure 16.2: The web server environment tier and its components

The environment is a key component in an application, and it is highlighted as a solid blue line in the preceding diagram. It provisions the underlying AWS resources required to deploy and run the web application. The resources in an environment can include an ELB, autoscaling group, and at least one or more EC2 instances.

Every environment has a CNAME and an alias in Amazon Route 53 pointing to the ELB. The registered domain name (for example, `myapplication.com`) will forward the end user's request to access the web application on the CNAME. The CNAME points to an ELB, where the ELB is a part of the autoscaling group and it sits in front of the Amazon EC2 instances. Depending on the actual load on the application, the number of EC2 instances will be scaled in and out.

The container type plays an important role. It decides the software stack to be installed on each EC2 instance and its infrastructure topology. The software stack may include one or more components, such as a programming language (for example, Python, PHP, or Java), a web server (for example, Apache web server), a web container (for example, Tomcat or Passenger), and Docker containers.

In each environment, each EC2 instance runs one of the container types along with a software **host manager (HM)**. Later in this chapter, we will look at containers in detail. The HM is indicated in the preceding diagram as a circle in the top-right corner in each EC2 instance. The HM is responsible for the following:

- Aggregating events and metrics for retrieval using the web console, API, or CLI deploying the application
- Monitoring the application log files for critical errors
- Monitoring the application server
- Generating instance-level events
- Patching instance components
- Rotating application log files and publishing them to the S3 bucket

By default, Elastic Beanstalk creates a security group and attaches it to the EC2 instances. The security group acts as a firewall and allows anyone to connect on HTTP port 80. It is possible to customize security groups as per the web application's actual requirement.

The worker environment tier

The worker environment tier includes an autoscaling group, at least one or more EC2 instances, and an IAM role. Optionally, it also creates an Amazon SQS queue if there isn't one. Each EC2 instance in an autoscaling environment gets installed with a daemon and the necessary support files for the programming language of choice. Pulling requests from an Amazon SQS queue and then sending the data to the web application that is already running in the worker environment tier to process those messages is the primary function of the daemon. If there are multiple instances in a worker environment tier, then each instance has its own daemon that reads from the same Amazon SQS queue. The following diagram helps us to understand the components of the worker environment tier at a very high level:

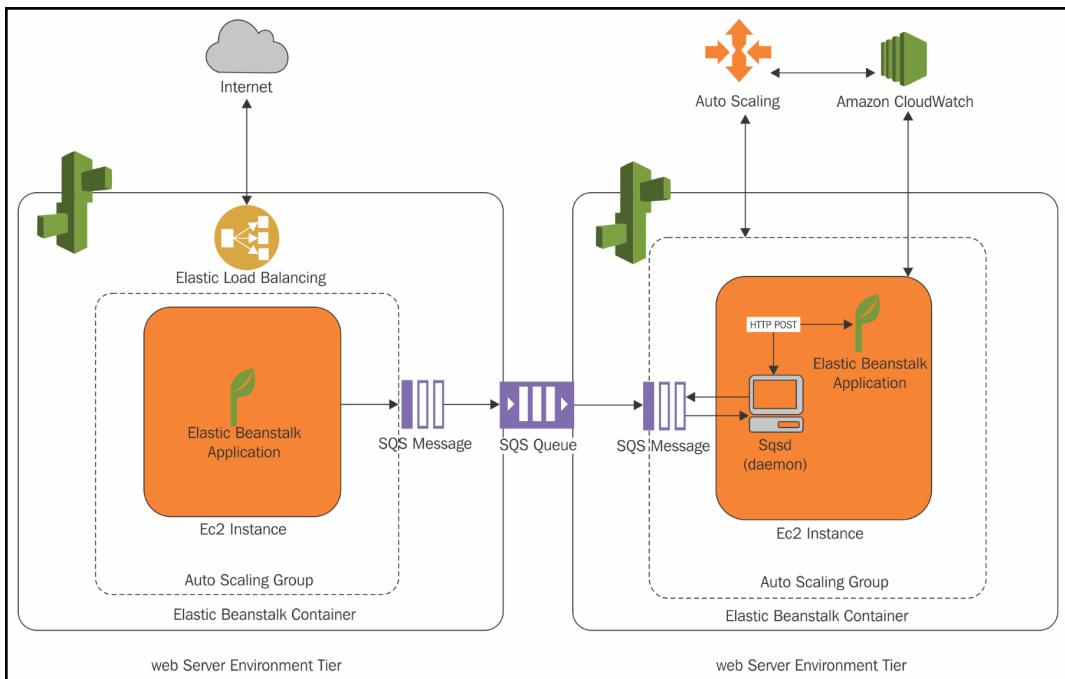


Figure 16.3: The worker environment tier and its components

Elastic Beanstalk-supported platforms

The platform supported by Elastic Beanstalk defines the instance software configuration. Platforms are broadly divided into two categories:

- **Preconfigured platforms:** These are also called Elastic Beanstalk-supported platforms. These platforms are available in multiple configurations of various programming languages, Docker containers, and/or web containers. Based on the selected platform, Elastic Beanstalk will install the specific stack of software on one or more Amazon EC2 instances. At present, all Linux platforms are running on Amazon Linux 2017.03 (64-bit). At a very high level, preconfigured supported platforms include the following:
 - Packer builder
 - Single container Docker
 - Multicontainer Docker
 - Preconfigured Docker Go
 - Java SE
 - Java with Tomcat
 - .NET on Windows Server with IIS Node.js
 - PHP
 - Python
 - Ruby (Passenger Standalone)
 - Ruby (Puma)
 - Node.js
 - Go
- **Custom platforms:** These allow for the creation of custom platforms based on one of the supported AMIs for OSes, such as RHEL, Ubuntu, and SUSE. The customized platform is created using a Packer tool. This is an open source tool and runs on major OSes. It makes it possible to create a customized platform using a customized language or a framework that is not currently supported in Elastic Beanstalk. The primary function of a Packer tool is to build a machine and container for multiple platforms from a single configuration.

Creating a web application source bundle

Whether deploying a new web application or updating a version for an existing web application in Elastic Beanstalk, it is essential to prepare a source bundle of source code. In general, the characteristics of source bundles for any programming language are as follows:

- It should be a single ZIP or WAR file.
- If there are multiple WAR files, it can be packed in a single ZIP file.
- The overall file size should not exceed 512 MB.
- It can have subdirectories but not parent directories.



To deploy a worker application in a worker tier, the application source bundle must also include the `cron.yaml` file.

A detailed understanding of preparing a source bundle can be obtained by visiting <https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/applications-sourcebundle.html>.

Getting started using Elastic Beanstalk

With the help of the following steps, an easy Elastic Beanstalk web application can be created, viewed, deployed, updated, and terminated.

Step 1 – signing in to the AWS account

You will need to understand the following guidelines to complete step 1 and sign in to the AWS account:

1. Sign in to the AWS account with the IAM user having sufficient privileges to work with AWS Elastic Beanstalk. In addition to the privileges to manipulate resources in Elastic Beanstalk, the IAM user also requires privileges to create, modify, and delete underlying resources in various AWS services, such as EC2, ELB, S3, and autoscaling. The requirements for such AWS service privileges vary from application to application.

2. Make sure the appropriate AWS region is selected from the top-right toolbar, as shown in the following screenshot:

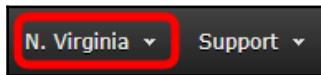


Figure 16.4: Selecting the appropriate AWS region to deploy a custom web application using Elastic Beanstalk

3. From the AWS dashboard, select **Elastic Beanstalk** from the **Compute** group, as shown in the following screenshot:

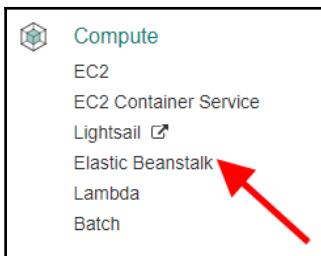


Figure 16.5: Selecting Elastic Beanstalk from the Compute group

Step 2 – creating an application

Creating a web application and preparing the source code to deploy a web application varies in each of the supported programming languages, such as Java, .NET, Node.js, PHP, Python, and Ruby. In this example, we have used a sample application provided by Amazon Elastic Beanstalk. It is very important to remember that AWS does not charge you for using Elastic Beanstalk services, but you need to pay for the resources you consume for hosting an application, such as EC2 and RDS. The steps for creating the sample application are as follows:

1. To create and deploy a sample web application on Elastic Beanstalk, follow the preconfigured URL provided by AWS: <https://console.aws.amazon.com/elasticbeanstalk/home#/newApplication?applicationName=getting-started&environmentType=LoadBalanced>.



The URL will require you to log in to the AWS account with valid credentials and privileges to create and manipulate Elastic Beanstalk and underlying resources.

2. The AWS Elastic Beanstalk **Create a web app** dialog box will appear, as shown in the following screenshot:

The screenshot shows the 'Create a web app' dialog box. It has three main sections: Application information, Environment information, and Base configuration.

Application information: Application name is set to 'getting-started'. A note says 'Up to 100 Unicode characters, not including forward slash (/)'.

Environment information: Environment name is 'GettingStarted-env'. Domain is 'Leave blank for autogenerated value' with 'us-east-1.elasticbeanstalk.com' selected and a 'Check availability' button.

Description: An empty text area.

Base configuration:

- Tier:** Web Server (Choose tier)
- Platform:** Preconfigured platform (selected). A dropdown menu shows 'Choose a platform --'. Custom platform (NEW) is also listed with a dropdown menu 'Choose a custom platform --'.
- Application code:** Sample application (selected). A note says 'Get started right away with sample code.' Upload your code (radio button) is also listed with a note 'Upload a source bundle from your computer or copy one from Amazon S3.' and a 'Upload' button.

At the bottom are 'Cancel' and 'Review and launch' buttons.

Figure 16.6: Creating a sample web application by using the preconfigured link provided by Amazon Elastic Beanstalk



Usually, it is first necessary to create an application, as it is a logical container for the Elastic Beanstalk components. Once the application is created, create the environment tier to deploy the web application. Note that in this sample URL, everything will be created in one go.

3. Choose the appropriate platform and click on **Review and launch**.

Configure the following parameters to review and launch the preconfigured sample web application. In this sample web application, **Tomcat** is required to be selected, as shown in the following screenshot:

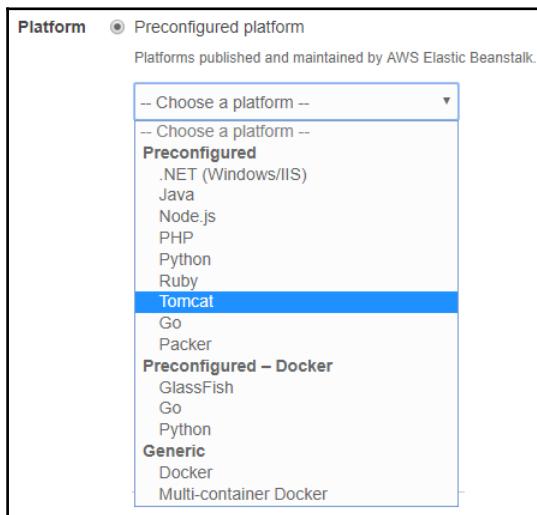


Figure 16.7: Selecting the appropriate platform

If the platform parameter is not configured, an error message is triggered, as shown in the following screenshot:

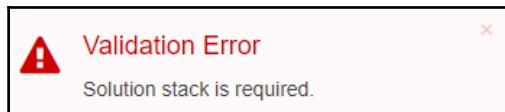


Figure 16.8: The preconfigured platform is required to configure

4. The preceding sample web application may take a few minutes to complete the underlying resource creation. The default configuration to create AWS resources is as follows:

The screenshot shows the 'Configure GettingStarted-env' page in the AWS Elastic Beanstalk console. At the top, there's a note about configuration presets: 'Configuration presets' with options for 'Low cost (Free Tier eligible)', 'High availability', and 'Custom configuration'. The 'Custom configuration' option is selected. Below this, it says 'Platform 64bit Amazon Linux 2017.03 v2.6.3 running Tomcat 8 Java 8' and a link to 'Change platform configuration'.

The page is divided into several sections:

- Tags:** Shows 'Tags: none' and a 'Modify' button.
- Software:** Shows 'AWS X-Ray: disabled', 'Rotate logs: disabled (default)', 'Log streaming: disabled (default)', 'Environment properties: 0', and a 'Modify' button.
- Instances:** Shows 'EC2 instance type: t1.micro', 'EC2 image ID: ami-5d063026', 'Root volume type: container default', 'Root volume size (GB): container default', 'Root volume IOPS: container default', and a 'Modify' button.
- Capacity:** Shows 'Environment type: load balancing, auto scaling', 'Availability Zones: Any', 'Instances: 1-4', and a 'Modify' button.
- Load balancer:** Shows 'Port: HTTP on port 80', 'Secure port: disabled', 'Cross-zone load balancing: disabled', 'Connection draining: disabled (de)', and a 'Modify' button.
- Rolling updates and deployments:** Shows 'Deployment policy: All at once', 'Rolling updates: disabled', 'Health check: disabled', and a 'Modify' button.
- Security:** Shows 'Service role: aws-elasticbeanstalk-service-role', 'Virtual machine key pair: --', 'Virtual machine instance profile: aws-elasticbeanstalk-ec2-role', and a 'Modify' button.
- Monitoring:** Shows 'Health check path: blank', 'Health reporting system: basic', and a 'Modify' button.
- Notifications:** Shows 'Email address: --' and a 'Modify' button.
- Network:** Shows 'VPC: --', 'Load balancer visibility: public', 'Load balancer subnets: none', 'Associate public IP address: --', 'Instance subnets: none', 'Security groups: none', and a 'Modify' button.
- Database:** Shows 'Engine: --', 'Instance class: --', 'Storage (GB): --', 'Multi-AZ: --', and a 'Modify' button.

At the bottom right, there are buttons for 'Cancel', 'Previous', and 'Create app'.

Figure 16.9: Configuring and creating resources for the web application

In the preceding screenshot, the major configuration and creation options along with their parameters are shown. They are described in more detail in the following table:

Configuration	Parameters
Tags	None. (In the current configuration, no tags are given; depending on the tags used, it appears in the configuration.)
Software	AWS X-Ray, Rotate logs, Log streaming, and Environment properties
Instances	EC2 instance type, EC2 AMI ID, Root volume type, Root volume size (GB), and Root volume IOPS
Capacity	Environment type, Availability Zones, and Instances
Load balancer	Port, Secure port, Cross-zone load balancing, and Connection draining
Rolling updates and deployments	Deployment policy, Rolling updates, and Health check
Security	Service role, Virtual machine key pair, and Virtual machine instance profile
Monitoring	Health check path and Health reporting system
Notifications	Email address
Network	VPC, Load balancer visibility, Load balancer subnets, Associate public IP address, Instance subnets, and Security groups
Database	Engine, Instance class, Storage (GB), and Multi-AZ

AWS resources and configuration, along with their parameters

By default, Elastic Beanstalk creates an application named `getting-started` and an environment named `Custom-env` using the following AWS resources:

- An EC2 instance to run the web application on a configured platform. In this sample web application, it is Tomcat.
- A security group for the EC2 instance. By default, it allows everyone to connect on HTTP port 80.
- An ELB accepts all incoming end user requests to access the web application and distributes the requests to the underlying healthy EC2 instances.
- The security group for the load balancer allows traffic on HTTP port 80. By default, traffic is not allowed on other ports.
- The autoscaling group replaces an unhealthy EC2 instance with a new instance. It scales out and scales in based on the traffic load on the application.

- An S3 bucket to store source code, logs, and other artifacts.
- CloudWatch alarms to monitor the load on the instances in the environment. It works with autoscaling to help in scaling EC2 instances in and out.
- AWS Elastic Beanstalk uses the CloudFormation stack to create underlying resources and change the configuration.
- A domain name that routes the end user's request to the web app, in the form of `subdomain.region.elasticbeanstalk.com`.
- Elastic Beanstalk creates a new web application version with a name sample application. This web application refers to the default Elastic Beanstalk sample application code files.
- The sample application code is deployed to `Custom-env`.



It is possible to change the default configuration by clicking on **Modify**. For example, if you have a compute-intensive or a memory-intensive web application, it is possible to change the EC2 instance type from `t1.micro` to `c4` or `x1`, respectively.

As shown in the following screenshot, Elastic Beanstalk will track the environment creation progress:

A screenshot of the AWS Elastic Beanstalk console showing the progress of creating a new environment named "GettingStarted-env". The top navigation bar shows "All Applications > getting-started > GettingStarted-env (Environment ID: e-9caxqmzkg8, URL: GettingStarted-env.kqdc2kvng.us-east-1.amazonaws.com)". Below the navigation, a message says "Creating GettingStarted-env" with a blue info icon, followed by "This will take a few minutes." A large black box contains a log of events:

```
2:19pm Created Auto Scaling launch configuration named:  
awseb-e-9caxqmzkg8-stack-AWSEBAutoScalingLaunchConfiguration-1Q9U3WVGQ2C60  
2:19pm Created security group named:  
awseb-e-9caxqmzkg8-stack-AWSEBSecurityGroup-QKTLTOK39FG  
2:19pm Created load balancer named:  
awseb-e-9-AWSEBLba-1BDBXR8WQ6FZX  
2:19pm Created security group named:  
sg-21c94b51  
2:18pm Using elasticbeanstalk-us-east-1-508708549947 as Amazon S3 storage bucket for environment data.  
2:18pm createEnvironment is starting.
```

Figure 16.10: Elastic Beanstalk tracks the environment creation progress

Step 3 – viewing information about the recently created environment

Once the Elastic Beanstalk application is created, it is possible to view information about the underlying resources from the environment dashboard in the Amazon Elastic Beanstalk management console. The environment dashboard shows the application health, application version, and the application's environment version. The Elastic Beanstalk environment indicates the pending state until the underlying AWS resources are created and the related web application is deployed on the environment:

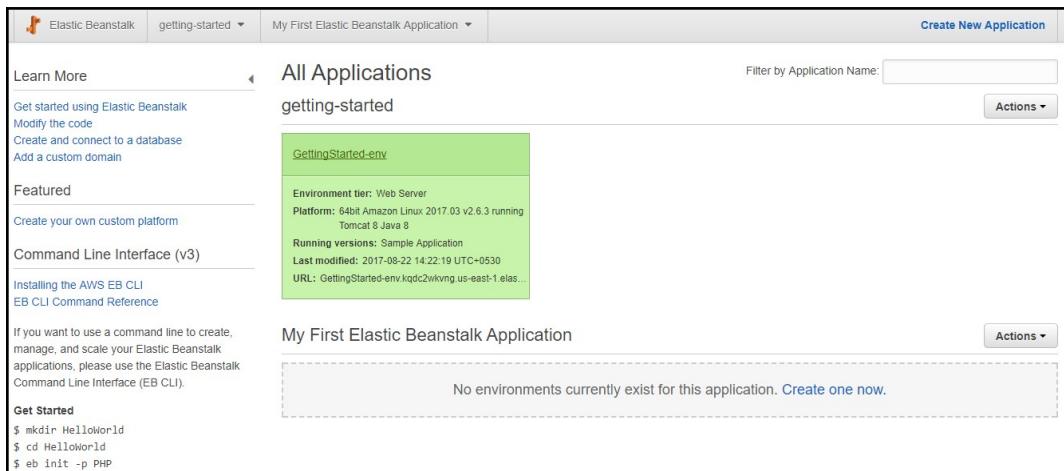


Figure 16.11: The Amazon Elastic Beanstalk dashboard

The preceding screenshot shows the basic AWS Elastic Beanstalk dashboard. Only one getting-started sample application and GettingStarted-env environment exist. By clicking on GettingStarted-env, it is possible to see the **Configuration**, **Logs**, **Health**, **Monitoring**, **Alarms**, **Managed Updates**, **Events**, and **Tags**, as shown in the following screenshot:

The screenshot shows the AWS Elastic Beanstalk dashboard for the 'getting-started' application. The left sidebar lists navigation options: Dashboard, Configuration, Logs, Health, Monitoring, Alarms, Managed Updates, Events, and Tags. The main area displays the 'Overview' for the 'GettingStarted-env' environment. It features a large green checkmark icon indicating 'Health: Green'. Below it, there is a 'Causes' button. To the right, the 'Running Version' is listed as 'Sample Application'. A 'Upload and Deploy' button is present. On the far right, there is a cartoon cat icon and text indicating the environment is running on '64bit Amazon Linux 2017.03 v2.6.3 running Tomcat 8 Java 8'. A 'Change' button is also visible. At the bottom, a 'Recent Events' table shows three log entries:

Time	Type	Details
2017-08-22 14:22:19 UTC+0530	INFO	Successfully launched environment: GettingStarted-env
2017-08-22 14:21:25 UTC+0530	INFO	Environment health has been set to GREEN
2017-08-22 14:20:26 UTC+0530	INFO	Created CloudWatch alarm named: awseb-e-9caxqmqzkg8-stack-AWSEBCloudwatchAlarmHigh-LA0042MM2H52

Figure 16.12: The Elastic Beanstalk application-specific dashboard

Step 4 – deploying a new application version

Over time, business needs change and a new version of a web application is developed. Elastic Beanstalk allows you to upload the new version of an application over an existing application. Elastic Beanstalk creates a new application version upon deploying a newer source bundle for an existing web application. A new version of your application can be deployed over an existing application as long as no update operations are in progress. Let's take a look at the steps to update a newer version for an existing web application:

1. Download the web application's updated source bundle from <https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/samples/java-tomcat-v3.zip>.



In the case of a custom-developed web application, it is necessary to create a source bundle with the updated source code.

2. Go to the **Elastic Beanstalk** dashboard.
3. Select the `getting-started` application and the `GettingStarted-env` environment.
4. Select **Upload and Deploy** from the **Overview** section, as shown in the following screenshot:

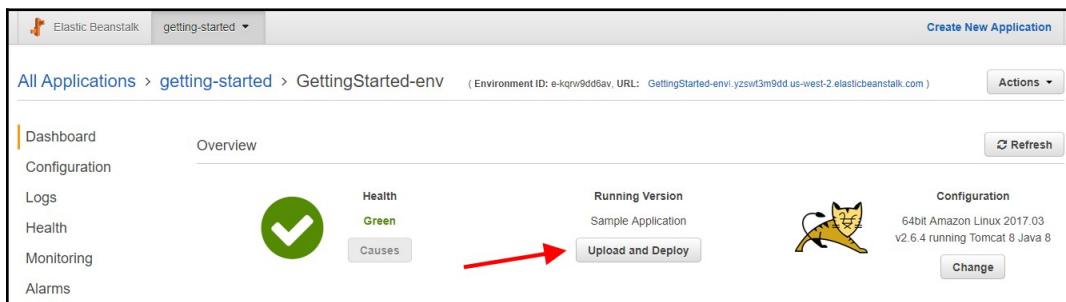


Figure 16.13: Application overview

5. We can now upload the new code, as shown in the following screenshot:

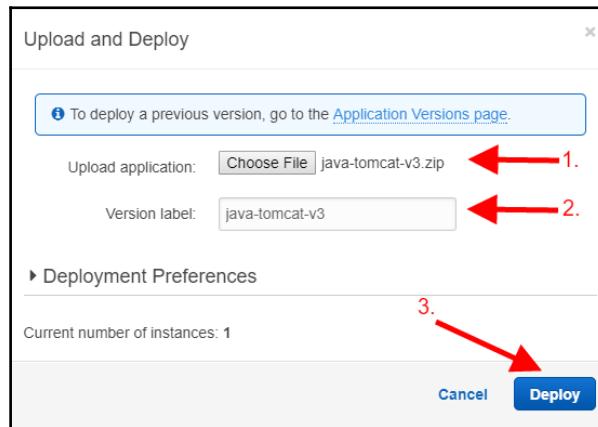


Figure 16.14: Uploading and Deploying the application

6. Click on **Choose File** to upload your source bundle.
7. By default, the version label will be the filename of the uploaded file. In this example, the filename is `java-tomcat-v3`. It can be changed to a meaningful and relevant name.
8. Finally, click on **Deploy** to deploy a new web application.

While Elastic Beanstalk is deploying a new version of a web application, it is possible to see a status of deployment on the web console. Once the deployment is completed, **Running Version** will be changed from sample application to `java-tomcat-v3` under the environment's **Overview**, as shown in the following screenshot:

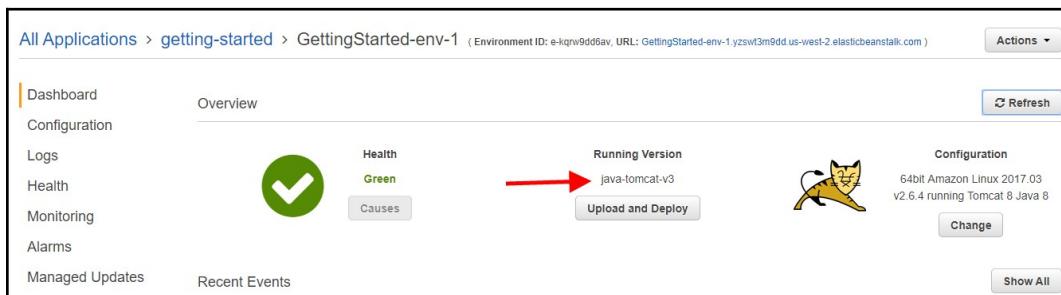


Figure 16.15: Application overview

Step 5 – changing the configuration

Amazon Elastic Beanstalk allows us to customize the web application environment at runtime for better performance. Some of the configuration changes can be simple and can take place quickly, while others may require us to delete and recreate the AWS resources, which can be time-consuming. In the event of replacing any underlying resources, web application downtime might result. Elastic Beanstalk will warn you about this before finally reflecting the configuration changes. For example, in the following steps, we are modifying the minimum number of EC2 instances from one to two in the autoscaling group. Additionally, we will verify that the same changes have been reflected in the existing environment. The steps for changing the configuration are as follows:

1. Go to the Amazon Elastic Beanstalk console.
2. Click on the `getting-started` web application, and then click on `GettingStarted-env`.

3. Go to the **Configuration** tab.
4. Select **Scaling** by clicking on the setting icon in the top-right-hand corner, as shown in the following screenshot:

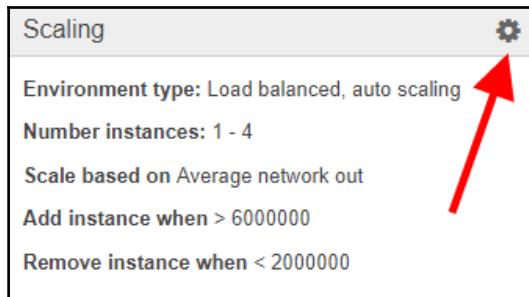


Figure 16.16: Modifying the Scaling configuration

5. Under the **Auto Scaling** section, modify **Minimum instance count** from **1** to **2**, as shown in the following screenshot:

A screenshot of the "Auto Scaling" configuration section. It includes the following fields:

- Minimum instance count: Minimum number of instances to run. A red arrow points to this field.
- Maximum instance count: Maximum number of instances to run.
- Availability Zones: Number of Availability Zones to run in.
- Custom Availability Zones: Specific Availability Zones to launch instances in.
- Scaling cooldown (seconds): The amount of time after a scaling activity before any further trigger-related scaling activities can occur.

Figure 16.17: Auto Scaling and the Minimum instance count

6. At the bottom of the configuration page, click on the **Apply** command button.

Once the configuration has been changed and applied, it may take several minutes to get reflected in the application stack. The time taken to reflect the changes depends on the configuration modification type.

Verifying the changes on the load balancer

After following the steps to change the configuration in the previous section, the changes on the load balancer can be verified using the following steps:

1. Go to the **Elastic Beanstalk** dashboard.
2. Select the getting-started application and the `GettingStarted-env` environment.
3. In the left-hand pane, select **Events**. After some time, there will be an update regarding recent configuration changes in the autoscaling group. The update should have successfully deployed the new configuration to the environment.
4. Go to the EC2 console.
5. Select **Load Balancers** under **LOAD BALANCING** in the left-hand pane.
6. Identify the load balancer, which is part of the web application deployed to Elastic Beanstalk.
7. In the lower pane, under the **Instances** tab, we can clearly see that two EC2 instances are part of the same ELB, as shown in the following screenshot:

Instance ID	Name	Availability Zone	Status	Actions
i-07acc1cd261eb70f9	GettingStarted-env	us-east-1a	InService	Remove from Load Balancer
i-08469c88a1b402474	GettingStarted-env	us-east-1b	InService	Remove from Load Balancer

Figure 16.18: The ELB under the EC2 console showing the modified minimum number of instances

Step 6 – cleaning up

It is essential to delete all unwanted versions, environments, and applications from Amazon Elastic Beanstalk to avoid incurring unwanted charges to the AWS account.

Therefore, with these six steps, we have seen how easy it is to create, view, deploy, update, and terminate an Elastic Beanstalk web application.

In the next section, we will gain an understanding of the version life cycle of an Elastic Beanstalk application.

The version life cycle

Elastic Beanstalk creates a newer application version upon uploading a newer source code bundle. Creating a newer version and not deleting the old unwanted application version leads to hitting the application version limit. As a result, it does not allow us to create any newer web application versions.

The default Elastic Beanstalk limits are as follows:

Resource	Default limit
Applications	75
Application versions	1,000
Environments	200

With the help of the application version life cycle policy for an application, hitting an application version limit can be avoided. Consequently, it will manage the number of available application versions at any given time. Once the life cycle policy is enabled, it will keep either the total count of recent versions (that is, the last 200 versions of the application) or the versions that are not older than the specified age in terms of days (that is, 180 days). The application life cycle can be configured using the web console, CLI, or API.

Deploying web applications to Elastic Beanstalk environments

Elastic Beanstalk allows you to create a new web application, update an environment with a newer web application version, or redeploy an earlier version of the application. Deploying a new web application is very quick. It creates an underlying required AWS resource and deploys a web application. It is important to understand the following points while deploying an application to Elastic Beanstalk:

- While updating a newer version of a web application, you can perform in-place updates. This means that the already deployed web application will be updated with a newer source bundle.
- Redeploying the application may result in restarting the web container or the web application server. As a result, the web application may be unavailable for a while during the deployment process.
- When an environment has more than one EC2 instances, a newer version of a web application can be deployed in a rolling manner. This helps to avoid total unavailability of a web application. It will deploy a newer version of a web application in instance batches.
- With the help of immutable updates, it is possible to always deploy a newer version of a web application to new instances rather than updating an existing instance. In this scenario, Elastic Beanstalk will create a new autoscaling group, and the newly created autoscaling group will serve traffic to the earlier created instances until the newly created instances pass the health checks.
- In a normal scenario, during the deployment of a newer version of a web application, Elastic Beanstalk performs an in-place update. As a result, the web application is unavailable until the deployment of a newer version completes. Blue-green deployment is a solution to avoid such unavailability; in this development method, a new infrastructure is created and, upon successful deployment of a newer version on a newer infrastructure, it just changes the CNAMEs of the old environment to the new environment to redirect traffic to the new version instantly.

Details of the various deployment methods can be obtained from the following table:

Deployment policy	Impact of failed deployment	Deploy time	Zero downtime	No DNS change	Rollback process	Code deployed to
All at once	Leads to web application downtime	Least	X	✓	Redeploy	Existing instances
Rolling	Deploys a new web application version in an instance batch	Moderate	✓	✓	Redeploy	Existing instances
Rolling with an additional batch	Offers minimal impact if the first batch fails, otherwise, it is similar to rolling deployment	Moderate to higher	✓	✓	Redeploy	New and existing instances
Immutable	Minimal	High	✓	✓	Redeploy	New instances
Blue-green	Minimal	High	✓	X	Swap the URL	New instances



The actual deployment time for rolling and rolling with an additional batch depends on the batch size.

Monitoring the web application environment

Once a web application is successfully deployed, it is very important to monitor its performance. Monitoring helps to find out whether there is any infrastructure bottleneck, performance issue, or underutilization of resources. The Elastic Beanstalk web console gives a high-level overview in terms of monitoring figures and graphs, as shown in the following screenshot:

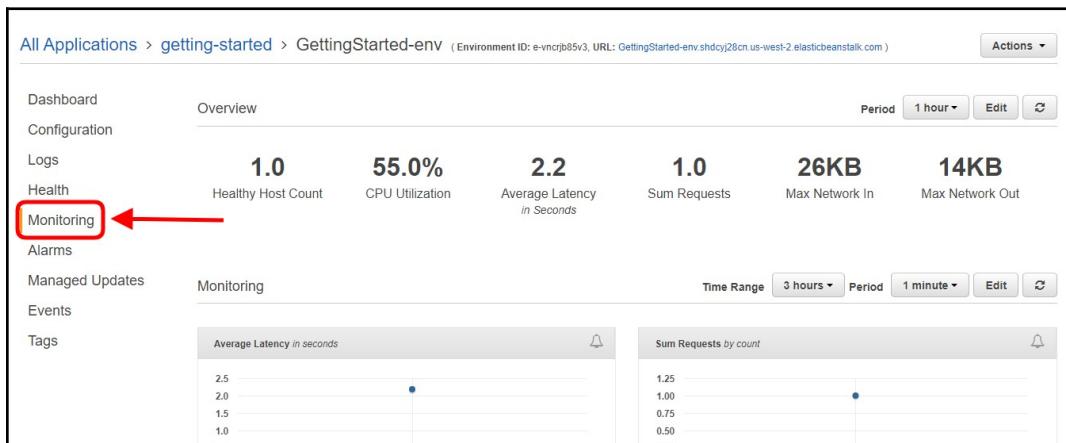


Figure 16.19: The Monitoring dashboard

Elastic Beanstalk also offers various other methods to monitor a deployed web application, such as basic health reporting, enhanced health reporting and monitoring, managing alarms, the Elastic Beanstalk environment's event stream, listing and connecting to server instances, and viewing logs from the Elastic Beanstalk environment's EC2 instances.



A detailed understanding of the various monitoring methods can be obtained from <https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/environments-health.html>.

Elastic Beanstalk best practices

Web application deployment on Elastic Beanstalk ultimately uses AWS services, such as EC2, ELB, ASG, SQS, S3, and many others. Points such as scalability, security, persistent storage, fault tolerance, content delivery, software updates, patching, and connectivity should be kept in mind when designing applications to deploy on AWS Elastic Beanstalk. The following list describes some of the best practices that you should follow while working with Elastic Beanstalk:

- Web applications should be as stateless as possible, fault tolerant, and loosely coupled to efficiently scale out and scale in as the end user's request increases and decreases, respectively.
- On AWS, security is a shared responsibility. AWS is responsible for providing the necessary physical resources (as and when) to make the cloud a safe place to deploy our applications, and we, as cloud users, are responsible for the security of the data traversing through the Elastic Beanstalk environment and the security of the application.
- Configure the SSL certificate to encrypt sensitive information transmission over a public network, such as the internet.
- Elastic Beanstalk deploys an application on an Amazon EC2 instance that does not have persistent storage. Applications should be designed to store data in a persistent data source, such as Amazon S3, EBS, Amazon DynamoDB, and Amazon RDS.
- It is highly recommended to design an application that is capable of automatically recovering from failure. Deploy an application and database across multi-AZs. For a mission-critical application and database for the enterprise, also design the DR site.
- End users may be accessing web applications from across the globe and various networks. They may experience poor performance due to higher latency. Amazon CloudFront can be used to avoid such poor user experience.
- Periodically, Elastic Beanstalk updates its underlined platform configuration with the latest software and patches. AWS recommends that you upgrade your application environments to the latest software and patches. These environment updates can be applied to all servers at once, or you can apply them in smaller batches so that your entire application environment does not go down.

Summary

- AWS Elastic Beanstalk is an orchestration service provided by AWS. It can deploy applications that orchestrate a number of AWS services, such as EC2 instances, S3 buckets and objects, CloudWatch, SNS, ELB, and autoscaling.
- Most of the deployment and infrastructure tasks, such as uploading a newer version of a web application and changing the size of the Amazon EC2 instances, can be done directly from the Elastic Beanstalk web console.
- The web server environment and the worker environment are the two environment tiers supported by Elastic Beanstalk.
- Every Elastic Beanstalk environment carries a CNAME and an alias in Amazon Route 53 pointing to the ELB.
- The Elastic Beanstalk source bundle should be a single ZIP or WAR file.
- To deploy a worker application in a worker tier, the application source bundle must also include the `cron.yaml` file.
- Creating a web application and preparing the source code to deploy a web application varies in each of the supported programming languages, such as Java, .NET, Node.js, PHP, Python, and Ruby.
- The environment dashboard shows the application health, application version, and the application's environment version.
- Elastic Beanstalk has a soft limit of 75 applications. This soft limit can be increased by raising a support ticket with AWS.
- Elastic Beanstalk can carry 1,000 application versions. This is a soft limit and can be increased by raising a support ticket with AWS.
- While updating a newer version of a web application, you can perform in-place updates. Redeploying the application may result in restarting the web container or the web application server.
- When an environment has more than one EC2 instance, a newer version of a web application can be deployed in a rolling manner.
- In a normal scenario, during the deployment of a newer version of a web application, Elastic Beanstalk performs an in-place update.
- Elastic Beanstalk monitoring helps to discover whether there is any infrastructure bottleneck, performance issue, or underutilization of resources.

- There are different deployment policies in Elastic Beanstalk namely *All at once*, *Rolling*, *Rolling with an additional batch*, *Immutable*, and *blue-green deployment*.
- The *All at once* deployment method deploys new versions of applications to all the instances simultaneously. Since the deployment is executed simultaneously, your application goes out of service for a short period of time during the deployment process.
- The *Rolling* update deploys new code/build to instances in batches. Each batch comprises one or more servers. Elastic Beanstalk takes one batch of instances out of service at a time and performs deployment. This prevents complete downtime on your environment. However, your environment operates at a reduced capacity during the deployment process.
- The *Rolling with an additional batch* update performs deployment in batches. Unlike the *Rolling* update, this method first deploys a new batch of instances before taking the old batch of instances out of service. This approach ensures that your environment operates at full capacity even during the deployment process.
- The *Immutable environment* update creates a new temporary autoscaling group behind a load balancer, launches one new instance with the new version of the application, and performs the health checks. If the health checks are successful on one instance, it deploys all additional instances in the new autoscaling group. Subsequently, it transfers all of these instances to the original autoscaling group after performing successful health checks on all the instances.
- In *blue-green deployment*, you deploy the new version of your application in a separate environment, perform health checks/tests, and, subsequently, swap CNAMEs of the old and new environments to redirect traffic to the new application version instantly.

17

Overview of AWS Lambda

In AWS, a compute resource such as AWS EC2 can be used to host small-to-large enterprise applications. Notably, infrastructure configuration complexity in enterprise applications may vary from application to application in different organizations. Handling day-to-day operational activities on these compute resources may be time-consuming and require additional resources to manage tasks. Even if an organization automates tasks, any periodical manual intervention in activity may create a hindrance for the management and maintenance of such resources.

To address this organizational issue, Amazon provides a hosted compute service called **Amazon Lambda**. The sole purpose of the Lambda service is to abstract server management and simplify the building of on-demand applications. Lambda provides an abstract layer for hosting application functions and manages underlined servers in the background. A piece of code can be uploaded in AWS Lambda; this code is known as a **Lambda function**. The Lambda function gets executed against an event on the AWS resource. These events can include a new object being uploaded into an S3 bucket, or new data being added in a DynamoDB table. AWS Lambda supports a wide range of events for a variety of AWS services.

Let's take a look at AWS Lambda in more detail. This chapter discusses the following topics:

- Introducing AWS Lambda
- Environment variables
- Tagging Lambda functions
- Lambda function over VPC
- Building applications with AWS Lambda
- AWS Lambda best practices

Introducing AWS Lambda

AWS Lambda is a serverless and event-driven compute service. It allows you to upload a piece of source code to execute against a valid event. The uploaded piece of code is called a Lambda function. At the time of writing this chapter, AWS Lambda supports Java, Node.js, C#, Ruby, Go, PowerShell, and Python programming languages. In the case of EC2 instances, you are charged for each running second. It is important to note here that, until mid-2017, AWS used to charge on an hourly basis for EC2 instances. In the case of AWS Lambda, charges apply for code runtime in increments of 100 milliseconds. Notably, charges are not applicable to uploading code. AWS does not charge you for just creating and keeping a Lambda function; you are charged for the amount of time it takes a Lambda function to run.

You can create Lambda functions that run on events so that (as with any new object placed in an S3 bucket) a new record is inserted into a DynamoDB table.

Alternatively, you can directly invoke a Lambda function using an API call. It can also execute code at a scheduled time, just like a **cron job** on a Linux server. Lambda functions run on highly available compute resources, such as a balanced CPU, memory, network, and other resources. Lambda automatically scales from a few requests per day to thousands per second, based on the actual load to the Lambda services in real time. AWS Lambda sets IT professionals free from creating an EC2 instance, server maintenance, patching, code monitoring, auto scaling, and more. It also eliminates the need for an operations team in an organization to manage servers.



Lambda does not provide complete control over a base OS such as EC2 instances. It is not possible to log in to a Lambda instance to customize the OS-level parameters. On the other hand, AWS Lambda supports the secured execution of native Linux executables by calling out from a supported runtime.

Understanding a Lambda function

AWS Lambda functions include source code along with all dependencies. Each Lambda function has its own configuration information, such as runtime, environment variables, handler method, IAM roles, tags, memory, timeout, VPC, and many other details that are defined at the time of creation.



The amount of memory that can be allocated to a Lambda function ranges between 128 and 1,536 MB. The minimum memory that can be allotted to a Lambda function is 128 MB. Based on the requirement, you can add more memory in increments of 64 MB. Depending on the allotted memory, AWS automatically allots proportional CPU power and other resources for executing the specific Lambda function. In the background, AWS uses the general-purpose instance type for a Lambda service, which is opaque to end users.

A Lambda function can be configured to execute within 1 to 900 seconds. Until mid-October 2018, this was only 1 to 300 seconds. The Lambda function execution time is called a **timeout**. If a Lambda function is running after the defined timeout, it is automatically terminated. The default configured timeout for a Lambda function is three seconds.

Source code can be written using code-authoring tools and editors. When creating a Lambda function using a web console, it automatically prepares a deployment package before uploading the source code. The steps to create a deployment package vary from one supported programming language to another. At the time of creating a Lambda function, you can configure the amount of memory that you need to execute the function. AWS automatically allocates proportional CPU power and other resources to execute a specific Lambda function. For example, a Lambda function with a memory space of 256 MB may be allocated twice as much CPU as a Lambda function with an allocated memory space of 128 MB.

Lambda is widely used by many organizations for deploying small-to-large scale applications. These applications range from small, event-driven applications to large, big data architectures for implementing stateless infrastructure.

When creating a Lambda function along with memory, here are a few more parameters that need to be defined:

- **Maximum execution time (timeout):** The maximum it can be is 15 minutes. This helps to prevent the Lambda function from running indefinitely. When timeout has been reached, the Lambda function execution terminates.
- **The IAM role (or execution role):** Lambda can assume an IAM role at the time of execution. Based on the privileges granted to the IAM role, the Lambda function can inherit the privileges for executing the function.

- **The handler name:** This refers to the method name to be used by AWS Lambda to start the execution. AWS Lambda passes on event information that triggers the invocation as a parameter to the handler method.



For testing purposes, Lambda functions can be invoked on-demand from the AWS Lambda console or by using the CLI.

To troubleshoot the AWS Lambda function, AWS Lambda automatically monitors functions through CloudWatch metrics. This provides several metrics, such as **Invocation count**, **Invocation duration**, **Invocation errors**, **Throttled invocations**, **Iterator age**, and **DLQ errors**. To manually validate the behavior of the code, it is possible to insert logging statements. Lambda will automatically push all logs from the code to CloudWatch Log groups.

The Lambda function invocation types

AWS Lambda supports two invocation methods: **synchronous** and **asynchronous**. The invocation type can only be specified at the time of manually executing a Lambda function. This Lambda function execution is called **on-demand invocation**. Two examples of this include calling a Lambda function from a custom application or manually executing it using a CLI/GUI.

On-demand invocation is done through the **invoke operation**. This allows you to specify the invocation type as synchronous or asynchronous. When a Lambda function is triggered using an AWS service as an event source, the invocation type is predetermined for each of the AWS services and it can't be changed. For example, Amazon S3 invokes a Lambda function non-concurrently and Amazon Cognito always invokes a Lambda to work synchronously.

The synchronous method blocks a thread's execution until the client receives a response from the Lambda function. On the other hand, the asynchronous method returns control immediately, giving it back to the calling thread without waiting for a response from the Lambda function.

Writing a Lambda function

At the time of writing this book, AWS Lambda natively supports Node.js, Java, Python, Go, PowerShell, Ruby, and C# for writing Lambda functions. Irrespective of the programming language used to write the AWS Lambda function, there is a common pattern to writing the code, including the following core concepts:

- The handler
- The context object
- Logging
- Exceptions

A handler needs to be specified at the time a Lambda function is defined. It points to a function in the source code. Upon the occurrence of a configured event to execute, the Lambda function starts to execute the code by calling the handler function defined in the code. At the same time, AWS Lambda passes event data to the handler function as a first parameter. It is recommended that you write a handler function that is capable of processing the incoming parameters. Based on the supplied parameters, the function may invoke any other subsequent function or method.

The context object, logging, and exceptions are explained in subsequent sections.



It is possible to write Lambda function logic directly in a handler function; however, it is recommended that you write separate subfunctions to incorporate core logic from the handler.

A Lambda function handler in Node.js

The following syntax can be used to write a handler function in Node.js:

```
exports.myHandler = function(event, context, callback) {  
  ...  
}
```

The `callback` parameter is optional, based on whether it is required to return information to the caller:

```
exports.myHandler = function(event, context, callback) {  
  ...  
  
  // Use callback() and return information to the caller.  
}
```

Here is a list of components that are part of the syntax:

- `event`: This parameter is used by AWS Lambda to pass event data to the handler function.
- `context`: This parameter is used by AWS Lambda to provide runtime information from an executing Lambda function to the handler function, such as the remaining execution time before AWS Lambda terminates.
- `callback`: Optionally, a `callback` function is used to return information from the executing Lambda function to the caller. Otherwise, by default, it returns a null value.
- `myHandler`: This is a function name and acts as an entry point for invoking the Lambda function.



Node.js v0.10.42 supports context methods (including `done()`, `succeed()`, and `fail()`) to gracefully complete the Lambda function execution; note that Node.js v6.10 and v4.3 both support callback functions.

A Lambda function handler in Java

The following syntax can be used to write a handler function in Java:

```
outputType handler-name(inputType input, Context context) {  
    ...  
}
```

Here is a list of components that are part of the syntax:

- `inputType`: This can be event data or custom input that is provided as a string or any custom data object. The Lambda function must be invoked with the input data for the handler to be successfully invoked. This input data can be serialized into the defined data type of the input parameter.
- `outputType`: When the Lambda function is invoked synchronously using the `RequestResponse` invocation type, it is possible to return the output of the Lambda function using a valid and supported data type.

If you invoke the Lambda function asynchronously with the help of the event invocation type, the `outputType` component should be `void`. For example, event sources such as Kinesis, Amazon S3, and Amazon SNS can invoke an asynchronous Lambda function using `event`. The `inputType` and `outputType` components can be one of the following:

- **Predefined AWS event types:** These are defined in the `aws-lambda-java-events` library.
- The **Plain Old Java Object (POJO)** class: This allows you to create your own POJO class. The Lambda function automatically serializes and deserializes input and output on the POJO type or JSON.
- **Primitive Java types:** These support primitive Java types such as `String` and `int`.

If the handler function is overloaded (that is, it has multiple methods with the same name), then the handler method selection will take place based on the following rules:

- The method with the largest number of parameters is selected.
- When two or more methods in a Lambda function have the same number of arguments, the method that has `context` as the last parameter is selected.

If the overloaded handler methods do not have a `context` parameter, the behavior is unpredictable.



A Lambda function handler in Python

The following syntax can be used to write a handler function in Python:

```
def handler_name(event, context):  
    ...  
    return some_value
```

Here is a list of components that are part of the syntax:

- `event`: AWS Lambda passes event data to the handler with this parameter. It is usually of the Python `dict` type.
- `context`: AWS Lambda passes the runtime information of the executing Lambda function. It is of the `LambdaContext` type.

Optionally, the handler can return values, based on the invocation type, to invoke the Lambda function in either of the following ways:

- In a synchronous execution using the RequestResponse invocation type, the Lambda function returns the result of a Python function call.
- In an asynchronous execution using the Event invocation type, the return value is discarded.

A Lambda function handler in C#

You can write a Lambda function handler in C# as a static method in a class or as an instance. You can access a Lambda context object using the definition of the ILambdaContext type, as given in the following syntax:

```
returnType handler-name(inputType input, ILambdaContext context) {  
    ...  
}
```

Here is a list of components that are part of the syntax:

- `inputType`: This can be event data or custom input, provided by an event source or string, respectively.
- `outputType`: When a Lambda function is invoked synchronously using RequestResponse, it is possible to return the output of the Lambda function using any of the valid and supported data types.

When invoking the Lambda function asynchronously with the help of the Event invocation type, `outputType` should be void. For example, event sources such as Kinesis, Amazon S3, and Amazon SNS invoke the Lambda function asynchronously using the Event invocation type.

Now that we understand the handler function, let's take a brief look at the other core concepts in AWS Lambda, which include the `context` object, logging, and exceptions:

- **The context object and how it interacts with Lambda at runtime:** As a second parameter, AWS Lambda passes a `context` object to the handler function. It can help the source code to interact with the AWS Lambda runtime. It can also help to find the remaining execution time before AWS Lambda terminates. In addition to this, asynchronous programming languages, such as Node.js, use callbacks. These support additional methods on the `context` object method to tell AWS Lambda to terminate the function and, optionally, return values to the caller.
- **Logging:** AWS Lambda functions can have a logging statement, depending on the programming language used to write the source code. These logs are written to CloudWatch Logs.
- **Exceptions:** AWS Lambda functions need to return the invocation state in terms of success or failure. Exceptions may be raised at the Lambda function initialization or function invocation. If there is an exception at the function invocation, then each supported programming language has different ways to return the success or the failure of the Lambda function execution. With synchronous execution, AWS Lambda will forward the result back to the client.

Deploying a Lambda function

A Lambda function can be created using the inline editor provided in the Lambda function dashboard; otherwise, you can create a function in your local machine and deploy it on Lambda. When you create a Lambda function in your local machine, you need to create a package that can be uploaded to Lambda for deployment. AWS Lambda function configuration requires a source code in a `.zip` or `.jar` file, consisting of the source code along with dependencies. Creating a deployment package varies for each supported programming language. For example, in Node.js, you simply zip up the contents of the directory along with all of the dependencies in a ZIP file and upload it to Lambda for deployment.

As shown in *Figure 17.1*, you just need to zip up all of the files in `MyFunctionFolder`, which includes the `index.js` and `node_modules` folders. Remember, you should not zip up `MyFunctionFolder` itself. The ZIP should contain only the `index.js` folder and all of the dependencies, but not its parent folder:

Name	Date modified	Type	Size
node_modules	23-09-2017 22:03	File folder	
index.js	23-09-2017 22:07	JavaScript File	27 KB

Figure 17.1: The deployment package

To create a Lambda function, it is essential to first create a deployment package. A deployment package can be created in two ways:

- Preparing a deployment package manually
- Writing code directly in a Lambda console, which will create a deployment package for you



Once the source code is converted into a deployment package, it can either be directly uploaded to Lambda from the local machine or first to an Amazon S3 bucket. The S3 bucket should be in the same region as where the Lambda function is being created. You should specify the same S3 link URL when creating a Lambda function.

AWS Lambda function versioning and aliases

Lambda function versioning makes it possible to publish one or more versions of the function code. This makes it possible to use different function versions in different environments, such as development, testing, preproduction, or production. Each Lambda function has a unique and immutable **Amazon Resource Name (ARN)**.

It is possible to create an alias for each Lambda function version. Each alias is a pointer to the specific Lambda function version. Just like Lambda functions, an alias is a resource and has a unique ARN for a function version to which it points. Aliases are mutable, that is, they can be modified to point to a different version of the Lambda function.



Note that an alias can only point to a function version; it cannot point to another alias.

Aliases primarily help to abstract the process of promoting the Lambda function version from one environment to another. To simplify this, we can use the Amazon S3 bucket as an example event source; when some new objects are created in a bucket, the Lambda function gets invoked. For triggering an event in the S3 bucket, you need to configure the Lambda function ARN in the bucket notification configuration. The configured Lambda function is invoked on a new object event in the bucket. Usually, every time a new version of a Lambda function is created, a function with a new ARN is generated. A new ARN for the function forces you to modify the bucket configuration where the event is configured. To avoid repeated S3 bucket configuration modifications, it is recommended that you use a Lambda function alias. The Lambda function alias can point to the ARN of any Lambda function.

The ARN of the Lambda function alias remains intact, that is, you don't need to change it. You can configure this alias ARN in the S3 bucket for triggering the event. Every time you create a new version of the Lambda function, you just need to update your alias pointing to the ARN of the newly created Lambda function. As a new version of a Lambda function is promoted from one environment to another, we just need to change the alias pointing to the latest stable Lambda function version and do not need to modify the notification configuration in the Amazon S3 bucket.

If you need to roll back a modification from a newer Lambda function version to an older one, this can be easily done by changing where the alias is pointing. This removes the need to update the event source mappings in the S3 bucket.

Environment variables

Reusable and efficient code often requires passing dynamic values at runtime. These runtime values may be environment types, file paths, path to store logs, table names, and more. With the help of environment variables, Lambda functions allow us to pass dynamic values at runtime. As a result, the code becomes reusable without making any changes to it.

Environment variables are key-value pairs and these key-value pairs are encrypted/decrypted using the **AWS Key Management System (KMS)**. Key-value pairs can be defined at the time of creating a Lambda function. Externally configured environment variables are also accessible within the Lambda function, using standard APIs supported by the different programming languages. For example, Node.js functions can access environment variables using `process.env`, where `process.env` refers to an object in Node.js. In Node.js, `process` is the global object and `env` is the sub-object of `process`, which provides all of the environment variables.

The rules for naming environment variables are as follows:

- Lambda functions can have any number of environment variables, but the total size should not exceed 4 KB.
- The first character must be [a-zA-Z]
- Consecutive valid characters can be [a-zA-Z0-9_]

Specific environment variables, such as `MemorySize` and `Timeout`, are saved as a snapshot for each Lambda function version. These settings are immutable.

Tagging Lambda functions

Tagging is useful for segregating and grouping Lambda functions with the help of key-value pairs. A specific set can be easily accessed and analyzed by customers using the tags by filtering multiple Lambda functions that contain the same tag. Tags are key-value pairs. They are associated with a Lambda function to organize them and find other details, such as the frequency of invocation and the cost of each function invocation. Primarily, tags help to group, filter, and allocate cost.

Lambda functions over VPC

Deploying AWS resources inside an Amazon VPC can be helpful when you do not want to expose your Lambda function over the internet or if you want your Lambda function to securely access your VPC resources. By default, the Lambda function is deployed in a non-VPC environment. To enable a Lambda function to access other AWS resources deployed in a private VPC, it is essential to provide details such as VPC, subnets, and security groups at the time of configuring it. VPC details are used by the Lambda function to create an **Elastic Network Interface (ENI)** to connect securely with other private VPCs.



AWS Lambda does not support a connection to resources within dedicated tenancy VPCs.

When a Lambda function requires internet access, rather than deploying it in a public subnet, deploy it in a VPC in a private subnet with a **Network Address Translation (NAT)** gateway or NAT instance in a public subnet.

Building applications with AWS Lambda

In a serverless architecture, the main components are the event source and the Lambda function. The event source can be a custom application or AWS. Each of the AWS event sources uses a specific format for event data. At present, the list of services supported by AWS Lambda as event sources can be found at <https://docs.aws.amazon.com/lambda/latest/dg/invoking-lambda-function.html>. These supported AWS services are broadly divided into **stream-based** services and **regular AWS** services.



Out of all of the supported AWS services, Amazon Kinesis Streams and Amazon DynamoDB streams are the only stream-based AWS services. The rest of the services are regular AWS services.

Apart from the previously mentioned supported AWS services, a custom user application can also create an event to trigger the Lambda function's invocation.

Event source mapping for AWS services

Regular AWS services publish events to invoke a Lambda function; this is also called a **push model**. A push model has the following behavioral characteristics:

- Regular AWS service resources maintain event source mappings with an event source. AWS provides APIs to manage event source mappings. For example, the S3 bucket notification configuration API enables us to configure an event source mapping on a bucket. This configuration mapping identifies the bucket event, which is published to a Lambda function that is configured on the bucket.

- As the event source invokes the Lambda function, it is essential to grant the necessary privileges to the resource, using a resource-based policy. This resource-based policy is referred to as a Lambda function policy.

The following diagram explains how Amazon S3 pushes an event to invoke a Lambda function:

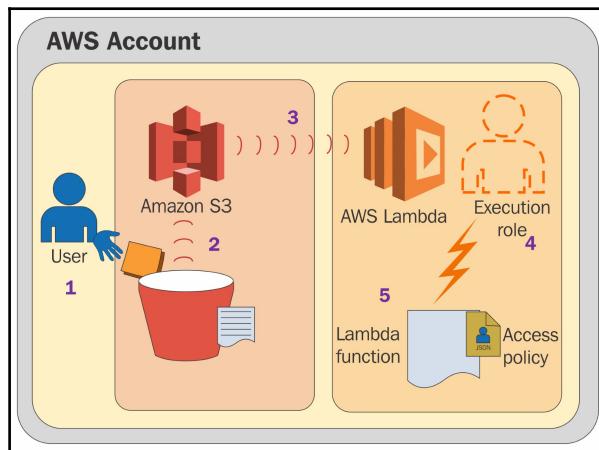


Figure 17.2: How a Lambda function is invoked

In the preceding diagram, S3 pushes an event for each new object created in a bucket to execute a Lambda function as follows:

1. A privileged user creates a new object in an S3 bucket.
2. S3 detects the object creation event.
3. The Lambda function that is configured in the bucket is invoked, according to the event source mapping defined in the bucket notification configuration.
4. An attempt is made by the event source to invoke the configured Lambda function. Immediately after the invocation call, the Lambda function refers to the attached policy to ensure that the Amazon S3 has the necessary permissions.
5. After the successful verification of the attached permission policy on the Lambda function, the Lambda function is executed.

Event source mapping for AWS stream-based services

Figure 17.3 explains the way a custom application writes a record to Kinesis Stream and the way that Lambda polls the stream:

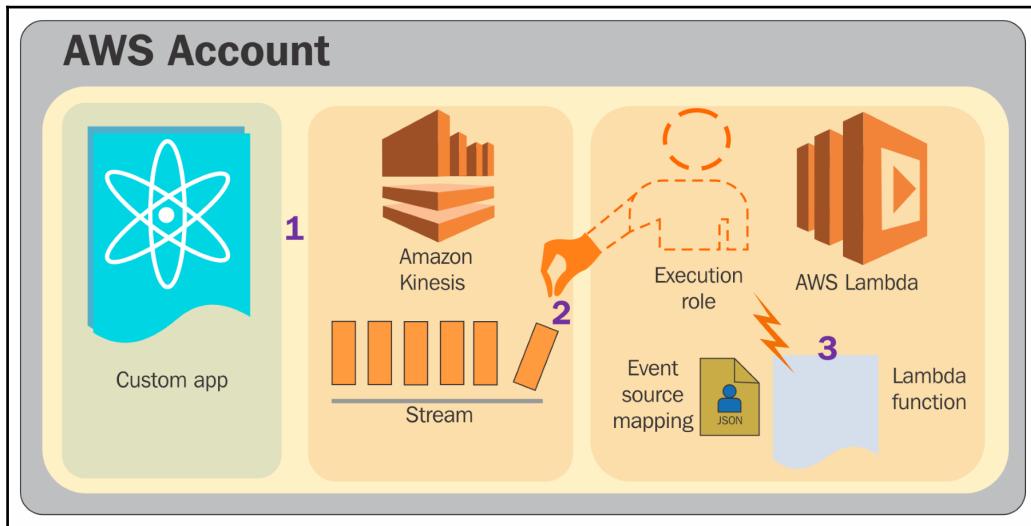


Figure 17.3: Event source mapping for AWS stream-based services

The steps for the preceding diagram are as follows:

1. The custom application writes records to Amazon Kinesis Stream.
2. At the same time, AWS Lambda continuously keeps polling the stream. As soon as it detects a new record on the stream, it invokes the AWS Lambda function. Based on the event configuration, it decides which Lambda function is to execute against which event source.
3. It then verifies that the attached IAM permission policy to the Lambda function allows it to poll the stream. If it is not true, then the AWS Lambda function is not invoked.

Event source mapping for custom applications

Figure 17.4 explains how a custom application deployed in an AWS account can also directly invoke the Lambda function. You create a Lambda function in one of the IAM user accounts and the same credentials will be used to invoke the Lambda function. You do not require additional permissions to invoke the function:

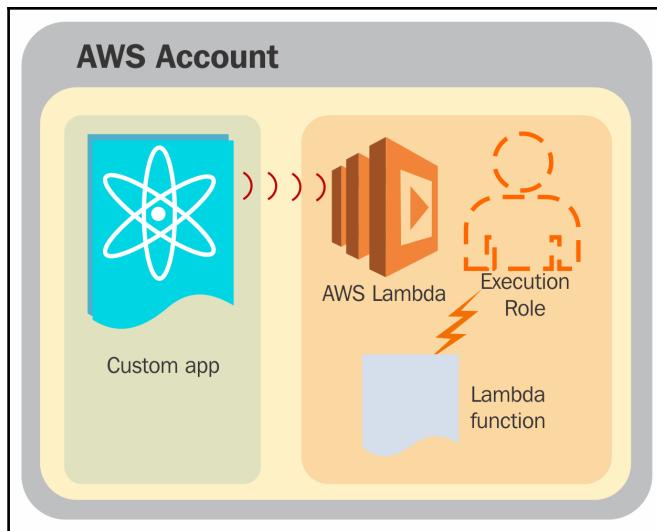


Figure 17.4: The custom application publishes events and invokes a Lambda function

It is also possible to deploy a custom application in **AWS Account A**, and invoke the Lambda function from **AWS Account B**. AWS Account B (that is, where the Lambda function is) must have cross-account permissions in the policy associated with the Lambda function; the following diagram explains this:

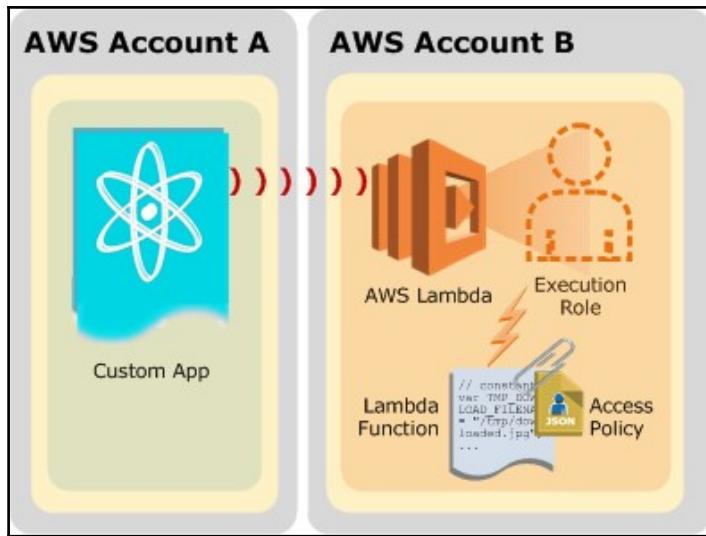


Figure 17.5: Lambda function execution in a cross-account

AWS Lambda best practices

The AWS Lambda best practices are as follows:

- It is best practice to write an AWS Lambda function in a stateless style. It should not have any rapport with the basic compute infrastructure.
- A persistent state should be stored in another cloud service, such as Amazon S3 or DynamoDB.
- It is recommended to separate core logic from the Lambda handler, as the handler is generally used as an entry point to the function.
- When a Lambda function is deployed over a VPC, it is a best practice to avoid using DNS resolution for a public hostname, as it may take several seconds to resolve, adding several billable seconds.
- It is recommended that you specify at least one subnet in each AZ with the Lambda function configuration.
- It is recommended that you make sure that sufficient subnet IPs are free to allow Lambda functions to scale. If there aren't any free subnet IPs, Lambda functions will not scale and the Lambda function failure will increase.

- Rather than reinitializing variables or objects on every invocation, use static initialization or constructor, global, or static variables and singletons. This helps to improve the performance of the Lambda function.
- Where possible, keep alive and reuse connections such as the database or HTTP that were established in an earlier invocation.
- To cope with frequent changes in the operational parameters, pass them using environment variables to avoid frequent changes in a Lambda function.
- It is best practice to pack all of the dependencies in a deployment of the Lambda function. Where possible, control the dependencies in the Lambda function to minimize the overall size and execution time.
- If you are using a Java programming language, put the dependencies in a separate `/lib` directory rather than putting all of the functions and source code in a single JAR with a large number of `.class` files.
- It is highly recommended that you use a simple framework and minimize the complexity of the dependencies to quickly load the container startup.
- It is best practice to use Lambda metrics and CloudWatch alarms to monitor the Lambda function's health rather than creating and maintaining custom metrics from Lambda function code.

Summary

- AWS Lambda is a serverless and event-driven compute service. It allows you to upload a piece of source code to execute against a valid event.
- AWS does not charge you for just creating and keeping a Lambda function. You are charged for the amount of time it takes a Lambda function to run.
- Unlike EC2 instances, Lambda does not provide complete control over the underlying OS.
- The amount of memory that can be allocated to a Lambda function ranges between 128 and 1,536 MB.
- The minimum memory that can be allotted to a Lambda function is 128 MB.
- Depending upon the allotted memory, AWS automatically allots proportional CPU power and other resources for executing the specific Lambda function.
- If a Lambda function is running after the defined timeout, it is automatically terminated.

- You can configure a Lambda function to run for a maximum of 15 minutes per execution.
- Lambda can assume an IAM role at the time of execution.
- The handler name refers to the method name to be used by AWS Lambda to start the execution.
- AWS Lambda supports two invocation methods—synchronous and asynchronous.
- AWS Lambda natively supports Node.js, Java, Python, Go, PowerShell, Ruby, and C# for writing functions.
- A handler needs to be specified at the time a Lambda function is defined.
- A Lambda function starts to execute the code by calling the handler function defined in the code.
- It is possible to write Lambda function logic directly in a handler function; however, it is recommended that you write separate sub-functions to incorporate core logic from a handler.
- To create a Lambda function, it is essential to first create a deployment package.
- A deployment package can be created manually or you can write code directly in the Lambda console and the console creates a deployment package for you.
- Each Lambda function is assigned a unique and immutable ARN.
- Lambda can have multiple versions of a function.
- You can assign an alias to a Lambda function version.
- An alias can only point to a function version. It cannot point to another alias.
- Deploying AWS resources inside an Amazon VPC can be helpful when you do not want to expose your Lambda function over the internet or if you want your Lambda function to securely access your VPC resources.
- It is also possible to deploy a custom application in AWS Account A, and invoke the Lambda function from AWS Account B.

18

Key Management Services

AWS **Key Management Service (KMS)** is a scalable encryption and key management service provided by Amazon. As the name suggests, KMS can be used for encrypting data and managing encryption keys. This chapter introduces you to KMS and explains how you can use it with other AWS services. However, before we get into the details of this chapter, it is important to understand some of the basics of encryption.

We will cover the following topics in this chapter:

- Introducing encryption
- How KMS works
- Types of keys
- Creating a **Customer Master Key (CMK)**
- Updating the administrators or the users of a key
- Tagging a key
- Enabling or disabling keys
- Using KMS with other AWS services

Introducing encryption

Encryption is the process of converting data into a format where only an authorized person is able to access it. Such conversion is called **encoding**. Data is encoded using an encryption key; an encryption key is a random string containing some random characters or bits. The randomness of the characters in a key is achieved by using algorithms to ensure that each key is unique and unpredictable.

There are two popular types of encryption:

- Symmetric encryption
- Asymmetric encryption

Symmetric encryption

Symmetric encryption uses a single key in order to encrypt or decrypt data. It uses a secret key that can be a number, a word, or a string of random characters. This key is used with an encryption algorithm in order to encode the data. Both the sender and the receiver of the data should have the same key to encrypt and decrypt the data. Symmetric encryption uses a number of different types of encryption algorithms, including the **Advanced Encryption Standard (AES)**, **Rivest Cipher 4 (RC4)**, **RC5**, **RC6**, the **Data Encryption Standard (DES)**, and **Blowfish**. Additionally, **AES128**, **AES192**, and **AES256** are widely used encryptions.

The following diagram demonstrates how symmetric encryption functions:

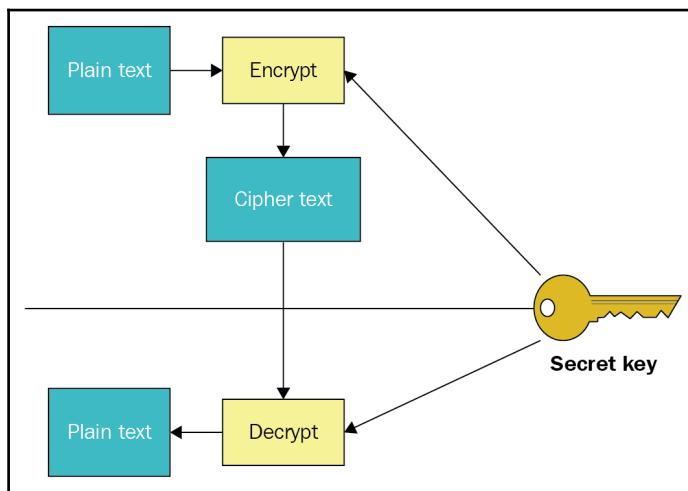


Figure 18.1: Symmetric encryption

Now, let's examine the workings of asymmetric encryption.

Asymmetric encryption

Asymmetric encryption uses a public key and a private key combination in order to encrypt or decrypt data. A public key is used to encrypt a message and this message can only be decrypted by a private key and vice versa. **Public-Key Cryptography Standards (PKCS)**, **Rivest–Shamir–Adleman (RSA)**, and the **Digital Signature Algorithm (DSA)** are some examples of asymmetric encryption algorithms. The following diagram displays the workings of an asymmetric encryption algorithm:

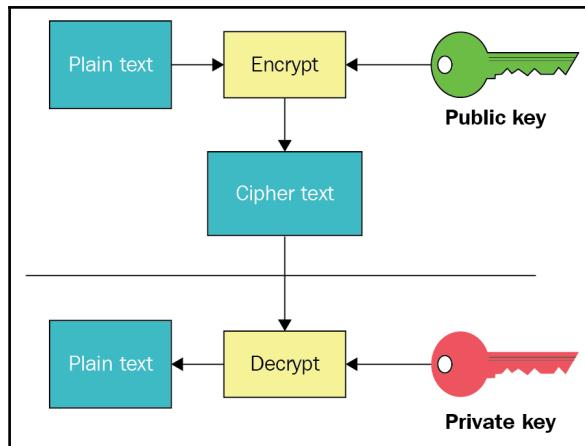


Figure 18.2: Asymmetric encryption

KMS uses a single key to encrypt or decrypt data. This means that KMS supports only the symmetric encryption algorithm. We will explore the workings of KMS in the following section.

How does KMS work?

KMS centrally stores and manages the encryption keys. These encryption keys are called CMKs. KMS can generate CMKs for you. Alternatively, you can use your own key management infrastructure to generate an encryption key and import it into KMS. AWS uses **Hardware Security Modules (HSMs)** in order to secure and protect your keys behind the scenes. KMS can process your request to encrypt or decrypt your data using these master keys. You can set up access policies on these keys and these allow you to determine who can access these keys to encrypt and decrypt the data.

KMS can be easily used with a number of AWS services. It also provides integration with client-side toolkits to encrypt data. KMS uses a method called envelope encryption in order to encrypt data. Here, KMS generates a data key that is used to encrypt the data and this data key is also encrypted using the CMK.

KMS does not store or manage data keys. In fact, when any AWS service encrypts your data using KMS, it also stores an encrypted copy of the data key along with the encrypted data.

When any AWS service needs to decrypt the data, it requests KMS to decrypt the data key using the respective CMK. The user who requests data from the AWS service should be authorized to do so on the CMK's access policy. If the user is authorized, then KMS returns the decrypted data key to the respective AWS service. The service can then use the decrypted data key to decrypt the data and return it in plaintext.

AWS CloudTrail records logs of all of the requests for using the CMKs. Using CloudTrail logs, you can determine who has used the CMK, when they used it, and what the context was for accessing it. The following diagram demonstrates the workings of KMS:

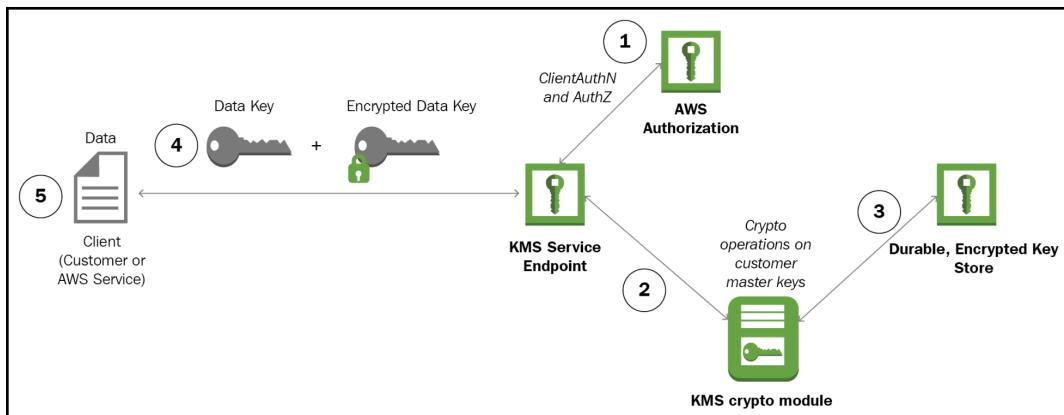


Figure 18.3: How KMS works

When a user, application, or an AWS service tries to access the encrypted data, the following sequence of actions will take place:

1. A user, application, or AWS service makes an authentication request for a data key.
2. Based on the authorization, KMS generates a data key.
3. KMS retrieves the CMK from storage and decrypts it using the **KMS crypto module**.

4. The data Key is encrypted with the CMK and the encrypted data key and plain text data key are returned.
5. The data key is used to encrypt the data and the encrypted data key is stored by the respective service that initiates the request.

Next, we will explore the various types of keys that are available in encryption and study the significance of each.

Types of keys

From the preceding example, you will have learned that there are two types of keys:

- **Data keys:** Data keys are encryption keys that are used to encrypt data. They can be used to encrypt small or large sizes of data or they can even be used to encrypt other keys. As mentioned previously, a data key can be encrypted using a CMK to secure the data key itself.
- **CMKs:** The master key or CMK, is used to encrypt and decrypt the data key. They can be of the following three types:
 - **Customer-managed CMKs:** AWS allows you to create your own CMKs. When you create a CMK, you are also authorized to manage it. The CMKs that you create and manage in your account are called customer-managed CMKs. Here's what you can do with customer-managed CMKs:
 - Create and maintain key policies related to CMKs.
 - Create IAM policies and apply them to CMKs.
 - Manage permissions on the CMKs.
 - Enable and disable the CMKs.
 - Rotate the keys.
 - Add tags.
 - Create aliases for the CMKs.
 - Schedule the CMKs for deletion.

- Use the CMKs in cryptographic operations.
- Audit the usage of the CMKs in CloudTrail logs.
- Use CMKs in AWS services to protect your data that is managed by the respective services.

You are charged a monthly fee for customer-managed CMKs if your usage exceeds the free tier limits for KMS.

- **AWS-managed CMKs:** Apart from allowing you to create your own CMKs, AWS also creates and manages CMKs that are used in AWS services where KMS is supported. AWS-managed CMKs have different aliases than the customer-managed CMKs. AWS-managed CMKs aliases start with `aws/service-name`; for example, the CMK that is stored in Redshift starts with `aws/redshift`.

You can use the AWS-managed CMKs from the respective AWS services; however, you can't manage these CMKs. You can view the keys, view their key policies, and analyze and audit their CloudTrail logs. AWS does not allow you to change these CMK permissions. Additionally, you can't use these CMKs in any other cryptographic operations directly. They can be used within the AWS services where they are created and managed by AWS.

In order to view the key policies that are associated with the AWS-managed CMKs, you can use the `GetKeyPolicy` operation. AWS does not allow you to view or change the policies associated with AWS-managed CMKs from the AWS console.

AWS does not charge you any fees for having these AWS-managed CMKs on your account. However, you are charged if your usage exceeds the free tier limit. Some AWS services will cover the usage cost for you, which means that you don't need to pay for using these keys.

- **AWS-owned CMKs:** Apart from customer-managed CMKs and AWS-managed CMKs, AWS also owns a number of CMKs, called AWS-owned CMKs. These CMKs are used in multiple AWS accounts for protecting customer data. You can't view, use, manage, or audit the usage of AWS-owned CMKs. These CMKs don't belong to your account but are shared across multiple AWS accounts. AWS services can implicitly use these CMKs for protecting your data. You are not charged a monthly fee or a usage fee for AWS-owned CMKs and they don't count against the AWS KMS limits of your account.

Different types of CMKs

Here's a quick comparison between customer-managed, AWS-managed, and AWS-owned CMKs:

Description	Customer-managed CMKs	AWS-managed CMKs	AWS-owned CMKs
Key creation	Keys are generated by customers.	Keys are generated by AWS on behalf of the customer.	Keys are generated by AWS to be used in multiple accounts.
Key usage	The customer can control key usage through the KMS and IAM policy.	This can be used only with specific AWS services where KMS is supported.	This is implicitly used by AWS to protect customer data; customers can't explicitly use it.
Key rotation	The customer can choose to rotate the keys once a year or manually rotate the keys.	Keys are rotated automatically once a year.	Keys are automatically rotated by AWS without any explicit mention of the rotation schedule.
Key deletion	This can be deleted.	This can't be deleted.	This can't be deleted.
User access	This is controlled by the IAM policy.	This is controlled by the IAM policy.	This can't be accessed by users.

Key access policy	This is managed by the customer.	This is managed by AWS.	N/A.
-------------------	----------------------------------	-------------------------	------

Creating a CMK

You can create a CMK by using the AWS console and the KMS API. The following steps describe the process for creating a CMK using the AWS console:

1. Log in to your AWS account and navigate to the KMS console at <https://console.aws.amazon.com/kms>.
2. KMS is region-specific service; this means that a key that is created in one region can't be used in another region. Ensure that you have selected the right region and then click on **Customer managed keys**, as shown in the following screenshot:



Figure 18.4: Creating Customer managed keys

3. From the subsequent screen, click on the **Create key** button.
4. After clicking on the button, it displays a screen, as shown in *Figure 18.5*. You can enter the alias name and description of the key as required and click on the **Next** button:

KMS > Customer managed keys > Create key

Add alias and description

Step 1 of 5

Create alias and description

Enter an alias and a description for this key. You can change the properties of the key at any time. [Learn more](#)

Alias

Description

► Advanced options

Cancel **Next**

Figure 18.5: Creating a key alias and description



Remember that you can't use the `aws` prefix in a CMK; it is reserved for AWS-managed keys. Also note that a description is optional. It is recommended that you give an alias name and description that is relevant to the usage of the key.

5. You can add a tag key and tag value in the subsequent screen if required; this is optional. Then, click on **Next** to move on to the next screen.

6. Select the administrators or users from the subsequent screen (as demonstrated in *Figure 18.6*). This screen displays a list of IAM users and you can choose a user ID who can administrate this key. There can be more users listed in this screen, depending on the number of users created in your IAM user list. You can check or uncheck the **Allow key administrator to delete this key** option at the bottom of the screen, as shown in the following screenshot:

The screenshot shows the 'Define key administrative permissions' dialog box. At the top, it says 'Key administrators'. Below that, there's a note: 'Choose the IAM users and roles who can administer this key through the KMS API. You may need to add additional permissions for the users or roles to administer this key from this console.' A 'Learn more' link is provided. A search bar and a page navigation bar (with page 1) are also present.

	Name	Path	Type
<input type="checkbox"/>	bhavin	/	User
<input checked="" type="checkbox"/>	vipul	/	User
<input type="checkbox"/>	AWSServiceRoleForSupport	/aws-service-role/support.amazonaws.com/	Role
<input type="checkbox"/>	AWSServiceRoleForTrustedAdvisor	/aws-service-role/trustedadvisor.amazonaws.com/	Role
<input type="checkbox"/>	lambda_basic_execution	/	Role

At the bottom, there's a 'Key deletion' section with a checked checkbox for 'Allow key administrators to delete this key'. At the very bottom, there are 'Cancel', 'Previous', and 'Next' buttons.

Figure 18.6: Define key administrative permissions

7. **Define key usage permissions** appears in the following screen. This displays a list of IAM users and you will need to choose which users you want to give usage permissions to:

Define key usage permissions

This account

Select the IAM users and roles that can use the CMK to encrypt and decrypt data with the AWS KMS API. [Learn more](#)

	Name	Path	Type
<input checked="" type="checkbox"/>	bhavin	/	User
<input type="checkbox"/>	vipul	/	User
<input type="checkbox"/>	AWSServiceRoleForSupport	/aws-service-role/support.amazonaws.com/	Role
<input type="checkbox"/>	AWSServiceRoleForTrustedAdvisor	/aws-service-role/trustedadvisor.amazonaws.com/	Role
<input type="checkbox"/>	lambda_basic_execution	/	Role

Other AWS accounts

Specify the AWS accounts that can use this key. Administrators of the accounts you specify are responsible for managing the permissions that allow their IAM users and roles to use this key. [Learn more](#)

[Add another AWS account](#)

[Cancel](#) [Previous](#) [Next](#)

Figure 18.7: Define key usage permissions

8. In the final step, you can review and edit the policy, as required:

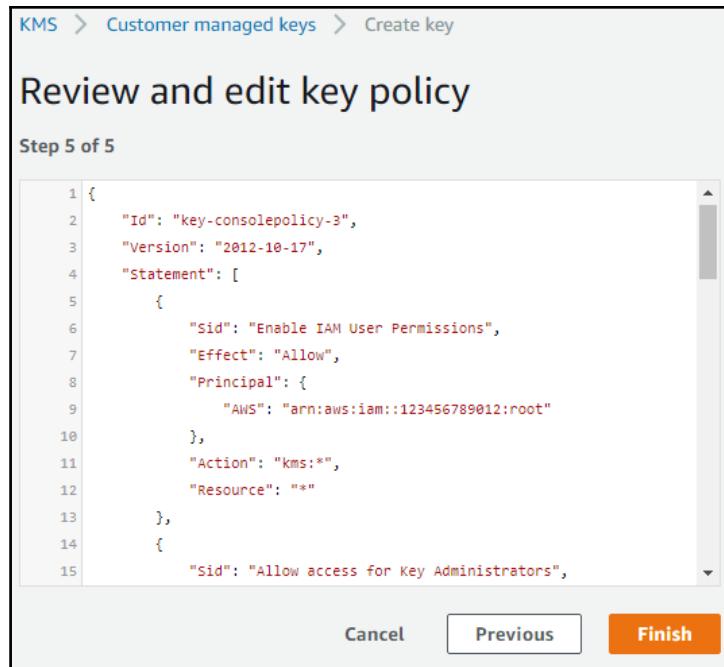


Figure 18.8: Review and edit key policy

9. When you click on the **Finish** button in the final screen, you will see a **Success** message, and the CMK is generated, based on the input you provided:

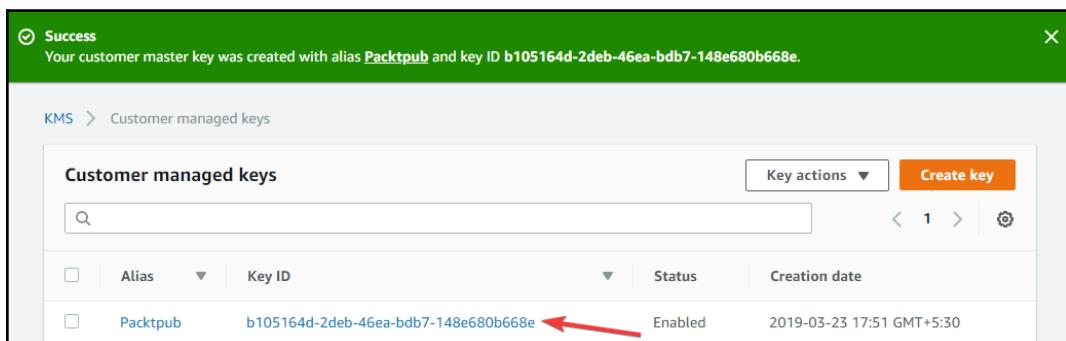


Figure 18.9: The CMK creation Success message

This way, the customer-managed CMK is created in your account and in the specific AWS region that you selected during the process.

Viewing existing keys

You can view existing keys by using the AWS console and the KMS API. The following steps describe the process of viewing existing keys using the AWS console:

1. Repeat steps 1 and 2 from the *Creating a CMK* section.
2. You can view the AWS-managed keys by clicking on **AWS managed keys** in the left panel. The following screenshot shows the left-panel view:

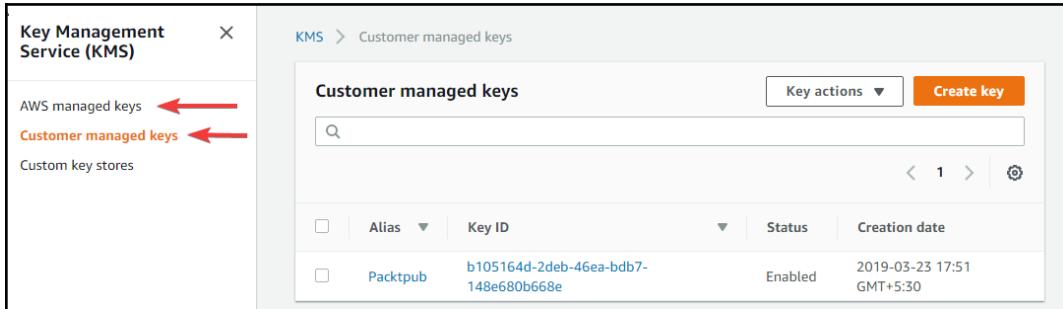


Figure 18.10: Viewing existing keys

Modifying existing CMKs

AWS allows you to modify the properties of customer-managed CMKs using either the console or KMS API. Remember, you can't modify AWS-managed keys. The following steps describe the process of modifying an existing CMK using the console:

1. Repeat steps 1 and 2 from the *Creating a CMK* section.
2. Click on a key that you want to modify.
3. From the subsequent screen, you can change the description of the key and the administrators or users to whom you have granted permission. You can also change the key deletion option and edit the details of another account. It also allows you to edit the policy manually from the policy view. You can refer to *Figure 18.11* in the next section, which displays the key administrators and the key users modification screens.

Updating the administrators or users of a key

The following steps describe the process of updating the administrators or users of a key:

1. If you want to add a key administrator, you can select the **Add** button. In the subsequent screen, you can choose or type in a user or role, and then click on the **Add** button again to finish adding the user as a key administrator.
2. If you want to remove a key administrator, check the box against the user or role and click on the **Remove** button:

The screenshot shows the 'Key policy' section of the AWS KMS console. It is divided into two main sections: 'Key administrators' and 'Key users'.
Key administrators: This section allows you to manage users and roles that can administer the key through the KMS API. It includes an 'Add' button (with a red arrow pointing to it), a 'Remove' button, and a search bar. A table lists one administrator: 'vipul' (User, Path '/').
Key deletion: A checkbox labeled 'Allow key administrators to delete this key' is checked.
Key users: This section lists users and roles who can encrypt and decrypt data from within applications. It includes an 'Add' button (with a red arrow pointing to it) and a 'Remove' button. A table lists one user: 'bhavin' (User, Path '/').
A red arrow also points to the 'Switch to policy view' button in the top right corner.

Figure 18.11: Modifying key administrators and key users

Tagging a key

You can tag existing keys using the AWS console and the KMS API. The following steps describe the process of tagging existing keys using the AWS console:

1. Repeat steps 1 and 2 from the *Creating a CMK* section.
2. You can either click on the key you want to add a tag to or select the checkbox provided next to the key. Then, from the **Key actions** menu, select **Add or edit tags**, as shown in the following screenshot:

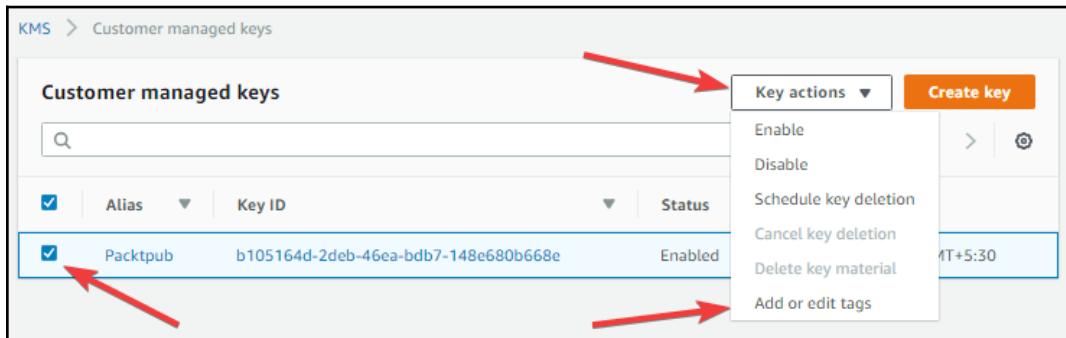


Figure 18.12: Add or edit tags

3. In the preceding screen, you can add or edit the tags as required.

Enabling or disabling keys

You can enable or disable a key by using either the AWS console or the KMS API. The following steps describe the process of enabling or disabling a key using the AWS console:

1. Repeat steps 1 and 2 from the *Creating a CMK* section.

2. Click on the key that you want to enable or disable and then click on the **Key actions** menu. By selecting **Enable** or **Disable** from the **Key actions** menu, you can enable or disable the key:

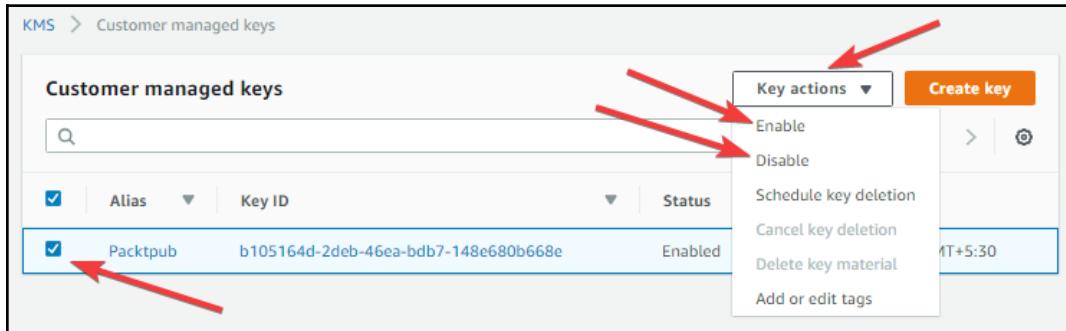


Figure 18.13: Enabling or disabling a key

AWS services supported by KMS

Many AWS services support KMS and this allows you to encrypt your data using the CMKs that you create in KMS or with AWS-managed keys.

The list of AWS services that support KMS is very long. However, some important services from a developer perspective include EBS, RDS, SNS, SES, CodeBuild, CodeCommit, CodeDeploy, CodePipeline, S3, Glacier, Lambda, Redshift, Kinesis, DynamoDB, SQS, and CloudTrail.



You can find the complete list of services supported by KMS at <https://docs.aws.amazon.com/kms/latest/developerguide/service-integration.html>.

Summary

- AWS KMS is a scalable encryption and key management service provided by Amazon.
- There are two types of popular encryption methods: symmetric and asymmetric.
- Symmetric encryption uses single key to encrypt and decrypt data.
- Asymmetric encryption uses a public key and a private key to encrypt and decrypt data.
- KMS supports only symmetric encryption using a single key to encrypt and decrypt data.
- There are two types of keys: data keys and customer master keys.
- A data key is used to encrypt the data.
- A CMK is used to encrypt a data key.
- KMS does not store or manage data keys. It centrally stores and manages the encryption keys. These are called CMKs.
- Data keys are stored along with the data by AWS services that use KMS to encrypt or decrypt data.
- There are a number of AWS services that support KMS to secure data stored on them.

19

Working with AWS Kinesis

In today's rapidly developing world, data is an asset. For large enterprises in this digital era, data is constantly being generated from various digital sites or sources, such as sensors, telemetry data, audio files, videos, application or server logs, gaming apps, and website click streams. It is, therefore, a challenge to collect, process, and analyze large amounts of data in real time. Usually, the data of any enterprise will exist in various formats, come from different sources, and generate data according to individual velocity rates.

Amazon Kinesis makes it possible to ingest, buffer, and process real-time data in various formats (or varieties). Each of these data generation formats can be of a different size (or volume), and each individual source can have different a data generation speed (or velocity). At the time of writing this book, Amazon Kinesis can be categorized into the following four web services, which we will be covering in this chapter:

- Amazon Kinesis Video Streams
- Amazon Kinesis Data Streams
- Amazon Kinesis Data Firehose
- Amazon Kinesis Data Analytics

Kinesis Video Streams

Amazon Kinesis Video Streams can be used as a building block for constructing applications for real-time video processing or batch-oriented video analytics using a live video feed from a video device that is connected to the cloud. It can be used to capture large amounts of video data from millions of sources, such as security cameras and smartphones. It is also possible to build custom applications for real-time video monitoring applications in order to process data frame-by-frame using the Amazon Kinesis Video Streams API library or simply monitor live stream on the AWS Kinesis web console.

Apart from the video data, it is also possible to use Amazon Kinesis Video Streams to catch time series data, such as audio, thermal imagery, RADAR data, and depth data. It automatically stores the video stream data for the specified retention period on the durable store along with encryption at rest. An alternative to real-time video processing is batch processing the video data. Amazon Kinesis Video Streams produces time-indexes stored data that is based on producer timestamps and ingestion timestamps.

The major advantages of using Amazon Kinesis Video Streams are as follows:

- It is pay as you go; that is, there is no commitment or lock-in period required to use this AWS service.
- It connects millions of cameras of various range.
- It gives you freedom from managing the infrastructure and allows you to focus purely on building a custom application.
- It can be configured to store streaming video data for the specified retention period.
- It allows you to stream data more securely, as it applies **Transport Layer Security (TLS)** encryption on data streaming from source devices and encrypts all data at rest using AWS KMS.
- Optionally, by using AWS IAM, it is possible to manage the access of data from other users or hosted applications.
- Amazon Kinesis Video Streams gives complete freedom to create, customize, deploy, and manage applications. For example, you can use open source **Apache MXNet** or **OpenCV** or third-party solutions provided in the AWS marketplace to process and analyze video streams.

The following diagram visualizes how Amazon Kinesis Video Streams' fully-managed web service works in a nutshell:

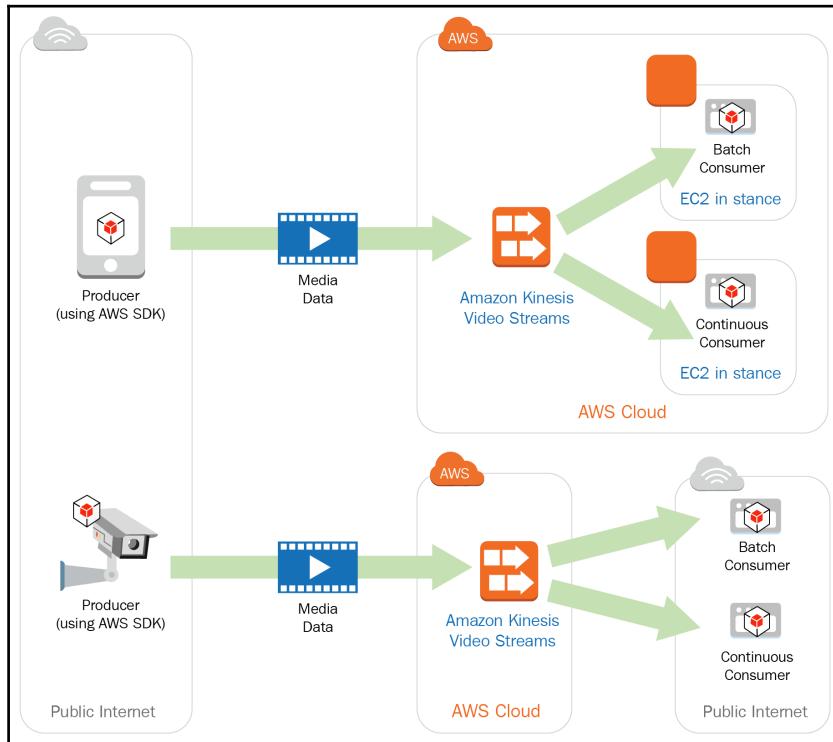


Figure 19.1: Functioning of Amazon Kinesis Video Streams

There are three major components in the preceding diagram. Let's understand them in the following points:

- **Producer:** Any real-world source (usually a camera) that feeds data (audio, video, or images) to the Amazon Kinesis Video Streams is called a **producer**. Even if a device is only feeding audio, images, RADAR data or thermal imagery, or data-depth data, it is called a producer. Each producer can generate more than one video stream and the live data can be fed to more than one Amazon Kinesis Video Stream. For example, a video camera may generate audio and video, and both can be fed to the two individual streams for processing. It all depends on the enterprise requirement, application logic, and architecture.



More details on cameras to be used for running the Amazon Kinesis Video Streams producer SDK can be found at <https://docs.aws.amazon.com/kinesisvideostreams/latest/dg/system-requirements.html>.

- **Kinesis Video Stream:** Usually, in real-world scenarios, each producer feeds live data to only one Kinesis Video Stream. Each stream acts as a platform for a producer to make itself available for the consumer. Optionally, it can also store the data for the specified retention period on the durable storage (S3).
- **Consumer:** Usually, applications consuming Kinesis Streams data in real time or in batch time are called **consumers**. The Kinesis Video Stream Parser Library makes it possible for these applications to get data from Kinesis Video Streams through low latency. It also parses each image frame boundary in the media, making it easy for an application to purely focus on processing and analyzing the frames.

The Kinesis Video Streams API

Kinesis Video Streams Producer SDKs are used by the producer (camera) to write media data to the Kinesis Video Streams and parser libraries, which are used by consumers (applications) to read and process media data from Kinesis Video Streams in real time or batch time. Let's explore both of these APIs in detail in the following subsections.

The producer API

The producer API provides a `PutMedia` API to be used by the camera to write data to the Kinesis Video Stream. It sends a stream of media fragments, where each fragment is simply a self-contained sequence of frames. There is a strict condition in place, in which the frame from one segment cannot be dependent on any of the frames from the other segment.

The consumer APIs

The two `GetMedia` and `GetMediaFromFragmentList` APIs make it possible for a consumer application to fetch data from Amazon Kinesis Video Streams. Let's take a look at these APIs in more detail:

- `GetMedia`: When using this API, the consumer application must identify the starting fragment. It knows where the fragment is found in the data store or is available in real time. Once the fragment is identified, `GetMedia` returns the fragments in the order of fragment numbers from the data store. Each fragment also contains the media data in Matroska (.mkv) format. This gives a consumer application the liberty to fail or fall behind, and then catch-up in real time without any additional effort. Once all of the fragments from the data store are returned and no new fragments are found, `GetMedia` will switch to start reading the fragments from the in-memory stream buffer in real time as it arrives. Real-time consumer applications are also called online consumers.
- `GetMediaFromFragmentList`: These APIs are used by batch-processing consumer applications and they are also called offline consumers. Usually, they fetch particular media fragments by combining the `ListFragments` and `GetMediaFromFragmentList` APIs. These APIs enable the consumer application to match video segments with a particular time range and then to fetch only those fragments sequentially, or in parallel, for processing. This mechanism is, mostly, suitable for the **MapReduce** application consumer.

In the following example, the customer wants to process video fragments for a particular day. This can be done by performing the following steps:

1. Initially, in the `ListFragments` API, specify the desired time range to be processed as a parameter.
2. Now, when we have the fragment metadata list, retrieve the fragments in any order for processing. It is also possible to divide the list into multiple sublists to enable multiple EC2 instances to fetch fragments using `GetMediaFromFragmentList` and process them in parallel.

The following diagram allows you to visualize the preceding steps:

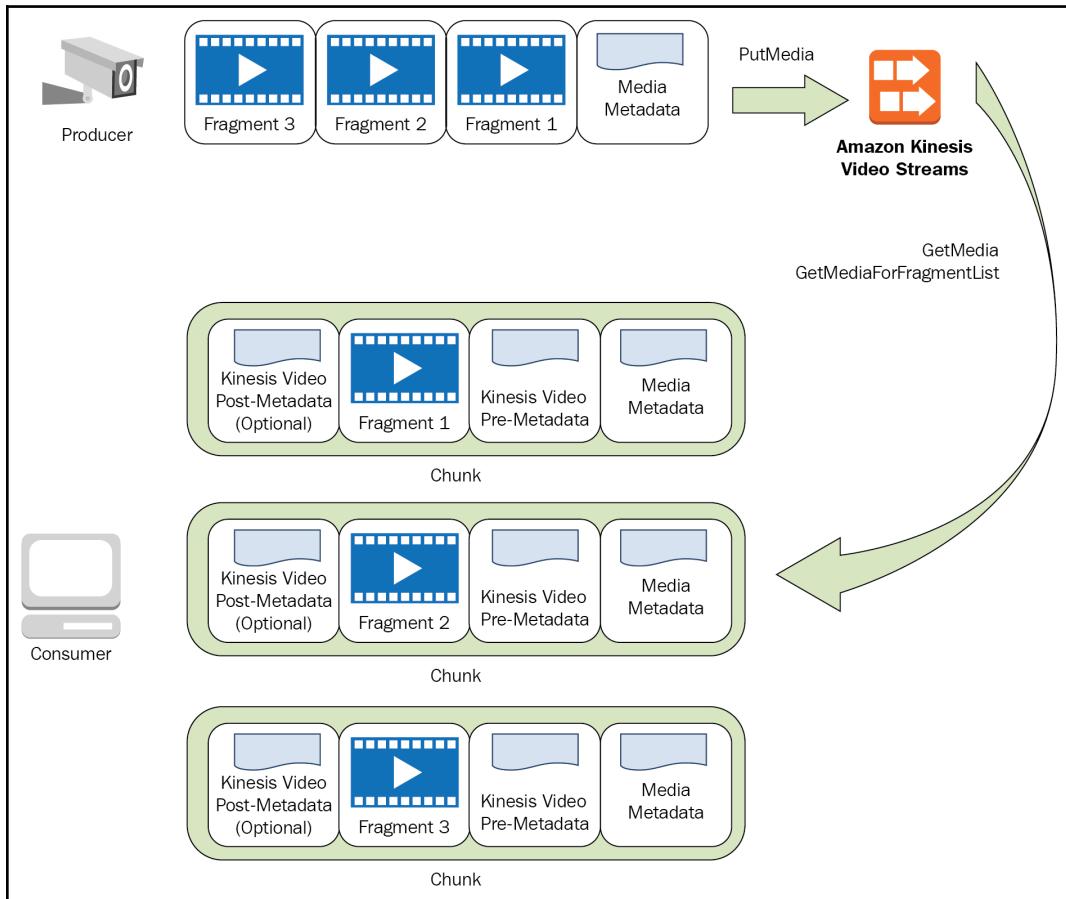


Figure 19.2: Kinesis Video Stream Flow

A producer is a source of Video Stream, such as a camera. Producers need to have Kinesis Video Streams SDKs installed. The SDK is used to send data to an existing Kinesis Video Streams using PutMedia requests. Kinesis Video Streams sends the video metadata in the payload, and then sends a sequence of media data fragments. When these fragments arrive at Kinesis Video Streams, they are stored as Kinesis Video Streams chunks. Each chunk consist of the following details:

- A copy of the media metadata
- A fragment
- The Kinesis Video Streams-specific metadata, such as the fragment number and producer-side timestamps

On the other hand, when a consumer application requests media metadata using the GetMedia and GetMediaForFragmentList APIs, Kinesis Video Streams returns the streams of chunks, starting with the fragment number specified in the request.



Amazon Kinesis Video Streams also supports the **HTTP Live Streaming (HLS)** protocol. It can be used for live viewing, or for you to view archive video. More details on this can be obtained at <https://docs.aws.amazon.com/kinesisvideostreams/latest/dg/how-hls.html>.

Kinesis Data Streams

Amazon Kinesis Data Streams is designed to collect and process large streams of data records in real time. Usually, this data consists of infrastructure logs, system logs, application logs, social media, and web click streams. It is possible to build a custom application to process the received data at Kinesis Data Streams using the **Kinesis Client Library (KCL)**. Such an application can run on Amazon EC2 and can fetch data from data streams as data records.

Architecture

The following diagram can help you to understand the high-level architecture of Amazon Kinesis Data Streams:

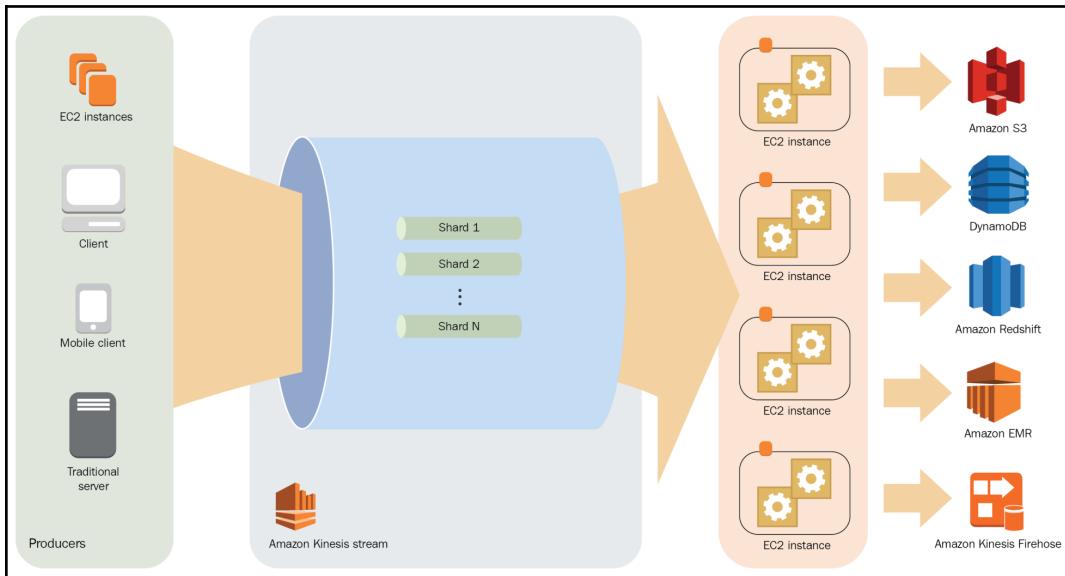


Figure 19.3: High-level architecture of Amazon Kinesis Data Streams

In the preceding diagram, we can see that more than one producer can feed data to the same Amazon Kinesis Data Streams; this is also called Amazon Kinesis Stream. The consumer application can be hosted on EC2 instances to continuously process data in real time. Processed data can be stored in other AWS services, such as Amazon S3, DynamoDB, Amazon Redshift, or Amazon Kinesis Data Firehose.

Kinesis Data Streams terminology

The various terminology involved in Kinesis Data Streams is as follows:

- **Kinesis Data Streams:** This is a managed service; it is used to collect data, store the data durably, and process large streams of data records in real time in order to support the processing of streaming big data. The **Service Level Agreement (SLA)**, on a monthly basis, is at least 99.9%. To achieve this SLA, the stream is partitioned into one or more shards. Each shard has a fixed unit of read and write capacity. This capacity is measured according to the number of data records.
- **Data records:** This is simply a unit to measure the amount of data that is stored in Kinesis Data Streams. Each data record is immutable and they consist of a sequence number (that is, a unique number within the shard), a partition key (for each shard there is a partition key to group the data by shard), and the data in a **Binary Large Object (BLOB)** format.
- **The retention period:** This defines the period time that is required to keep the data records accessible in the data stream by other AWS services or third-party applications as per the enterprise's needs. The default time is 24 hours from the time of the creation of the data record. It can be configured for a maximum of 168 hours (7 days) with additional charges.
- **Producers:** This is an entity that places records into Amazon Kinesis Data Streams. As shown in *Figure 19.3*, a single data stream can be shared among multiple producers and can be anything from an EC2 instance to a traditional server. It is easy to logically group these producers in order to share a data stream.
- **Consumers:** These are applications (called Amazon Kinesis Data Streams applications) that usually run on EC2 instances. Consumers keep fetching data from the data streams constantly, process the data, and then place it in an appropriate store, such as S3, DynamoDB, Redshift, or as per the enterprise architecture requirement.
- **The Amazon Kinesis Data Streams application:** Consumer applications are responsible for processing all of the data records received in Kinesis Data Streams. They can be run on EC2. Additionally, AWS Lambda can be directly triggered by Kinesis Data Streams to process the data records. The same AWS Lambda execution can place the data records in S3 for future usage and then store them in the database (DynamoDB) in order to update the transaction as per the enterprise's requirements. The consumer application fetches data records from the shards using **Amazon Kinesis Client Library (KCL)**, the Kinesis Data Stream API using the SDK in any one of the following ways:

- **With enhanced fan-out:** The customer needs to register for enhanced fan-out. It provides dedicated throughput of up to 2 MiB/second to each of the registered customers. It typically takes a message propagation delay of 70 ms. It charges for data retrieval in addition to consumer-shard hourly cost. It pushes records over HTTP/2.
- **Without enhanced fan-out:** The customer does not need to register for the without enhanced fan-out approach. It provides a fixed throughput of 2 MiB/second per shard. This throughput is shared if there are more customers. It typically takes an average message propagation delay of 200 ms. This delay can increase up to 1000 ms if there are more than five consumers. It pulls models over HTTP:

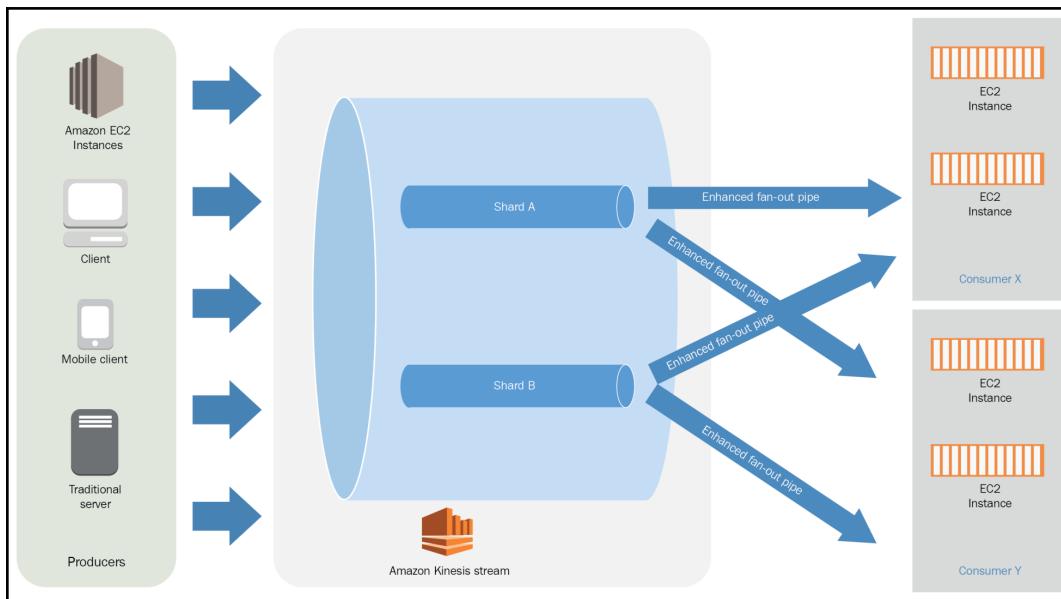


Figure 19.4: Kinesis Data Stream Application

However, when the consumer application is using an API to fetch data records from the data stream, then you have a legacy to subscribe to the individual shards.

- **Shards:** Each shard is a partition in a data stream. Adding more shards will increase stream throughput. It is important to remember, especially when the data record sequence is required for the enterprise application, that shards provide scalability for each stream, but the data record order is only preserved within a shard and not between shards. On the other hand, this makes it possible to uniquely identify each data record in the stream. Each shard can support five transactions per second for reading. The total read rate cannot exceed 2 MBps. In comparison, the total write rate is 1,000 records per second, using a size of 1 MBps. The total Kinesis Data Streams capacity is the sum of the capacity of each shard and can be derived as follows:

Total read capacity in MB for a stream = No. of shards x 2 MB per second

Total write capacity in MB for a stream = No. Of shards X 1 MB per second

Total write capacity in terms of records = No. of shards x 1000 records per second

For example, if there are four shards per stream, then the total capacity will be as follows:

8 MB/second = 4 x 2 MB/second

4 MB/second = 4 x 1 MB/second

4000 records/second = 4 x 1000 records/second

- **The partition key:** Single and multiple data records can be written to Kinesis Data Streams using the PutRecord and PutRecords APIs, respectively. Each write must specify the name of the Kinesis Data Streams, which will eventually store and process the data, a partition key, and a data BLOB. The specified partition key will be used by Kinesis Data Streams as the input to an MD5 hash function in order to map the partition key and its associated data to a specific shard, based on a 128-bit integer value. As a result of this mechanism, each write to Kinesis Data Streams using the same partition key will go to the same shard again and again.
- **The sequence number:** When data is written to Kinesis Data Streams using the PutRecord or PutRecords APIs, it will assign a sequence number to the data records; this sequence number is unique within each shard and continues to increase over time.

- **KCL:** The applications use KCL to consume and process data records from Kinesis Data Streams. KCL is an abstract Java library layer, which supports multiple programming languages, other than Java, with the help of a multi-language interface called **MultiLangDaemon**. This daemon continues to run in the background whenever KCL is used for a language other than the Java programming language. Even in situations where KCL calls are made from languages other than the Java programming language, it requires to have Java installed on the same machine in order to keep the **MultiLangDaemon** interface running. The application that is using KCL is compiled to make it fault-tolerant in order to consume and process data from Kinesis Data Streams.
- **The application name:** Every application inside the region of any given AWS account must be unique. This is in order to identify the application, and the same name is used in the control tables of Amazon DynamoDB and CloudWatch metrics.
- **Server-side encryption:** As soon as data enters Amazon Kinesis Data Streams, it can be encrypted, using the Amazon KMS master key.

Kinesis Data Firehose

The main differences between Kinesis Streams (video and data) and Firehose is that Kinesis Stream stores data persistently for a specified retention period and it allows you to write consumer applications. On the other hand, Firehose doesn't store data persistently nor does it allow you to create a consumer application. It acts as an ingesting service, which means that it collects and transforms data in real time before storing it in supported storage.

At the time of writing this book, Kinesis Data Firehose can be integrated with Amazon's S3 bucket, Amazon Redshift cluster, Amazon Elasticsearch cluster, and Splunk. In general, answering the following questions will help you to select the right service:

- *Are ingesting messages required to be stored in the stream?*
- *Are you looking for a managed solution or are you planning to manage a solution yourself?*
- *How many consumer applications will be reading data from the stream?*

- *What is the storage destination for the data? Is it S3, Redshift, Elasticsearch, Splunk, or any other storage or AWS services?*
- *Does it require any custom or additional processing before the message is stored in storage?*

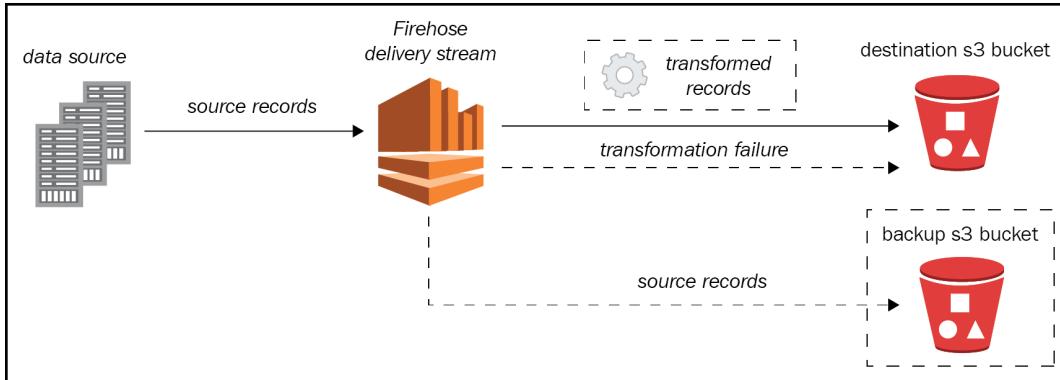
Kinesis Data Firehose – key concepts

The key concepts involved in Kinesis Data Firehose are as follows:

- **Kinesis Data Firehose delivery streams:** This can be easily created from the AWS console, SDK, API, or CLI. It is a managed delivery system, and once the stream is created, it can be modified at any given time. Once it is modified, it will take a few minutes to implement and the version number will be incremented by 1. While creating the stream, we need to consider the following factors:
 - While creating it, is essential to define the name and the source. The source can be Kinesis Stream, direct PUT, or another source.
 - Optionally, with the help of AWS Lambda, record transformation can also be done. Additionally, format conversion at the output can be enabled or disabled; this is possible in the **Apache Parquet** or **Apache ORC** format.
 - It is essential to select an appropriate persistent storage destination and its parameters from the available options.
 - Finally, configure buffering, compression, logging, and the IAM role settings.
- **The record:** This is the actual data that is sent by the producer to Kinesis Data Firehose. It cannot exceed more than 1,000 KB.
- **The data producer:** This is the actual entity that keeps sending real-time data to Kinesis Data Firehose. For example, it could be a web or application server constantly sending log data or an existing Kinesis Data Stream.
- **The buffer size and buffer interval:** The buffer size is measured in MB and it specifies the size for the incoming streaming data. The buffer interval is measured in seconds and it specifies the period of time before data is delivered to the destination.

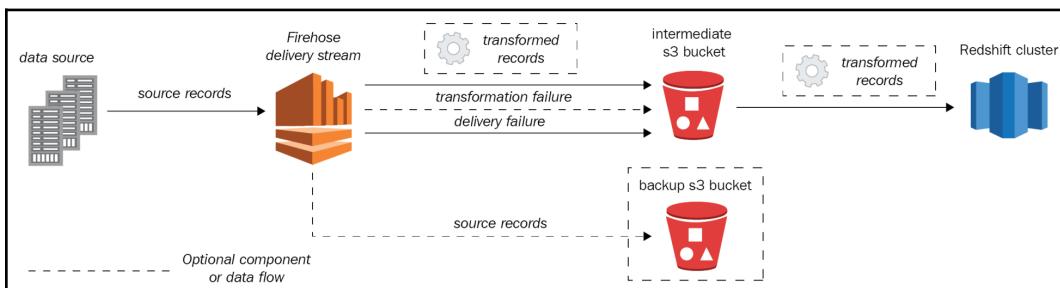
Kinesis Data Firehose – data flow

Let's now explore how data actually flows when a different destination storage is configured. The following diagram is useful for understanding the data flow when the destination is an S3 bucket for Kinesis Data Firehose:



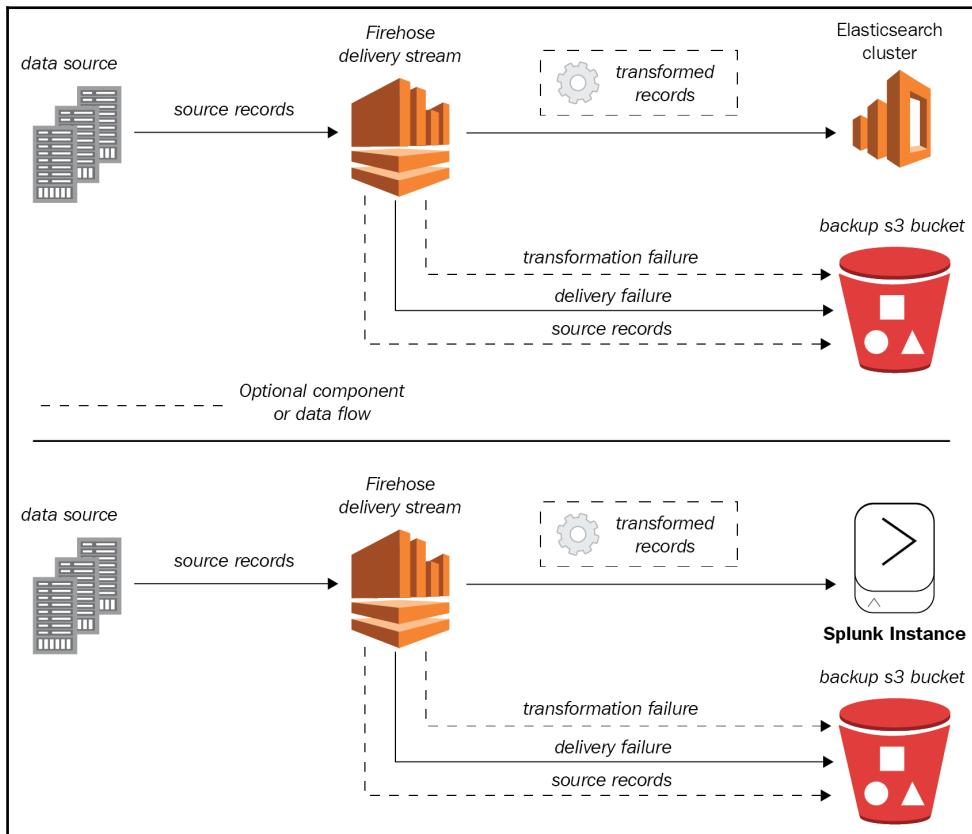
Various data sources feed data in real time to the Kinesis Data Firehose delivery stream constantly. When data transformation is enabled, it is possible to store the actual data in its original state inside another Amazon S3 bucket as a backup. Otherwise, when the transformation is enabled, it will start processing and delivering to the destination S3 bucket as per the buffer size and interval configuration.

The following diagram is useful for understanding the data flow when the destination storage is Amazon Redshift:



Various data sources feed data in real time to the Kinesis Data Firehose delivery stream constantly. First, the streaming data will be stored in the intermediate Amazon S3 bucket. When transformation is enabled, you can choose to store the original data (prior to transformation) in a separate Amazon S3 bucket. When storing data to the Amazon S3 bucket, all records are stored whether or not the transformation or delivery fails. Finally, it will issue the Amazon Redshift `COPY` command to load the data from the Amazon S3 bucket to the Amazon Redshift cluster.

The following diagram is useful for understanding the data flow when the destination is the Amazon Elasticsearch cluster or Splunk instance. In both situations, various data sources feed real-time data to the Kinesis Data Firehose delivery stream constantly. Optionally, it will back up the actual data concurrently and, when transformation is enabled, the transformed records are stored directly in the destination:



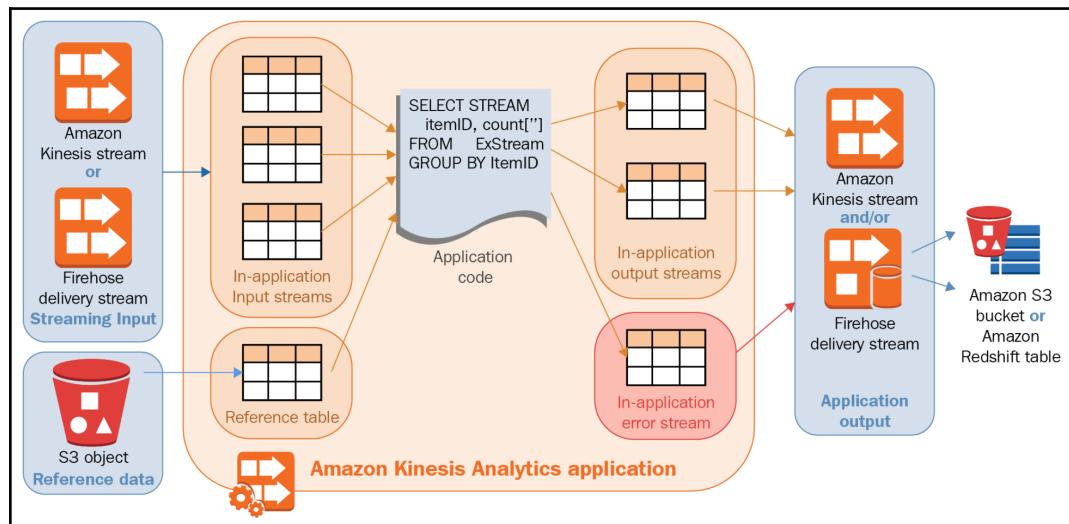
Kinesis Data Analytics

Kinesis Data Analytics is a managed and scalable AWS service that processes and analyzes real-time streaming data using SQL queries or customized Java applications. It can ingest data from Kinesis Streams or the Kinesis Firehose delivery stream. It is widely used to generate real-time metrics, create a feed for the real-time dashboard, and create time series analytics.

Kinesis Data Analytics for SQL applications

In order to start working with Kinesis Data Analytics, we need to understand that an application is a primary resource that is responsible for continuously reading and processing streaming data in real time. Additionally, an application can be stored at a configured application output, such as Amazon Kinesis Streams and/or the Kinesis Data Firehose delivery stream. It is also essential to write the application code in SQL.

The following diagram is useful for understanding how to write the application code in SQL:



Each application has a name, description, version ID, and status. In addition to this, the basic properties for each application are as follows:

- **Input configuration:** This defines the streaming source (Kinesis Data Streams or Kinesis Data Firehose) for the application. It also helps to map the streaming source to the in-application input stream. The in-application stream can be compared to continuously updating a table against which you can perform the `SELECT` and `INSERT` SQL operations. It also consists of a timestamp column (timestamps and the `ROWTIME` column) in each in-application stream. With the help of the application code, it is possible to create an additional in-application stream to store intermediate query results. To improve the throughput, the single-source stream can be partitioned into multiple in-application input streams.
- **Application code:** This is simply a series of SQL statements that continuously process the input and produce an output. These SQL statements can be written against in-application streams or the reference table from Amazon S3 bucket. If a situation arises, it also performs `JOIN` operations on both of these sources.



More information on the supported SQL language elements by Kinesis Data Analytics can be found at <https://docs.aws.amazon.com/kinesisanalytics/latest/sqlref/analytics-sql-reference.html>.

By default, Kinesis Data Analytics provides an in-application error stream for each application. If, for any reason, an application has an issue in processing records, such records are stored in the error stream.

- **Output:** When the application code executes the SQL queries, the query result goes to the in-application streams. Kinesis Data Analytics creates one or more in-application streams in order to hold the intermediate data. Additionally, you can choose to configure your application to do the following:
 - Store data persistently.
 - Store data in an in-application output stream.
 - Store data in an external destination or AWS service, using the Kinesis Data Firehose delivery stream or Kinesis Data Streams.

We have seen earlier that Kinesis Data Firehose can store persistent data in an Amazon S3 bucket, Amazon Redshift cluster, Amazon Elasticsearch cluster, or Splunk instance. On the other hand, using Kinesis Data Streams stores data in other AWS services.



Make sure that Kinesis Data Analytics has sufficient permissions in the input and output resources to perform essential operations.

Kinesis Data Analytics for Java applications

In order to support a Java application for processing real-time streaming data, Kinesis Data Analytics uses the Apache Flink application. Apache Flink is an open source stream processing framework written in Java and Scala. To process Kinesis Streams using Apache Flink (the Java application), we first write an application on a local machine. Once this is done, upload the application to the S3 bucket to make it available for Kinesis Data Analytics.

Such applications are mostly written using DataStreamAPI from Apache Flink. Broadly speaking, Apache Flink programming can be done in the following two categories:

- **The data stream:** In general, this programming model is used to process the continuous flow of structured data records.
- **The transformation operation:** In general, this programming model is used to take one or more data streams as input and produce one or more data streams as the output.

In general, each application consists of a minimum of one data stream with a source, a data stream with one or more operators, and at least one data sink. The key components of Kinesis Data Analytics for Java applications are as follows:

- **Connectors:** This is a software component that is responsible for moving in and out of the Amazon Kinesis Data Analytics application. The different types of connectors are as follows:
 - **Sources:** This type of connector provides data to the application from other data sources, such as Kinesis Data Stream.

- **Sinks:** This type of connector sends data from an application to the destination, such as the Kinesis Data Firehose delivery stream or another destination.
- **Asynchronous I/O:** This type of connector reads data from sources other than Kinesis Data Streams and Data Firehose, such as a database.
- **Operators:** Apache Flink operators are used to transform and aggregate one or more data stream to new data streams in the Kinesis Data Analytics for Java application. Apache Flink offers approximately 25 built-in stream processing operators. When these operators transform the data, the new stream contains modified data from the original data.
- **Timestamps:** Kinesis Data Analytics for Java applications can track events using the following timestamps:
 - **Processing time:** This is the system time where respective operations are executed using a Java application.
 - **Event time:** This is the period of time indicating when each of the events occurred on the producer.
 - **Ingestion time:** This indicates the time at which an event is entered to the Kinesis Data Analytics service.
- **Fault tolerance:** The Kinesis Data Analytics for Java application (the Apache Flink application) uses a checkpoint mechanism to implement fault tolerance. The checkpoint is simply an up-to-date backup of a running application. Every time an application is updated, a fresh checkpoint is created. If, for any reason, an application fails, then a recent checkpoint is used to recover the application without losing data. The application's checkpoint behavior can be configured as per requirements, using parameters such as `CHECKPOINTING_ENABLED`, `CHECKPOINT_INTERVAL`, `CONFIGURATION_TYPE`, and `MIN_PAUSE_BETWEEN_CHECKPOINTS`. It is also possible to take a manual backup of the application using snapshot.
- **Scaling:** For a Kinesis Data Analytics for Java application (the Apache Flink application), it is possible to run parallel executions of the tasks and allocation of resources, using `PARALLELISM_CONFIGURATION` properties, such as `PARALLELISM` and `PARALLELISM_PER_KPU`.



More information on this can be found in the Apache Flink documentation at <https://ci.apache.org/projects/flink/flink-docs-release-1.6/dev/parallel.html>.

- **Granting permissions:** Amazon Kinesis Data Analytics requires permission to read records from the specified streaming source and it requires permission to write an application output to the sinks that are specified in the application configuration. Such privileges can be granted using the IAM role, which can be assumed by Amazon Kinesis Data Analytics. Each IAM role has two policies attached to it:
 - **Trust policy:** This indicates who can assume the role.
 - **Permission policy:** This specifies the set of permissions to grant.

Summary

- Amazon Kinesis Video Streams can be used as a building block for building applications for real-time video processing.
- Apart from the video data, it is also possible to use Amazon Kinesis Video Streams to catch time series data, such as audio, thermal imagery, RADAR data, depth data, and more.
- Kinesis automatically stores video stream data for the specified retention period on the durable store along with encryption at rest.
- Kinesis can be configured to store streaming video data for the specified retention period.
- Amazon Kinesis Video Streams gives complete freedom to create, customize, deploy, and manage applications.
- Any real-world source (usually a camera) that feeds data (audio, video, or images) to the Amazon Kinesis Video Streams is called a producer.
- Usually, applications consuming Kinesis Video Streams data in real time or in batch time are called consumers.
- Kinesis Video Streams Producer SDKs are used by the producer (camera) to write media data to the Kinesis Video Streams.
- The producer API provides a **PutMedia** API to be used by the camera to write data to the Kinesis Video Stream.
- The two **GetMedia** and **GetMediaFromFragmentList** APIs make it possible for a consumer application to fetch data from Amazon Kinesis Video Streams.

- Amazon Kinesis Video Streams also supports the **HTTP Live Streaming (HLS)** protocol. It can be used for live viewing, or for you to view archive video.
- Amazon Kinesis Data Streams is designed to collect and process large streams of data records in real time.
- More than one producer can feed data to the same Amazon Kinesis Data Streams.
- Kinesis Data Streams is used to collect data, store the data durably, and process large streams of data records in real time in order to support the processing of streaming big data.
- Single and multiple data records can be written to Kinesis Data Streams using the **PutRecord** and **PutRecords** APIs, respectively.
- When data is written to Kinesis Data Streams using the PutRecord or PutRecords APIs, it assigns a sequence number to the data records.
- The main differences between Kinesis Streams (video and data) and Firehose is that Kinesis Stream stores data persistently for a specified retention period and Firehose doesn't store data persistently.
- Kinesis Data Analytics is a managed and scalable AWS service that processes and analyzes real-time streaming data, using SQL queries or customized Java applications.

20

Working with AWS CodeBuild

In the previous chapter, we saw how important a source code repository is in today's world, where software development and distribution happen at a phenomenal speed. Usually, an enterprise-grade remote source code repository (that is, AWS CodeCommit or Git) consists of several branches, such as the development branch, QA branch, staging branch, and master branch.

The main purpose of a development branch on a remote repository is to allow various teams of developers to fork a new branch for a bug fix, hotfix, or feature development, as and when required. They will clone this newly-forked branch on their local development laptop or a computer. During a development, each of the developers in a team will merge their code to the forked branch on a remote repository. Before merging this recently developed code with the development branch, it is essential to carry out a peer review and several other tests, such as unit tests, integration tests, and UI tests.

Now, during the development of a particular feature or bug fix, you may need to carry out tests several times, as soon as they are committed to the central remote repository. When and how to merge branches and carry out tests varies for each enterprise, as per their standards.

The following topics will be covered in this chapter:

- Introducing AWS CodeBuild
- Understanding AWS CodeBuild
- Working with AWS CodeBuild

Introducing AWS CodeBuild

In general, there are various models and strategies available to perform automated testing on an application or a source code. One of the most famous in testing strategies is Mike Cohn's **Test Pyramid**. In general, 70% of the time are spent on unit testing (that is, code testing). This can be done using AWS CodeBuild. Another 20% and 10% of the total time is spent on the service testing and UI testing, respectively, using third-party tools. The following diagram helps to illustrate this. The test starts with unit testing, which is more isolated and faster to carry out; this is at the bottom of the pyramid. As we move toward the top, testing becomes slower and more integrated:

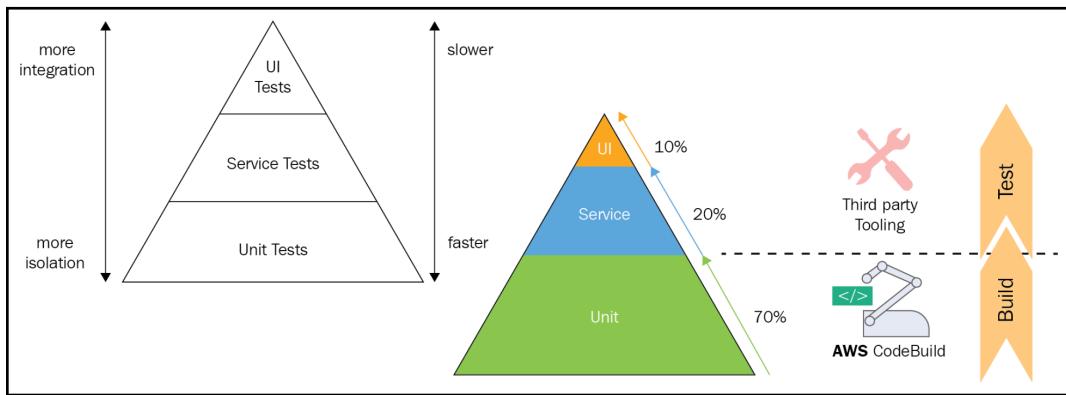


Figure 21.1: Mike Cohn's Test Pyramid

If the application development language is compiler-based, such as .NET, Java, Go, or iOS, we need to convert the committed source code into binary before it is deployed to the test servers. On the other hand, when the development language is an interpreter language, such as PHP, Ruby, Python, and Node.js, it doesn't require us to create and build the code.

However, even if we use an interpreter language for coding, it is recommended to ensure the following points for the source code:

- Inspect the code and audit it for any security concerns, and ensure that it doesn't have any security flaws.
- Confirm the desired functionality of the code.
- Identify programming syntax errors.
- Standardize code patterns and formats.
- Eliminate or reduce bugs due to undesired application usage and logic failures.

All of these tasks can be atomized with the **Continuous Integration (CI)** server. AWS CodeBuild not only helps to compile a code (converting source code into the executable binary codes or artifacts), but can also perform various types of unit tests. Once the build is successful, and if it produces artifacts, AWS CodeBuild can place them in a designated S3 bucket.

Understanding AWS CodeBuild

AWS CodeBuild is a fully-managed CI service. It not only compiles a code, but it's also capable of running unit tests and producing artifacts (software packages). It supports most popular programming languages, such as .NET Core, Go, Java, Node.js, PHP, and Python, and most build tools, such as Apache Maven and Gradle. It can perform multiple builds concurrently and it scales automatically. We do not need to provision, manage, and scale a fleet of build servers. We will only be charged per minute. AWS CodeBuild builds can be triggered manually or automatically with the Jenkins plugins for AWS CodeBuild.

AWS CodeBuild can be managed using the web console, the AWS CLI, AWS SDKs, or AWS CodePipeline. AWS CodePipeline is a **Continuous Delivery (CD)** service. The heart of AWS CodePipeline is the pipeline. It defines a workflow for code changes through a release process. AWS CodePipeline and CodeBuild are closely integrated, as shown in the following diagram:

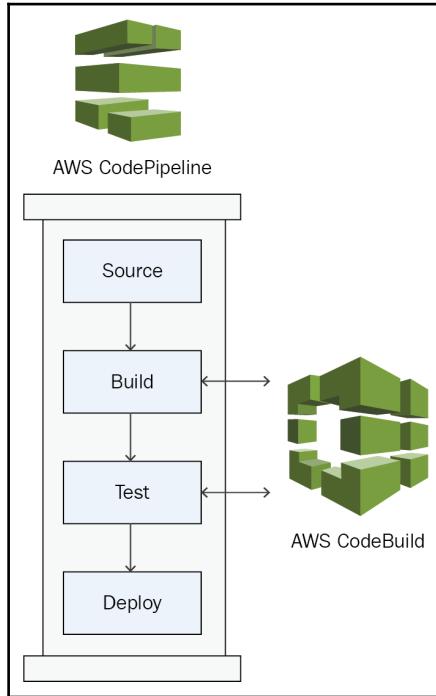


Figure 21.2: Integration of AWS CodePipeline and CodeBuild

Once artifacts are produced using AWS CodeBuild (with the help of AWS CodePipeline), they can be stored or deployed at the artifact's repository or on servers in the desired environment (such as development, test, QA, or production).

Working with AWS CodeBuild

The AWS CodeBuild document has beautifully explained the way AWS CodeBuild works closely with AWS core services, such as Amazon SNS, Amazon S3, and Amazon CloudWatch Logs; and AWS DevOps services, such as AWS CodePipeline, as shown here:

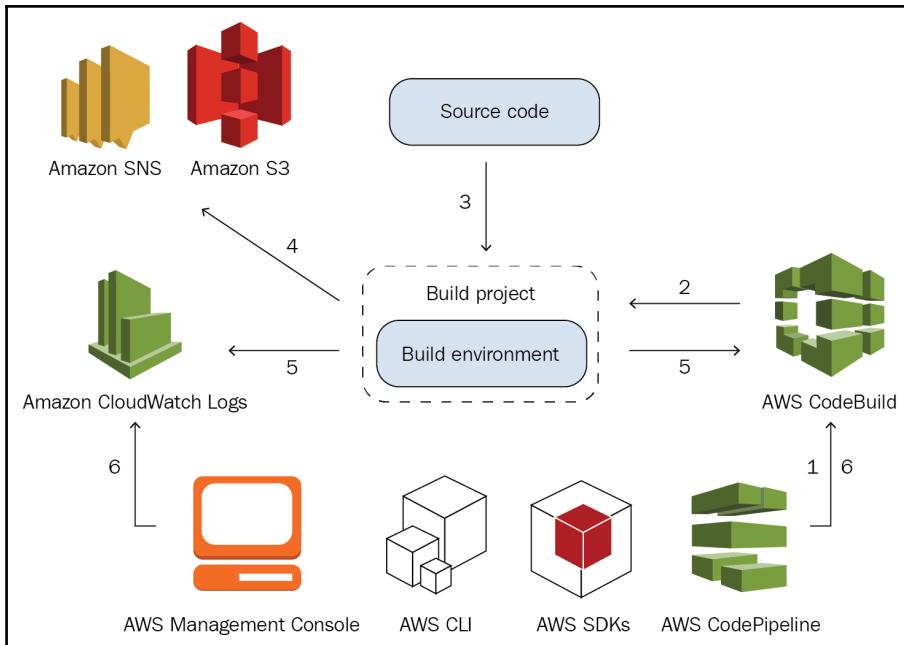


Figure 21.3: AWS CodeBuild with core AWS services

Let's discuss the preceding diagram:

1. The **Build project** is an essential element for the AWS CodeBuild service, and it defines how CodeBuild runs a build. It can be created using the AWS CodeBuild web console, the CLI, or SDK. It defines the steps to be carried out to run a build. Later in this chapter, you will see how to create a build project configuration using a web console.
2. Once AWS CodeBuild's build executes manually or automatically, according to the **Build project** configuration, it will create a **Build environment**. The **Build project** clearly consists of the configuration to be used by the build environment, such as where to get the source code from, the identifier for the Docker image, the build timeout, VPC, and the compute configuration to use to run a build. In other words, AWS CodeCommit will flit between servers to run Docker containers to run a build. At this stage, **AWS CodeBuild** will spin a required number of build servers with a Docker container that has the specifications defined in step 1.

3. **AWS CodeBuild** downloads the source code from the configured source repository to the recently-built project. The **Build project** consists of the build environment, as shown with a dashed rectangle in the preceding diagram.
4. When a build is completed successfully, if it produces any artifacts, these artifacts are uploaded to an Amazon S3 bucket specified in the build configuration.
5. While step 4 is executing, the **Build environment** keeps pumping information to CodeBuild and **Amazon CloudWatch Logs**.
6. While the build is in progress, its latest status can be obtained using the CodeBuild console, the AWS CLI, or AWS SDKs.

Configuring a build project in AWS CodeBuild

Earlier, we saw that the build project defines how CodeBuild runs a build. It can be configured from an AWS CodeBuild web console using **Create build project**, as shown in the following screenshot:

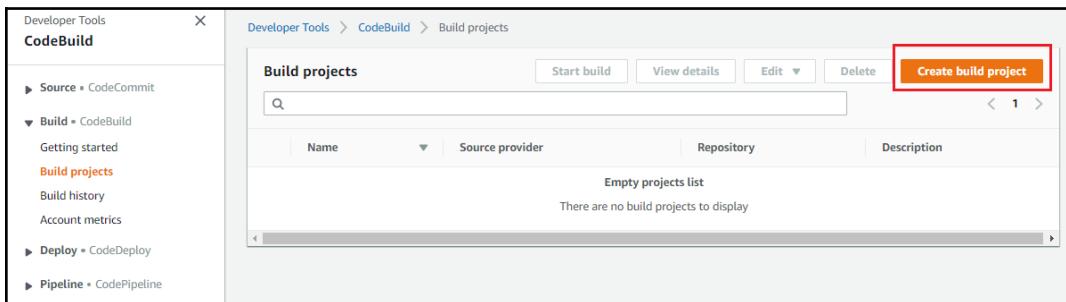


Figure 21.4: Create build project screen

The build project configuration is explained in fragments, as follows.

Project configuration

The project configuration specifies the build identifier, such as the **Project name**, **Description**, **Build badge**, and **Tags**, as shown in the following screenshot:

The screenshot shows the 'Project configuration' screen for an AWS CodeBuild project. It includes fields for 'Project name' (set to 'LearnCodeBuild'), 'Description - optional' (containing 'Learn AWS CodeBuild Project configuration.'), and 'Build badge - optional' (with 'Enable build badge' checked). A 'Tags' section contains a single entry ('BuildType' key with 'Test' value) and buttons for 'Add tag' and 'Remove tag'.

Key	Value	Action
BuildType	Test	Remove tag

Figure 21.5: Project configuration screen



The **Build badge** provides an embeddable, dynamically-generated image. This image is called a **badge**. The purpose of a badge is to display the status of the latest build for a project. When the build badge feature is enabled, AWS CodeBuild will generate a publicly-accessible URL for the CodeBuild project. It doesn't require any authentication, as it doesn't contain any security information and allows anyone to view the status of a CodeBuild project.

Source code

The source code configures the source code repository from one or more provider, such as Amazon S3, AWS CodeCommit, GitHub, BitBucket, or GitHub Enterprise. The following screenshot shows the source code configuration provision from part of the build project:

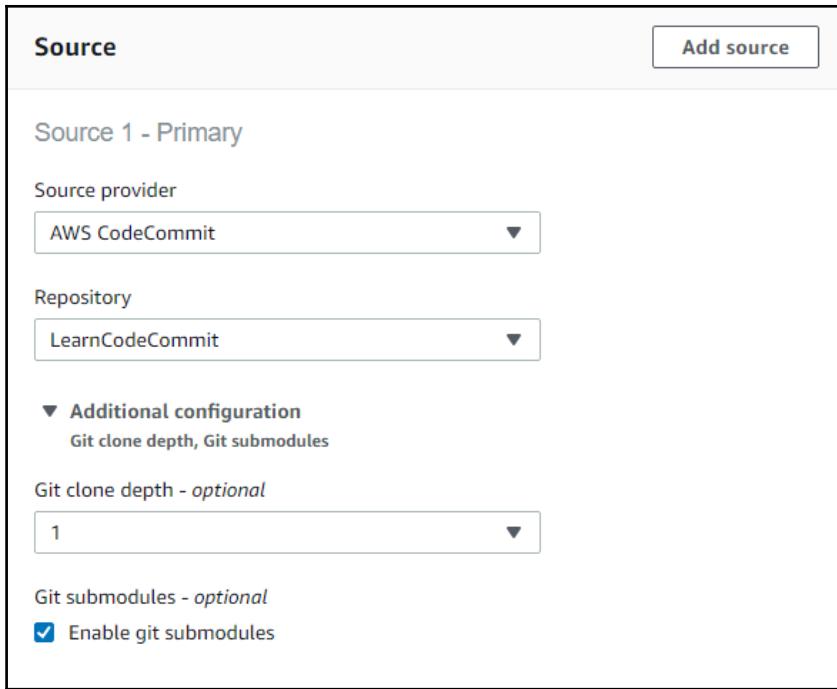


Figure 21.6: Source code configuration



Git submodules make it possible to use parent development projects stored in a separate repository to use as an integrated part of the current source project. More details on how to configure each of these sources can be obtained at <https://docs.aws.amazon.com/codebuild/latest/userguide/sample-build-badges.html>.

Environment

The **Environment** allows us to configure a Docker image identifier managed by an AWS CodeCommit or a **Custom image** repository. This Docker image is used to create a build environment:

Environment image

Managed image
Use an image managed by AWS CodeBuild

Custom image
Specify a Docker image

Operating system

Ubuntu

Runtime(s)

Java

Image

aws/codebuild/java:openjdk-11

Image version

Always use the latest image for this runtime version

Privileged

Enable this flag if you want to build Docker images or want your builds to get elevated privileges

Service role

New service role
Create a service role in your account

Existing service role
Choose an existing service role from your account

Role name

codebuild-Learn Codebuild-service-role

Type your service role name

▶ Additional configuration
Timeout, certificate, VPC, compute type, environment variables

The programming language runtimes are now included in the standard image of Ubuntu 18.04, which is recommended for new CodeBuild projects created in the console. See [Docker Images Provided by CodeBuild](#) for details.

Figure 21.7: Environment image

The **Environment image**, as a **Managed image**, can be configured as follows:

- **Operating system:** This defines the build server operating system. At the time of writing, two options are available: **Ubuntu** and **Windows Server**.
- **Runtime:** Select the desired compiler or interpreter. At the time of writing, the following runtime environments are available:
 - **.NET Code**
 - **Android**
 - **Base**
 - **Docker**
 - **Golang**
 - **Java**
 - **Node.js**
 - **PHP**
 - **Python**
 - **Ruby**
- **Runtime version:** Select the desired major version for the selected runtime. These version choices vary according to the selected runtime.
- **Image version:** This indicates the Docker image tag. More information on how AWS CodeBuild manages images can be found at <https://docs.aws.amazon.com/codebuild/latest/userguide/build-env-ref-available.html>.
- **Privileged:** In general, Docker runs each container (process) in isolation. By default, it runs on a host, but it cannot access the resource outside to the container—that is, a process running inside a container cannot access the resources available on the host. Sometimes, a process may be running inside a container and it may need to access resources from a host machine; in such a situation, it is recommended to enable **Privileged**.
- **Service role:** During a build execution by AWS CodeBuild, we need to create and terminate various AWS resources, such as VPC, EC2, S3, SNS, and CloudWatch Logs, to successfully complete a build. The **Service role** provides sufficient privileges to AWS CodeBuild to create/terminate these resources, as and when required.

You can configure the **Environment image** option as a **Custom image**, as shown in *Figure 21.8*:

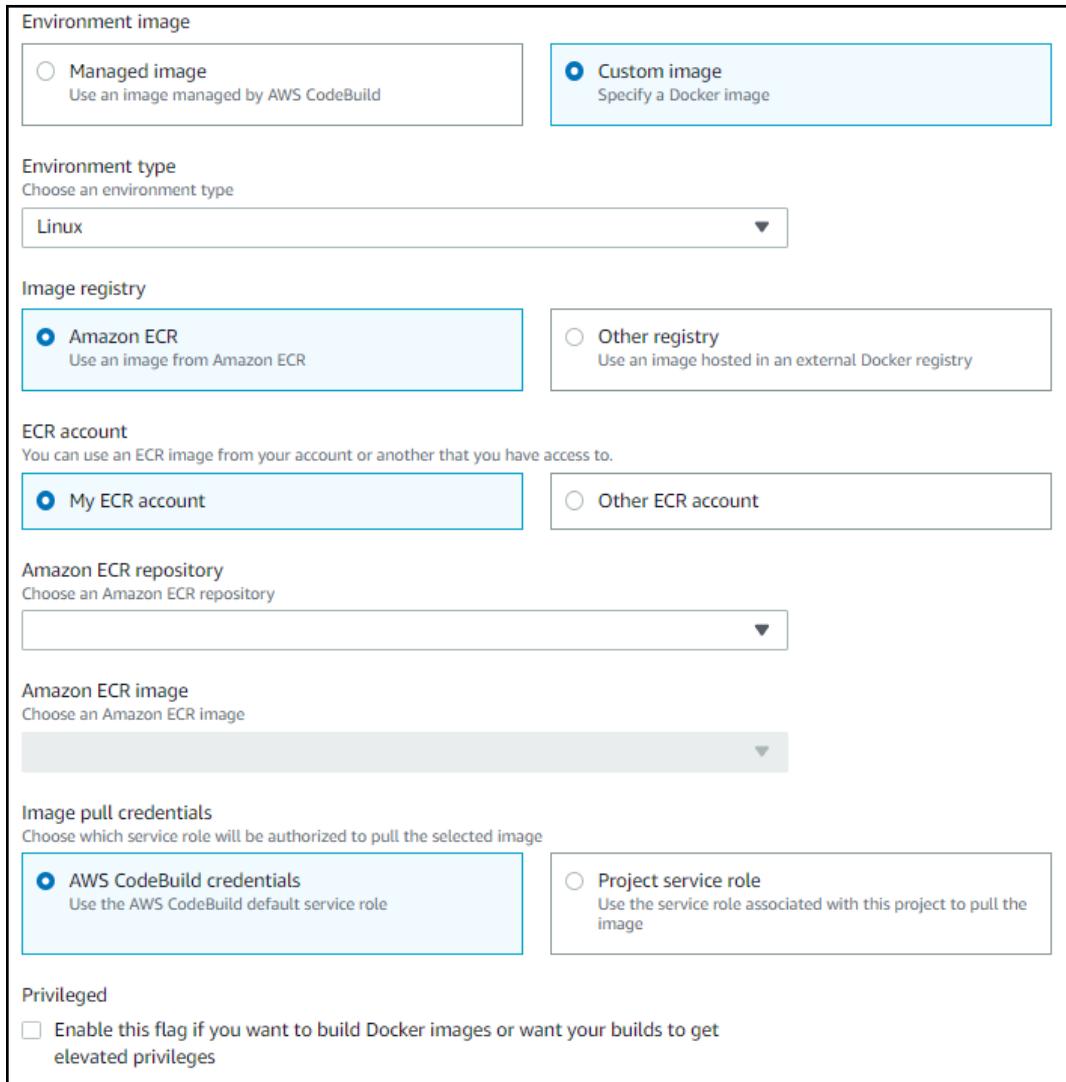


Figure 21.8: Configuring the Environment with a custom image

The fields in the preceding screenshot are explained as follows:

- **Environment type:** This defines the build server operating system. At the time of writing, two options are available: **Linux** and **Windows**.
- **Image registry:** It is possible to use an existing Amazon ECR or a third-party Docker image registry.
- **ECR account and Other ECR account:** This configures Amazon ECR from within the same AWS account or another ECR account.
- **Amazon ECR image:** It will provide a drop-down list of available Docker images in recently-configured ECR accounts. Select an appropriate Docker image identifier for the build. If you do not have any Amazon ECR repositories, this option is disabled.
- **Image-pull credentials:** The privileges required to pull Docker images for the recently-configured Docker image source can be elevated by using either the default created IAM service role or one specifically created for the task. The ECR account, other ECR account, and Amazon ECR image options don't appear when the other repository is configured as a source. The **External registry URL** option appears, as shown in the following screenshot:

Environment image

Managed image
Use an image managed by AWS CodeBuild

Custom image
Specify a Docker image

Environment type
Choose an environment type

Linux

Image registry

Amazon ECR
Use an image from Amazon ECR

Other registry
Use an image hosted in an external Docker registry

External registry URL

docker

<docker repository>/<docker image name>

Registry credential - *optional*

You can supply a secret name or ARN from AWS Secrets Manager to authenticate to an external registry. Providing a credential requires using the project's service role to pull the image.

Privileged

Enable this flag if you want to build Docker images or want your builds to get elevated privileges

Figure 21.9: External registry URL

- **Registry credential:** Usually, accessing the private Docker registry requires credentials. The ARN or the resource name from AWS Secrets Manager can be used. More detailed steps on using AWS Secrets Manager with AWS CodeBuild can be obtained at <https://aws.amazon.com/blogs/devops/how-to-use-docker-images-from-a-private-registry-in-aws-codebuild-for-your-build-environment/>.

Additional configuration

When you select **Custom image** while configuring the environment, you can also set up additional configuration parameters, as shown in the following screenshot:

▼ Additional configuration
Timeout, certificate, VPC, compute type, environment variables

Timeout
Default timeout is 1 hour

Hours	Minutes
1	0

Timeout must be between 5 minutes and 8 hours

Queued timeout
Default time in build queue is 8 hours

Hours	Minutes
8	0

Timeout must be between 5 minutes and 8 hours

Certificate
If you have a self-signed certificate or a certificate signed by a certification authority, choose the option to install it from your S3 bucket.

<input checked="" type="radio"/> Do not install any certificate	<input type="radio"/> Install certificate from your S3 bucket
-----------------------------------------------------------------	---------------------------------------------------------------

VPC
Select a VPC that your AWS CodeBuild project will access.

vpc-d56996b2

Subnets
Select the VPC subnets that AWS CodeBuild should use to set up your VPC configuration. For high availability, we recommend selecting multiple subnets from multiple Availability Zones. Ensure that your subnets include a NAT gateway.

subnet-1c97f436 us-east-1b X

Default Subnet AZ-1b

subnet-35670c50 us-east-1a X

Default Subnet AZ-1a

Security groups
Select the VPC security groups that AWS CodeBuild should use to work with your VPC. Ensure that your security groups allow outbound connections.

sg-70e41d0b X

default

Compute

<input checked="" type="radio"/> 3 GB memory, 2 vCPUs
<input type="radio"/> 7 GB memory, 4 vCPUs
<input type="radio"/> 15 GB memory, 8 vCPUs

Environment variables

Name	Value	Type	Remove
		Plaintext	X

Add environment variable

Create parameter

Figure 21.10: Additional configurations for Custom image

The various options in the configuration are as follows:

- **Timeout:** It specifies the timeout for a particular build run, in hours and minutes.
- **Queued timeout:** It specifies a time, in hours and minutes, for which a build is allowed to be queued before it times out.
- **Certificate:** If you have a self-signed or third-party certificate, you can choose to install it from an S3 bucket; otherwise, you can select **Do not install any certificate**.
- **VPC:** It is possible to specify a VPC to be used to create a build runtime. It helps to achieve compliance and enforce enterprise security and network shaping.
- **Subnets:** A VPC may have n number of subnets, depending on the CIDR. Here, a maximum of 16 subnets can be selected. These subnets are used to spin the compute (that is, EC2) instances.
- **Security groups:** A maximum of five security groups can be attached. These security groups are attached to the spun EC2 instance created during the build runtime environment.
- **Compute:** There are a number of compute types supported with the AWS CodeBuild that you can use to create a build environment. You can refer to <https://docs.aws.amazon.com/codebuild/latest/userguide/build-env-ref-compute-types.html> for the list of supported compute types.



CodeBuild supports Docker images for a custom build environment. The uncompressed Docker image size in Linux and the Windows environment are 20 GB and 50 GB, respectively. More details on the supported compute types can be obtained at <https://docs.aws.amazon.com/codebuild/latest/userguide/build-env-ref-compute-types.html>.

- **Environment variables:** Optionally, it is possible for a particular build to only override environment variables.

Buildspec

The **Buildspec** configuration provides two options, that is, **Use a buildspec file** and **Insert build commands**. The following list explains both options:

- When **Use a buildspec file** is configured, it provides a provision to specify a buildspec file manually, in a YAML format. It is also possible to include a buildspec file as a part of the source code in the remote source code repository. It must be named `buildspec.yml` and placed in the root of the remote source code repository:

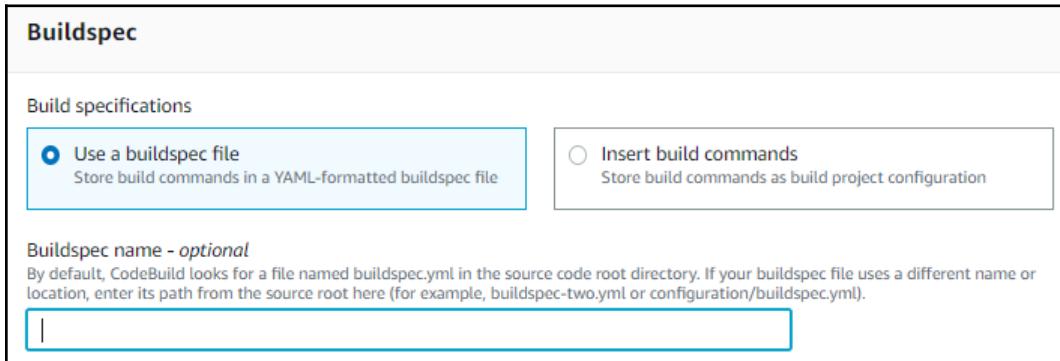


Figure 21.11: Buildspec configuration

- When **Insert build commands** is configured, it provides an embedded editor to edit and write a buildspec file in a YAML format. Writing a buildspec is one of the important elements for a successful build run, but it is beyond the scope of this chapter:

The screenshot shows the 'Build specifications' tab in the AWS CodeBuild console. The 'Insert build commands' option is selected, indicated by a blue border around the radio button and the descriptive text below it. The main area displays a YAML buildspec file with line numbers from 9 to 30. The file structure includes sections for phases, build, and post_build, each containing command definitions. A vertical scroll bar is present on the right side of the code editor.

```
9      # key: "value"
10
11+ phases:
12+   #install:
13+     #runtime-versions:
14       # name: version
15       # name: version
16+   #commands:
17     # - command
18     # - command
19+   #pre_build:
20+     #commands:
21       # - command
22       # - command
23+   build:
24+     commands:
25       # - command
26       # - command
27+   #post_build:
28+     #commands:
29       # - command
30       # - command
```

Switch to single line

Figure 21.12: Buildspec specifications



To learn more about the **Buildspec** tab, visit <https://docs.aws.amazon.com/codebuild/latest/userguide/build-spec-ref.html>.

Artifacts

You can configure the artifacts store when it is expected that the build run will produce artifacts as a result of a successful run. The following screenshot shows the AWS CodeBuild web console for a build project for an artifact:

The screenshot shows the 'Artifacts' configuration page in the AWS CodeBuild web console. The page has a header with 'Artifacts' and a 'Add artifact' button. The main section is titled 'Artifact 1 - Primary'. It includes fields for 'Type' (set to 'Amazon S3'), 'Bucket name' ('codebuildtestartifacts'), and an optional 'Name' field. A checked checkbox for 'Enable semantic versioning' is present. A 'Path' field is set to an empty string. An 'Namespace type' dropdown is set to 'None'. Under 'Artifacts packaging', the 'None' option is selected, showing a description: 'The artifact files will be uploaded to the bucket.' There is also a 'Zip' option. At the bottom, there is a checkbox for 'Remove artifact encryption' and a link to 'Additional configuration'.

Artifacts

Artifact 1 - Primary

Type

Amazon S3

You might choose no artifacts if you are running tests or pushing a Docker image to Amazon ECR.

Bucket name

codebuildtestartifacts

Name - *optional*
The name of the folder or compressed file in the bucket that will contain your output artifacts. Use Artifacts packaging under Additional configuration to choose whether to use a folder or compressed file.

Enable semantic versioning
Use the artifact name specified in the buildspec file

Path - *optional*
The path to the build output ZIP file or folder.

Example: MyPath/MyArtifact.zip.

Namespace type - *optional*

None

Choose Build ID to insert the build ID into the path to the build output ZIP file or folder, e.g. MyPath/MyBuildID/MyArtifact.zip. Otherwise, choose None.

Artifacts packaging

None
The artifact files will be uploaded to the bucket.

Zip
AWS CodeBuild will upload artifacts into a compressed file that is put into the specified bucket.

Remove artifact encryption
Remove encryption if using the artifact to publish a static website or sharing content with others

► Additional configuration
Cache, encryption key

Figure 21.13: Configuring artifacts in CodeBuild

The different options under artifacts are as follows:

- **Type:** It can be either **No artifacts** or **Amazon S3**. Select **Amazon S3** when the build is producing artifacts. It can be used further by other AWS services and/or third-party DevOps tools for CI/CD.
- **Bucket name:** Select the desired bucket name where you want to store the artifacts. Artifacts are stored in the specified bucket at the end of each successful build run.
- **Name:** Optionally, a folder or a compressed filename can be specified to store an artifact inside or as a filename, respectively.
- **Enable semantic versioning:** When a build executes using a buildspec multiple times, it is essential that artifact filenames are unique each time. Usually, the buildspec also defines the artifact name. A specified artifact name in a buildspec overrides a specified artifact name at the respective CLI to be used. For example, a date and timestamp can be inserted into an artifact name at build time to make sure each artifact has a unique name. It is also possible to use declared variables from a buildspec file and a CodeBuild environment variable.
- **Path:** It can be used to store the build output (artifacts) inside a particular directory or a nested sub-directory within the specified S3 bucket.
- **Namespace type:** Optionally, it can be enabled to insert the build ID, along with the configured name and path to store the output artifact in a specified S3 bucket. For example, if the path is configured to `MyArtifacts`, the name is configured as a ZIP file, and the namespace type is configured to the build ID, the artifact would be stored in `MyArtifacts/build-ID/MyArtifact.zip`.
- **Artifact packaging:** It specifies the type of build output artifact to create. By default, it is **None**, which creates output in the specified S3 bucket and a path. Optionally, it can be configured as a ZIP file; it creates output in a ZIP file at a specified S3 bucket and a path.
- **Disable artifact encryption:** Artifact encryption is enabled by default with the S3 artifact type. When the **Disable artifact encryption** option is checked, artifacts are not encrypted. This option is only available with the S3 artifact type.

Logs

When you create a new CodeBuild project, you can configure build output logs to be uploaded in Amazon CloudWatch Logs, as well as in S3. *Figure 21.14* shows the **Logs** configuration options when creating a CodeBuild project:

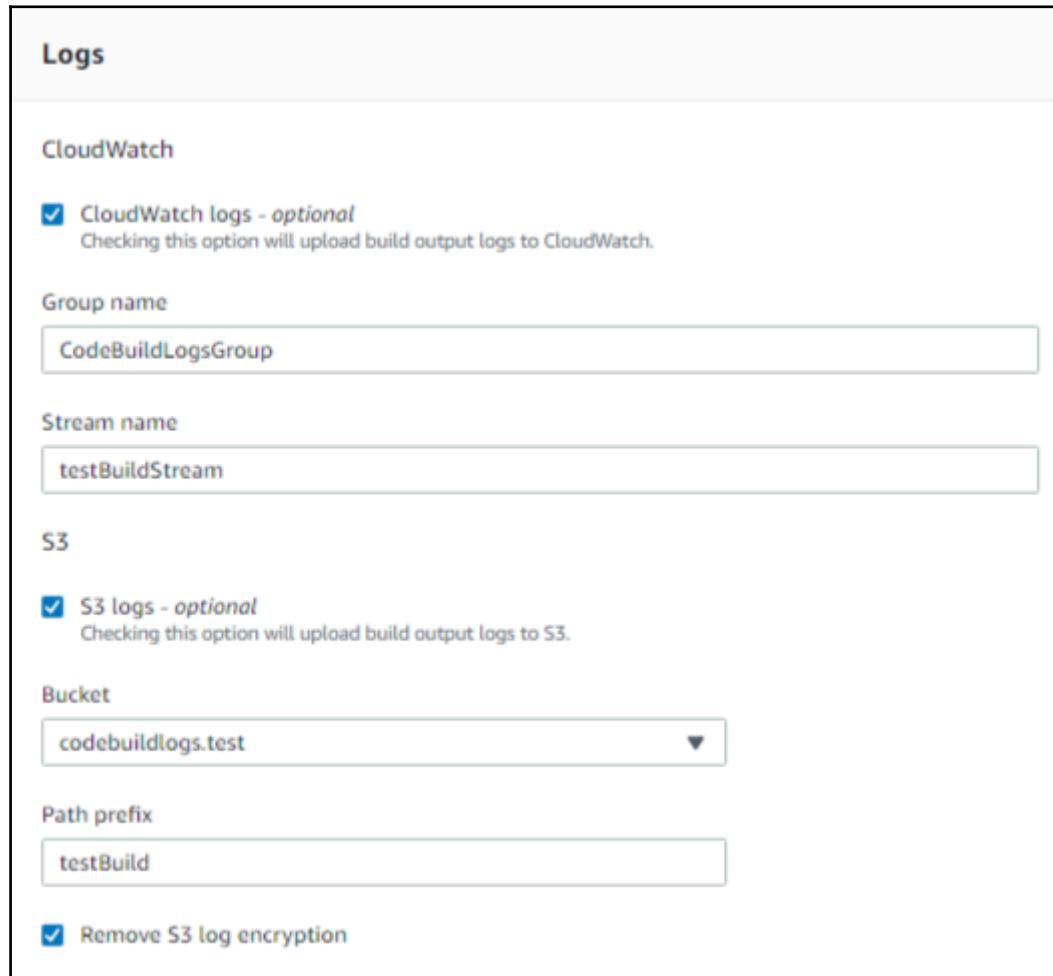


Figure 21.14: Configure CodeBuild logs

Logs can be useful for compliance or troubleshooting purposes:

- **CloudWatch:** As shown in *Figure 21.14*, checking the **CloudWatch** option will upload build output logs to CloudWatch. We need to provide an Amazon CloudWatch log group name and stream name. More information on the topic can be found at <https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/Working-with-log-groups-and-streams.html>.
- **S3:** As shown in *Figure 21.14*, checking the **S3** option will upload the build output logs to S3. By default, this option is unchecked. It can be enabled to store the S3 logs for a build to a specific S3 bucket. Each log file begins with the specified prefixes.

Summary

- An enterprise-grade remote source code repository consists of several branches, such as the development branch, QA branch, staging branch, and master branch.
- The main purpose of a development branch on a remote repository is to allow development team members to fork a new branch for a bug fix, hotfix, or a feature development, as and when required.
- AWS CodeBuild is a fully-managed continuous integration service. It not only compiles code, but is also capable of running unit tests and producing artifacts (software packages).
- AWS CodeBuild supports most popular programming languages, such as .NET Core, Go, Java, Node.js, PHP, and Python, and build tools, such as Apache Maven and Gradle.
- AWS CodeBuild can be managed using the web console, AWS CLI, AWS SDKs, or AWS CodePipeline.
- Once artifacts are produced using AWS CodeBuild (with the help of AWS CodePipeline), they can be stored or deployed in the artifact's repository or servers in the desired environment.
- The build project is an essential element for the AWS CodeBuild service, and it defines how CodeBuild runs a build. The build project clearly consists of the configuration to be used by the build environment, such as where to get the source code from, the identifier for the Docker image, the build timeout, VPC, and the compute configuration to use to run a build.

- CodeBuild supports Amazon S3, AWS CodeCommit, GitHub, BitBucket, or GitHub Enterprise as a code repository.
- The CodeBuild configuration environment type defines the build server operating system. CodeBuild supports Linux and the Windows operating system.
- When a build is completed successfully, and if it produces any artifacts, these artifacts are uploaded to an Amazon S3 bucket specified in the build configuration.
- CodeBuild supports Docker images for a custom build environment. The uncompressed Docker image sizes in Linux and the Windows environment are 20 GB and 50 GB, respectively.
- The buildspec configuration provides two options, that is, **Use a buildspec file** and **Insert build commands**.
- You can configure how artifacts are stored in CodeBuild when it is expected that the build run will produce artifacts as a result of a successful run.
- When you create a new CodeBuild project, you can configure the build output logs to be uploaded in Amazon CloudWatch logs, as well as in S3.

21

Getting Started with AWS CodeDeploy

CodeDeploy is a fully managed build service by Amazon. It can be used to automate code deployment to any instance on AWS EC2 or an on-premises environment. This chapter introduces you to CodeDeploy, and describes how to use CodeDeploy in your development projects.

The following topics will be covered in this chapter:

- The need for CodeDeploy
- Introducing CodeDeploy
- Components of CodeDeploy

The need for CodeDeploy

In general, once software development (that is, such as feature, bug-fix, or hot-fix) is done, and code is committed last time to a specific forked source control branch, then it gets merged to a development and, subsequently, into the master branch. Usually, a peer code review takes place before merging a code into the development branch or any other branch from the forked branch. Now, if this development has been executed using a language that requires a compilation such as Java, VB.NET, or C#, it needs to trigger a **build process**. Again, upon committing a new change or development, a build process is triggered with the new change.

In broad terms, the build process is nothing but compiling the project. If this development or change is done at the level where compilation is not required, such as HTML, JavaScript, or PHP, then it doesn't need to trigger the build process. Once the build process is triggered and when output is in the form of artifacts, it will get stored in the relevant artifact's storage.

In any case, irrespective of whether a build has been triggered, it needs to deploy these compiled artifacts, such as a JAR file, or non-compiled code, such as HTML/CSS pages, to the server. Here, the deployment process comes into play. As every piece of software development can be different and unique, it may require a particular procedure to install, configure, and make sure it is running to achieve maximum application availability. This may include tearing down old servers and building new ones to deploy these new codes. In DevOps, servers are treated as cattle, and rather than fixing them, new servers are created for efficiency. To execute and monitor these processes, a small agent application may be deployed to target servers.

So now, we understand that AWS CodeDeploy is the process of strategically deploying (**blue/green** or **rolling**) any new change (new code or artifacts) to the servers. These target servers may be any environment, such as development, testing, pre-production, or production servers.

Introducing CodeDeploy

AWS CodeDeploy is a software deployment service to automate application deployment to the AWS EC2 instances, on-premises servers or VMs, serverless Lambda functions, or Amazon ECS services. In other words, it can be easily tightly integrated with server, serverless, or containerized architecture and AWS services. This service supports software deployment in many forms, such as the following:

- Source code, such as PHP/Python and many more
- Web and configuration files, such as HTML/XML
- Executable
- Packages
- Scripts
- Multimedia files
- AWS Lambda functions

In a nutshell, it does the following:

- Automatic software deployments to the AWS cloud or on-premises
- Eliminates the need for error-prone manual operations
- Maximizes application availability

AWS CodeDeploy can seamlessly work with Amazon S3 buckets, GitHub repositories, or Bitbucket repositories to read code and deploy to the servers, serverless, or a containerized environment.

Components of CodeDeploy

To start working with AWS CodeDeploy, we begin by building an application. An application is nothing but a logical container for AWS CodeDeploy components such as **revision**, **deployment group**, and **deployment configuration**. To determine what, where, and how to deploy a specific configuration file, the placement of a revision is required. Later in this chapter, we will see in detail what a revision is. But, in short, it is a source of a revised application to deploy. This configuration file is a YAML or JSON file and is called an **AppSpec** file.



For a detailed understanding of an AppSpec file, click here: <https://docs.aws.amazon.com/codedeploy/latest/userguide/application-specification-files.html>.

As it supports code deployment in a server, serverless, and containerized environment, each of these has a different configuration requirements. Hence, at the time of creating an application, it is essential to define an appropriate targeted environment. The following screenshot shows the **Compute Platform** options on the AWS CodeDeploy web console while creating a new application:

The screenshot shows the 'Application configuration' page of the AWS CodeDeploy web console. It includes fields for 'Application name' (set to 'LearnCodeDeploy') and 'Compute platform' (set to 'EC2/On-premises'). A dropdown menu lists other options: 'EC2/On-premises', 'AWS Lambda', and 'Amazon ECS'. A large orange 'Create application' button is visible at the bottom right.

Figure 22.1: Compute platform options on the AWS CodeDeploy web console

Once an application with an appropriate target compute platform is created, it will be listed on the AWS CodeDeploy web console under **Applications**:

The screenshot shows the AWS CodeDeploy Applications page. At the top, there are buttons for 'View details', 'Deploy application', and 'Create application'. Below that is a search bar and a navigation bar with arrows. The main table has columns for 'Application name', 'Compute platform', and 'Created'. One row is visible, showing 'LearnCodeDeploy' as the application name, 'AWS Lambda' as the compute platform, and 'Just now' as the creation time. There are also back and forward navigation arrows at the bottom of the table.

Application name	Compute platform	Created
LearnCodeDeploy	AWS Lambda	Just now

Figure 22.2: Applications in the AWS CodeDeploy web console

As shown in the preceding screenshot, once application configuration has been carried out with the help of the following AWS CodeDeploy components, we can configure actual code deployment. Let's understand the purpose of each of these components:

- **Application:** This must be unique, as it helps to identify application deployment configurations designed specifically for the application/software/code. Also, AWS CodeDeploy uses the same application name to act as a container to isolate and ensure that the correct combination of revision, deployment configuration, and deployment group is referenced during a deployment.
- **Compute platform:** AWS CodeDeploy supports servers, serverless, or a containerized platform to deploy a code, as follows:
 - **EC2/On-premises:** Application deployment configuration can support all on-premises servers, an Amazon EC2 instance, or a mix of both. This type of application (here, *application* means AWS CodeDeploy's logical container with all the code-deployed components) supports all the possible variants of code such as source code, and executables using in-situ or the blue/green deployment type. Later on in this chapter, we will see more details about various deployment types.
 - **AWS Lambda:** This type of application configuration is used to deploy an updated version of an AWS Lambda function.

- **Amazon ECS:** An Amazon ECS containerized application as a task set can be deployed using this type of application configuration. An updated version of the containerized application is installed as a new replacement task intended to perform a blue/green deployment. Finally, upon successful deployment of new task set, an existing original task set is terminated. Until such time as this new task set is under development, all application traffic is sent to the existing task set, and once this new task set is ready, all the traffic is routed to this new task set.
- **Deployment group:** This is a logical grouping of the EC2 instances based on the tags (key-value pairs) attached to them. An individual EC2 instance, or an instance from an Auto Scaling group can be a member of a deployment group based on the specified tags. In the case of on-premises instances, first, it is required to register the instances with CodeDeploy and then tag them. Also, we need to install the AWS CodeDeploy agent to connect with AWS to the public endpoint.



More information on managing on-premises instances can be obtained from this URL: <https://docs.aws.amazon.com/codedeploy/latest/userguide/instances-on-premises.html>.

- **Deployment configuration:** This defines rule conditions to define deployment success and failure during an AWS CodeDeploy's deployment. In other words, the progress of a deployment through a deployment group is determined by the constraint. During this deployment, the minimum number of healthy instances can be defined for a targeted platform Amazon EC2/on-premises instance. It can be defined as a percentage and primarily serves two purposes:
 - It helps to determine whether the overall deployment succeeded or failed. Deployment is considered to be successful if the application revision was successfully deployed at least on the minimum number of healthy instances.
 - It also makes sure that a specified number of instances is healthy during a deployment to allow deployment to proceed. For example, if a specified percentage of the instances are healthy after the code is deployed, it is safe to take a load of the production requests.

- **Deployment type:** This describes the method to be used to make the latest software available on instances in a deployment group, as follows:
 - **In-place deployment:** Once each instance application to be installed is stopped and the latest application version is deployed, finally, the newly installed application is started and validated. It can be used only with the AWS EC2/on-premises instances target platform. It cannot be used with the AWS Lambda compute platform. It is recommended to use a load balancer for this deployment method, as it will deregister each instance one by one to perform an upgrade.
 - **Blue/green deployment:** This minimizes the downtime by running two identical production environments called **blue and green**. Usually, the blue environment is the original, and the green is the revised version. The actual behavior of blue/green deployment in AWS CodeDeployment varies according to the target compute platform.
 - **Blue/green on an EC2/on-premises compute platform:** In this scenario, the original instances in a deployment group are replaced by instances having a revised version of an application using the following steps:
 1. New instances are provisioned to host a new revised version of the application.
 2. The latest revised application is installed on the new instances.
 3. An optional wait time can be configured to perform application testing and system verification.
 4. Once these newly created instances are ready, they are registered with the **Elastic Load Balancer (ELB)** to route production requests to them. Once they have started serving a revised version of an application, original instances from the original environment are deregistered. These deregistered instances can be terminated or kept alive for other purposes.



In this target compute platform, blue/green deployment only works with the EC2 instances. It doesn't work against on-premises instances.

- **Blue/green on an AWS Lambda compute platform:** By default, the deployment type for the AWS Lambda compute platform is blue/green deployment. Hence, there is no need to specify the deployment type. As soon as the revised version of AWS Lambda has been successfully deployed and tested, traffic is routed from the original function to this revised function. Optionally, it is possible to specify Lambda functions to perform validation tests and a traffic shift pattern (that is, **canary**, **linear**, or **all-at-once**).
- **Blue/green on an Amazon ECS compute platform:** Once a successful application deployment is performed, traffic is shifted from the original task set to a revised task set in the same service. To reroute the traffic, it uses a specified port and protocol of a load balancer listener. To carry out validation tests, a test listener can be used to serve traffic to the newly revised task set.
- **IAM Instance profile:** To successfully execute a deployment process on an AWS EC2 instance, we may need to access the Amazon S3 buckets or GitHub repositories where the applications are stored. During a deployment, it may create a new AWS EC2 instance to implement blue/green deployment. To make these instances compatible with AWS CodeDeploy, this additional IAM role is required. If the target compute platform is AWS Lambda or Amazon ECS, this IAM role is not required. The deployment process in ECS deploys an Amazon ECS service, and similarly, a deployment process in Lambda deploys a serverless version of a Lambda function; hence, either of these does not require an instance profile for Amazon EC2 instances.
- **Revision:** A revision is defined as a bundle of a specific version of deployable content such as source code, post-build artifacts, web pages, executable files, and deployment scripts, along with an AppSpec file such as YAML or JSON. Revisions can be accessed by the AWS CodeDeploy agent from GitHub or an Amazon S3 bucket. For a target compute platform, AWS Lambda, or Amazon ECS, a revision file is the same as the AppSpec file. For a target compute platform EC2/on-premises, it requires an AppSpec file along with a source file to be deployed or scripts to be run.



More details on building a revision for a target compute platform EC2/on-premises can be obtained from the following URL: <https://docs.aws.amazon.com/codedeploy/latest/userguide/application-revisions-plan.html>.

- **Service role:** This role grants required permissions to AWS CodeDeploy to successfully execute the deployment process—for example, read EC2 instance tags, publish information to SNS topics, retrieve information from AWS CloudWatch alarms, and allow AWS CodeDeploy to access target instances.
- **Target version:** This indicates the version of the application revision that has been uploaded to the source code repository to deploy to the targeted compute platform. In other words, it indicates the application revision version currently being deployed. During automatic deployment, this is the revision that is pulled for deployments.

An important thing to remember is that AWS CodeDeploy will try to deploy new code in a deployment group whether they are in a running or stopped state. And while calculating a minimum healthy host, stopped instances are considered failed instances. Hence, having so many instances in a stop state may lead to a deployment failure.



Further details regarding the instance health can be obtained from the following URL: <https://docs.aws.amazon.com/codedeploy/latest/userguide/instances-health.html#minimum-healthy-hosts>.

In the case of an application, if the targeted platform is AWS Lambda, production routing can be defined in any one of the following patterns:

- **Canary:** In this approach, once an updated version of the AWS Lambda function is deployed, production traffic is shifted in two increments from the old existing AWS Lambda function. In order to specify the percentage of the traffic shifted, an AWS Lambda function can be configured from the pre-defined canary options for updating itself in the first increment. The function also needs to specify the interval in minutes before the remaining traffic is shifted in the next increment. It helps to roll out the new software version slowly and incrementally to safely assess the quality of the new version against production traffic.
- **Linear:** In this approach, request traffic from the original AWS Lambda function is shifted linearly to a new Lambda function in equal increments with the same amount of time between each increment. The percentage of traffic shifted in each increment and the amount of time between each increment can be specified by configuring the pre-defined linear options.

- **All-at-once:** In this aggressive approach, you may not have adequate time to safely assess the quality of the new version against production traffic. All traffic is shifted from the original AWS Lambda function to the updated Lambda function version all at once.

Summary

In this chapter, we learned how to automate application deployment to the AWS EC2 instances, on-premises servers or VMs, serverless Lambda functions, or Amazon ECS services using CodeDeploy. We looked at the various components of CodeDeploy, such as revision, deployment group, and deployment configuration.

In the next chapter, we will be working with **CodePipeline**.

22

Working with AWS CodePipeline

In today's rapidly growing world, applications are also required to grow rapidly in order to meet end user expectations. To achieve agility, constant innovation, and faster time to market, it is essential to implement **Continuous Integration/Continuous Deployment (CI/CD)** in order to deploy rapidly changing application code and to avoid any errors that may be caused manually.

AWS CodePipeline plays an important role here: it acts as a fast and reliable continuous delivery service in the DevOps era. It automates the steps that are required to fetch new code from a source code repository (such as Git or CodeCommit) in order to build it and make it ready for deployment through the software release process. The software release process first entails deploying code to the development environment, then to the testing environment in order to conduct various tests to make sure that the code matches enterprise standards, and finally, to the production environment, where it will be used by end users.

The following topics will be covered in this chapter:

- Introducing CodePipeline and workflows
- AWS CodePipeline usages
- AWS CodePipeline – a higher-level view
- AWS CodePipeline concepts
- Working with CodePipeline

Introducing CodePipeline and workflows

AWS CodePipeline is able to release new features or bug fixes more frequently by atomizing builds. It performs tests and makes code ready to be deployed to various environments. It can be configured using the AWS CodePipeline web console or a CLI. A **workflow** defines the various steps in a software release process and it can be customized as per an enterprise's requirements. The workflow briefly instructs AWS CodePipeline on how new code changes will progress through each stage of the release process, including how and where the newly modified code should be built, tested, and deployed.

Every code change made to the source code repository is automatically pushed through the set of actions defined in the workflow (or pipeline). This is to make the modified code run through a standardized process after every commit. Optionally, Amazon CodePipeline sets manual approval at the end of each stage of the pipeline for the reviewer to check. When manual approval is enabled, then only changed code will progress forward in a workflow (or pipeline). On the other hand, during workflow execution, if any action fails at any stage for any reason, it automatically stops the workflow: for example, if a build or a unit test fails.

The workflow (pipeline) is extensible and works seamlessly with many other source code repository, build, testing, and deployment tools from AWS or third parties at each stage of the pipeline. It uses different services at each stage of the pipeline; for example, AWS CodeCommit can be used as a source code repository, while AWS CodeDeploy can be used as a deployment service to deploy code to EC2 instances. CodeDeploy also allows you to trigger an AWS Lambda to execute custom logic during any stage of the pipeline. It also gives you the freedom to build a custom plugin or use third-party plugins that can be used in the pipeline at any stage. Additionally, with the help of AWS IAM, privileges can be controlled – that is, you can control who can create, modify, and delete workflows.

You should also note that charges per month are applicable only for active pipelines; there are no charges for setting up pipelines.

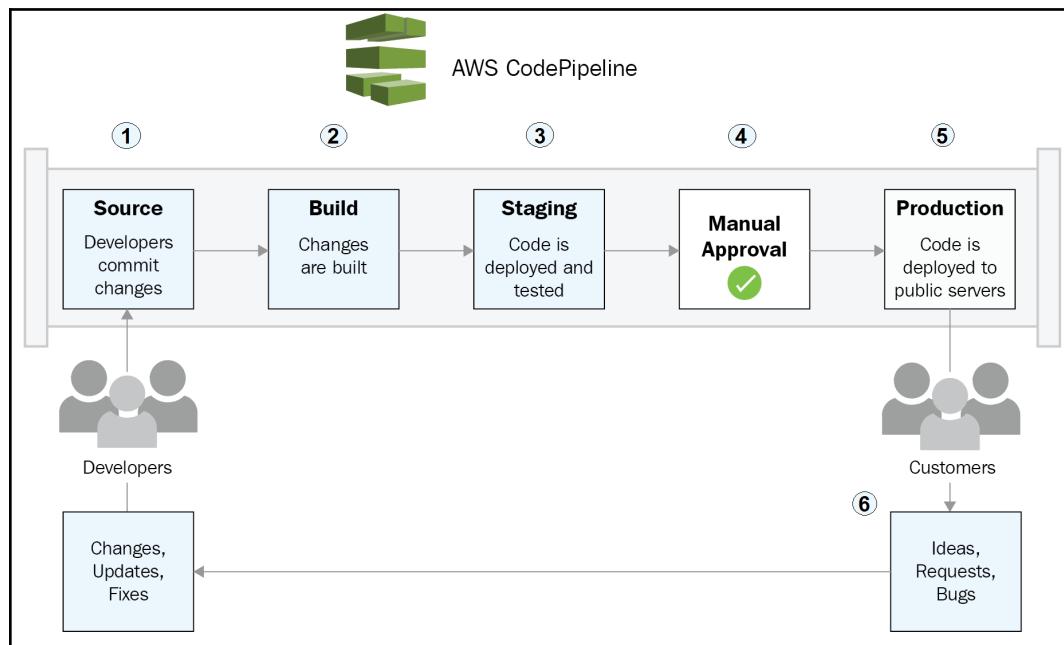
AWS CodePipeline usages

In general, AWS CodePipeline can be used to automatically build, test, and deploy applications from the source code repository in order to build, test, and deploy applications in various environments. In a broader sense, AWS CodePipeline is useful in the following scenarios:

- **To automate a consistent release process:** A consistent set of steps can be defined and carried out every time a code change occurs. Amazon CodePipeline executes each stage of a release automatically according to the defined criteria.
- **To speed up delivery while improving quality:** All the steps of a release process are clearly specified and carried out automatically, which speeds up the new release process.

AWS CodePipeline – a higher-level view

The following diagram is useful for gaining an understanding of the release process using Amazon CodePipeline at a very high level:



The various steps of the preceding diagram can be explained as follows:

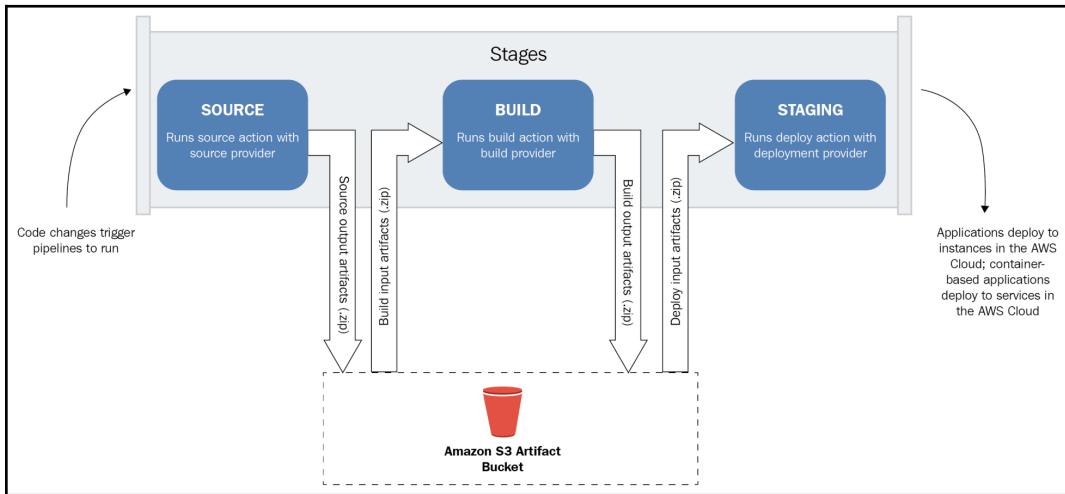
1. **Source:** The release process cycle begins as soon as developers commit a change to the source code repository.
2. **Build:** CodePipeline promptly and automatically identifies changes in the source code repository. It can perform the configured code quality tests. Once the code passes the tests, it builds the code and then compiles it.
3. **Staging:** The recently built code is deployed to the staging environment in order to carry out tests such as integration and load tests.
4. **Manual Approval:** In contrast with continuous deployment, in continuous delivery, the final stage for deploying the code (that is, the build stage) to production requires human intervention. This means that a developer is required to manually approve the results or the progress so far.
5. **Production:** Once approved, CodePipeline deploys the application to the production environment; it does so with the help of CodeDeploy, which deploys the application code to the EC2 instance, AWS ElasticBeanstalk, or AWS OpsWorks. If the application is containerized, then it can be deployed to the Amazon ECS service.
6. Once the application is deployed, it is constantly monitored for performance and bugs. Using monitoring results and customer feedback, developers may introduce a new code change to the application, and so the whole pipeline will execute automatically.

Here, in this CodePipeline example, we have referred to a simple pipeline construct. However, they can be complex and use many other AWS or third-party services and tools.

A high-level view of the input and output artifacts at each stage of the pipeline

Each pipeline in AWS CodePipeline can consist of several stages; each of these stages requires input and output artifacts. These artifacts are stored in the artifact store for the pipeline. The artifact store is simply an Amazon S3 bucket, which is also called the **artifact bucket**. When creating a new pipeline, the default location or custom location in S3 can be configured to be used by an artifact store.

A common artifact store is available for all pipelines in the same AWS regions. During automatic pipeline execution, in order to isolate the artifacts for each pipeline, an individual folder is created in the artifact store. At the end of each execution stage, the output artifacts are zipped and the file is transferred to the artifact store. The same artifact will then be used as an input for the next stage. The following diagram is useful for understanding the high-level artifacts workflow for each stage of AWS CodePipeline:



The various steps of the preceding diagram can be explained as follows:

1. AWS CodePipeline automatically identifies a new commit to the source code repository and triggers and executes a pipeline. In the first step, the **SOURCE** stage, CodePipeline will fetch updated source code from the source code repository and will provide an output as an artifact. It is stored in the **Amazon S3 Artifact Bucket**.
2. The output of the previous step (that is, the artifact) is fetched from the artifact bucket as an input for the **BUILD** stage. Again, the output of the **BUILD** stage (that is, the artifact) is stored in the **Amazon S3 Artifact Bucket** to be used during later stages in the pipeline.
3. The output artifact (build) from a **BUILD** stage can be used for a **STAGING** stage to conduct tests or production deployment.



You can find out more details about the various stages of integration into the AWS CodePipeline at <https://docs.aws.amazon.com/codepipeline/latest/userguide/integrations-action-type.html>.

AWS CodePipeline concepts

In order to work effectively and efficiently with AWS CodePipeline, it is essential to understand some of the concepts and terms that are used with a pipeline.

CI with AWS CodePipeline

It is a common software development practice for each developer in a team to frequently commit their application code into the respective branches of the centralized source code repository. For example, a group of developers or an individual developer working on a bug fix will commit code into a specific bug fix branch. The purpose of this software development methodology is to identify a bug at an early stage. A bug can become quite obvious when several developers are working on the same part of the application development at the same time.

By identifying an error at an early stage of the software development lifecycle, each change is then built and verified. Put simply, CI mainly focuses on constantly building and testing newly committed code.

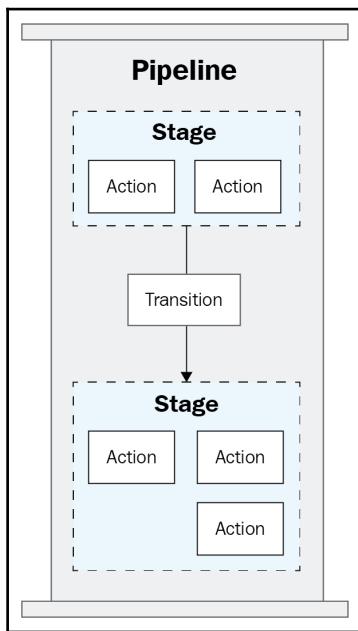
Continuous delivery with AWS CodePipeline

Continuous delivery is another popular software development methodology that is used alongside CI to automate a new release process in an agile environment in order to minimize the time to market. In this methodology, every stage is automatic, such as the build, test, and deployment to production stages.

During the final stage—deployment to production—a person or an automated test decides when the final push takes place. Any software change that is successfully tested can be immediately deployed to production; however, you may or may not want to deploy the changes to production right away. Depending upon the criticality of the workload or application, organizations follow a change control process and may wait for the change control board to approve the changes for deployment to production.

Working with CodePipeline

AWS CodePipeline automates and manages the release process workflow through the use of pipelines. A pipeline is simply a workflow that defines how software changes should take place during a release process. The following diagram is useful for understanding the concepts of pipelines:



The various steps of the preceding diagram can be explained as follows:

- A pipeline in AWS CodePipeline can be created using a web console, SDKs, or a CLI. At the same time, we can define a default or a custom Amazon S3 bucket to use as an artifact store. This is created in the same region as the pipeline is created in. When a default Amazon S3 bucket is selected to use as an artifact store, the naming convention is `codepipeline-region-123456789EXAMPLE`, where `region` is the actual AWS region in which the pipeline is being created. For every pipeline, an individual folder is created in the Amazon S3 bucket to store input/output artifacts at each stage individually. In every region, there must be at least one Amazon S3 bucket as an artifact store. If a pipeline performs a cross-region action, it is essential to configure an artifact bucket for each of the regions used in the action.

- Every commit or change that is made to the configured source code repository in the source action is called a **revision**. The repository can be any valid repository, such as the GitHub repository, the AWS CodeCommit repository, or a version-enabled Amazon S3 bucket. As soon as new code is committed to the source code repository, the revision runs separately through a dedicated pipeline. It is possible to process multiple revisions in the same pipeline, but this is not recommended, as each stage can process only one revision at any given time. It is important to remember that when a pipeline source stage is configured with multiple sources, then all of them run again—even if a commit takes place only in one source.
- A pipeline workflow in AWS CodePipeline is broken into multiple stages. Usually, these stages are designed to perform a particular task in the software release lifecycle. For example, there may be a source stage where a code is fetched from a source code repository as soon as it is committed. Similarly, there could be a build stage to compile the source code, and a deployment stage to deploy a build to the production environment. It is essential to have a unique name for each stage of a pipeline. Each stage consists of one or more sequenced actions to be carried out, and each action must be completed successfully in order to complete the stage. Each stage can process only one revision at a time and the revision must run through all the stages of a pipeline before the next revision can run.
- One or more sequenced actions can exist in each stage of a pipeline, and each action can perform a specific task on the input artifact. When a stage in a pipeline has more than one action, then they can be configured to execute in sequence or in parallel order.

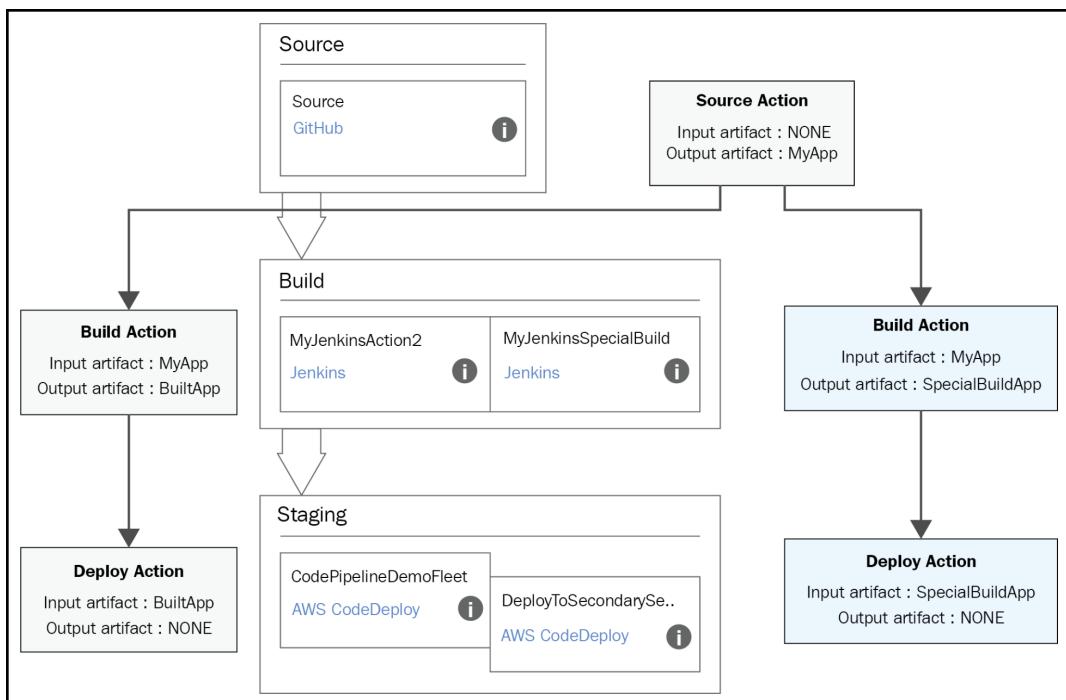


More details on the valid action types and providers in AWS CodePipeline can be obtained at <https://docs.aws.amazon.com/codepipeline/latest/userguide/reference-pipeline-structure.html#actions-valid-providers>.

Once a revision starts running through a pipeline, the output is zipped and stored in an Amazon S3 artifact bucket on the successful completion of a stage. It is then fetched from the Amazon S3 artifact bucket by a consecutive stage as an input:

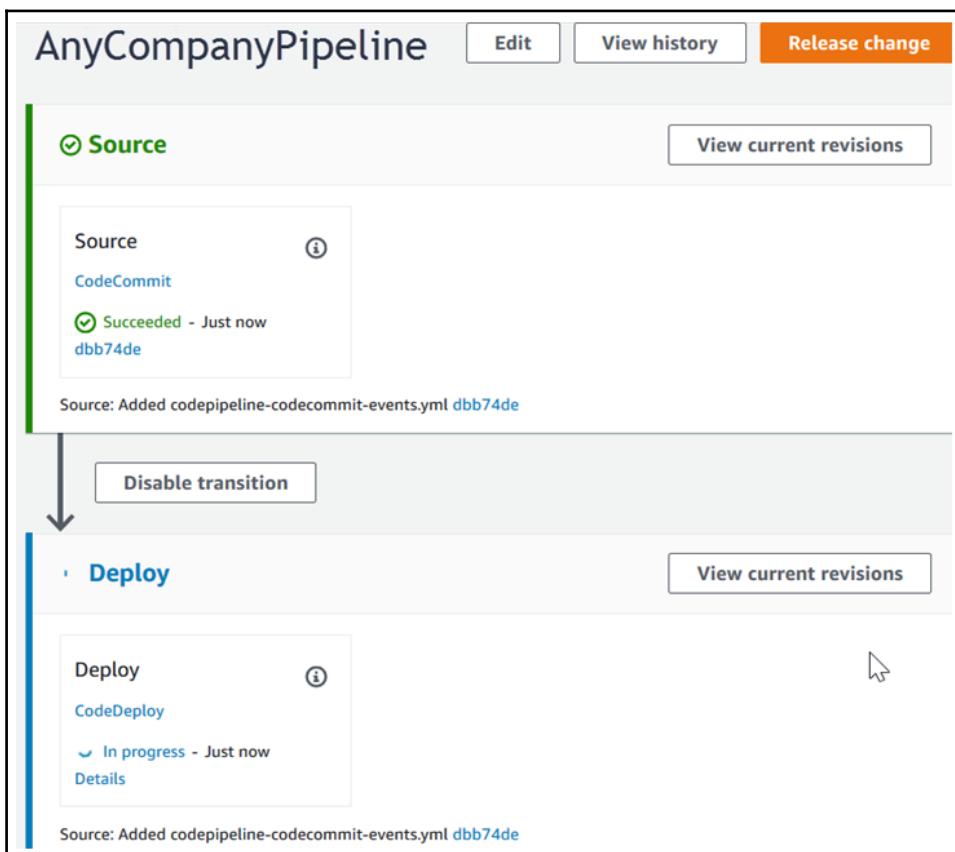
- Every action in a stage has a type, which is based on the action type; it may have one or both of the following:
 - An input artifact: This is accepted by a stage to be processed by the actions in the stage.
 - An output artifact: This is produced by the successful complete execution of an action.

It is important to remember that every output artifact in the pipeline must have a unique name. Additionally, every input artifact name for an action must match with the output artifact name. It is irrelevant whether the action that is using an artifact as an input is the output of the successive stage or an action in a preceding stage. The following diagram is useful for understanding this:



The various steps of the preceding diagram can be explained as follows:

- A revision that is progressing from one stage to another in a pipeline is called a **transition**. In the AWS CodePipeline web console, transition arrows help to connect stages in order of execution. A revision automatically progresses from one stage to another following the successful completion of that stage. In a pipeline, the behavior of the automatic transition of a revision from one stage to another can be changed by enabling or disabling the transition for the relevant stage.
- The approval action prevents the transitioning of the revision from one stage to another. Usually, in CD, approval is given just before the deployment to production stage, where human intervention or an automated test decides when to allow the transition to go further:



- The failure of an action indicates that the stage has not been successfully completed. Even if a single action fails, the revision can't transition from one action to another in the stage. In this case, the stage is considered a failure and no more transitions take place in the pipeline. AWS CodePipeline will pause the pipeline until one of the following occurs:
 - Manually retrying the failed action in a failed stage
 - Starting AWS CodePipeline again from the revision
 - Another revision is made in a source stage action

Summary

CodePipeline facilitates continuous deployment on AWS. It can be used to automate the software deployment process, allowing a developer to quickly model, visualize, and deliver code for new feature updates. This chapter introduced you to CodePipeline and explained how you can use it in your development projects.

In the next chapter, we will come to understand the CI/CD mechanism in AWS.

23 CI/CD on AWS

In today's rapidly growing world, every enterprise faces the challenge of matching its pace with customer expectations and software delivery speed. **Continuous Integration/Continuous Delivery (CI/CD)** is a part of **DevOps** that can help to achieve rapid software changes while maintaining system stability and security.

We will cover the following topics in this chapter:

- Understanding CI/CD
- AWS tools for CI/CD

Understanding CI/CD

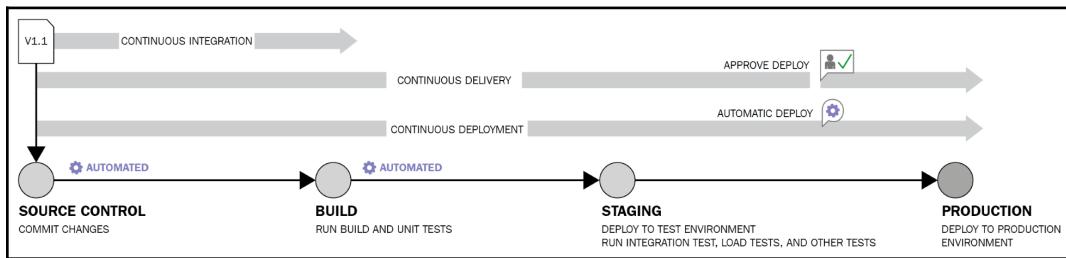
So far, we have seen various individual AWS web services for DevOps, such as CodeCommit, CodeBuild, CodeDeploy, and CodePipeline. Respectively, these AWS services act as a source code repository and continuous integration service that is responsible for compiling source code, running tests, and producing software packages that are ready to deploy. The deployment service can automate software deployment on various AWS compute services and on-premises servers, and automate the build, test, and deploy phases for the release process.

To implement an Agile process and DevOps culture in the enterprise's IT, manually moving code from development to production may be time consuming and error prone. Also, in the case of failure at any step, it may be time consuming to undo the change and restore the previous stable version. If such an error occurs in a production environment, it may also directly impact the business's availability. On the other hand, we can think of CI/CD as an assembly line. The heart of CI/CD is writing or designing the **CodePipeline**.

Steps can be managed from the very beginning when a new piece of code is merged with the master branch from the development (usually, the master branch in the repository consists of the final code for use in the business) till the last step of deploying the compiled and tested software to the production environment. CI/CD stops the delivery of bad code to the production environment by embedding all the necessary tests, such as unit tests, integration tests, load tests, and so on, in the pipeline.

It is a best practice to carry out such tests on the test environment, which is identical to the production environment, to calibrate and test code more precisely.

Implementing continuous delivery or continuous deployment is one of the options in CI/CD practices where passing builds are deployed directly to the production environment and application changes run through the CI/CD pipeline. Let's look at some basic terminology with the help of the following screenshot:



CI

When CI is implemented, each of the developers in a team integrates their code into a main branch of a common repository several times during a day. AWS CodeCommit or third-party tools such as **GitHub** and **GitLab** can be used as a common source code repository to store developing code. Earlier, in traditional software, the development took place in isolation and was submitted at the end of the cycle. The main reason for frequently committing code several times to a main branch of a common repository is to reduce integration costs. Since, developers can identify conflicts between the old and the new code at an early stage, it is easy to resolve these conflicts and less costly to manage.

Any time developers commit a new code to the common branch at the common repository, no manual methods exist to make sure that the new code quality is good, doesn't bring any bug to the software, and doesn't break the existing code. To achieve this, it is highly recommended you perform the following automatic tasks every time a new change is committed to the repository:

- Execute automatic code quality scans to make sure the new code adheres to enterprise coding standards.
- Build the code and run automated tests

The goal of the CI is to make new code integration simple, easy, and repeatable to reduce overall build costs and identify defects at an early stage. Successful implementation of CI will make changes in the enterprise and will act as a base for CD. CI refers to the build and unit testing stages of the software release process. Every new release commit triggers an automated build and test.

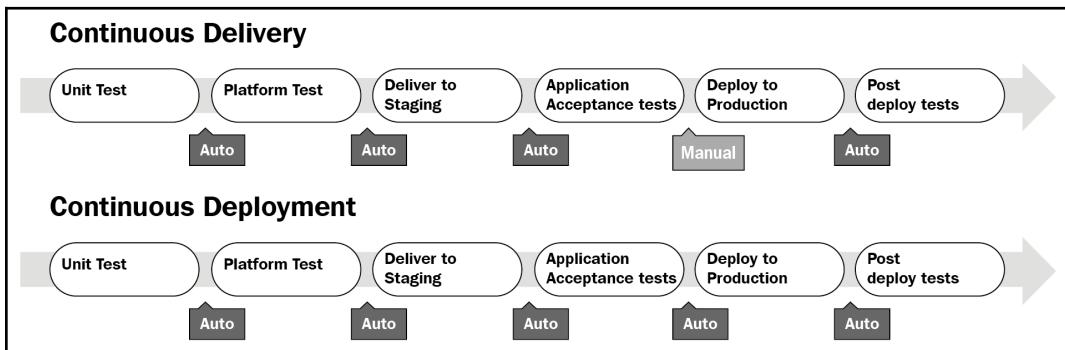
CD

CD is the next phase of CI, and it automates pushing the applications to the various environments, from development to test and pre-production. It is also possible to have more than one development and testing environment for application changes that are staged for various individual tests and review. Even if code has passed all the tests, CD doesn't automatically push the new change to the production. But it makes sure that the code base is deployable and builds are ready to deploy to the production environment at any time. Teams can be confident that they can release the software whenever they need to without complex coordination or late-stage testing. Amazon CodePipeline can be used to build pipelines, and Amazon CodeDeploy can be used to deploy newly built code to the various AWS compute services or in various environments from development to production.

In general, with the help of the CD release schedules (daily, weekly, fortnightly, or monthly) as per the business requirements, new features are deployed to production on these specific schedules. If the business permits, then develop software in small batches, as it is easy to troubleshoot if there is a problem.

Continuous deployment

Continuous deployment is an extension of continuous delivery. Continuous deployment automatically deploys each build and makes sure that the build has passed the full life cycle to the production environment. This method doesn't wait for human intervention to decide whether to deploy a new build to the production environment; as soon as it has passed the deployment pipeline, the build will be deployed to the production environment. The following diagram helps us understand this process:

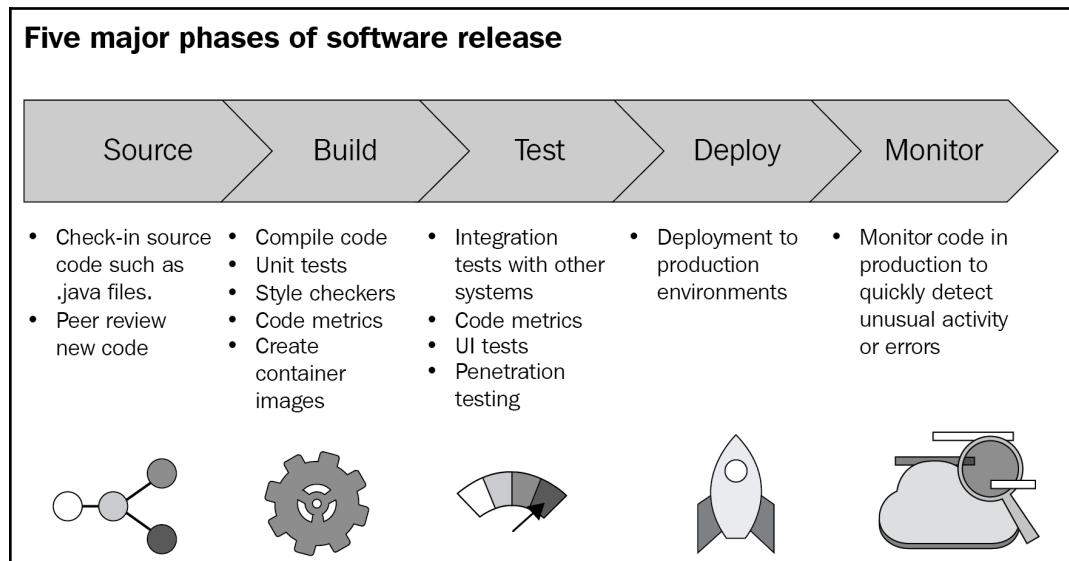


The preceding diagram helps to understand the difference between continuous delivery and continuous deployment. Continuous delivery requires human intervention to finally deploy newly built software to the production. This method may help business people decide when to introduce a new change to the market. On the other hand, continuous deployment doesn't require any human intervention to deploy newly built software that has passed all the tests in the deployment pipeline and will be directly deployed to the production.

Based on the business needs, either of these models can be implemented in DevOps. It is important to understand that at least continuous delivery has to be configured to successfully implement DevOps.

AWS tools for CI/CD

At a higher level, the software development life cycle can be broken down into five steps, which are as source, build, test, deploy, and monitor, as shown in the following screenshot:

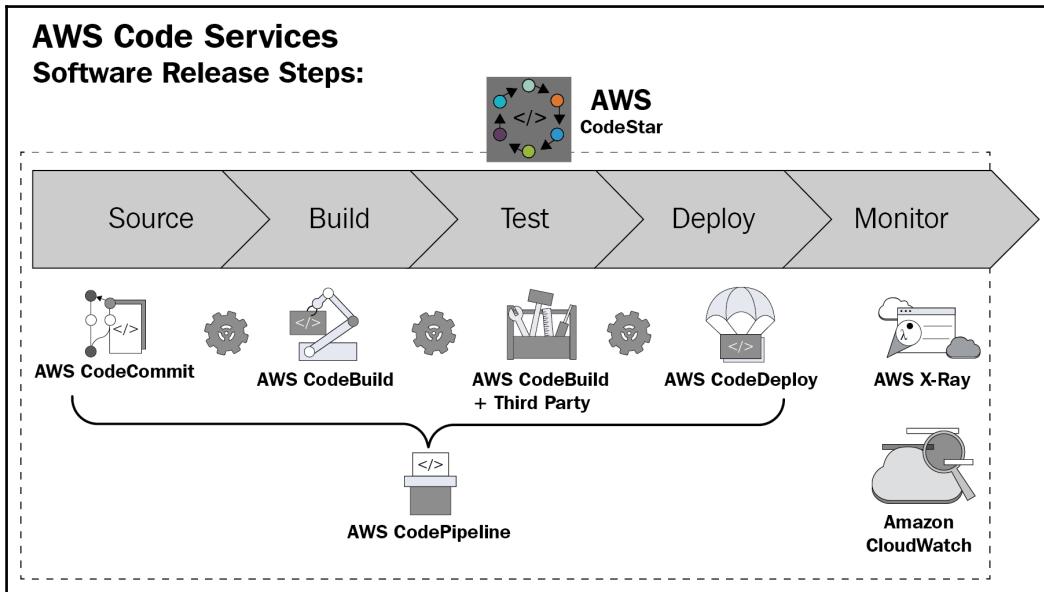


We will go through each of these steps in detail:

- **Source:** At this first stage of the software development life cycle, developers commit their source code many times a day to a common branch in the common source code repository to identify integration errors at the early stages. Once a feature, development, bug fix, or hotfix is complete on the relevant branch, it will be merged with the master branch to release the development into production, passing through various environments, such as development and test, and passing all the tests specified in the CodePipeline.

- **Build:** In an earlier stage, once development is peer reviewed and merged to the production, it usually triggers the build process. The source code will be fetched from the specified source code repository and branch, and the code quality will be checked against enterprise standards. Once the code quality is accepted, the code is ready to compile. If the source code is written in a compiler programming language such as Java, C/C++, or .NET, it will be converted to binary. This binary output is also called **artifacts** and is stored in the artifacts repository. The Docker container files are converted into the Docker container image and are stored to the container image repository.
- **Test:** At this stage, artifacts are fetched from the artifacts repositories and deployed to one or more test environments to carry out one or more tests, such as load testing, UI testing, penetration testing, and so on. In general, it is recommended to have a test environment that's identical to the production environment in order to carry out precise tests.
- **Deploy:** Once source code has been successfully passed through all the build and tests phases that are defined in the CodePipeline, it is ready to be deployed into production. In the case of continuous delivery, the source code will wait for human intervention before it is deployed into the production environment at the pre-defined released date. If continuous deployment has been configured, then it doesn't matter that an output artifact of one action is used in consecutive stages or some future stage. The code will be deployed to production automatically.
- **Monitor:** Once the revised software has been deployed to production, it is time to monitor the application and infrastructure resources for the errors and resource consumption respectively.

Now let's see the same software development life cycle with AWS services:



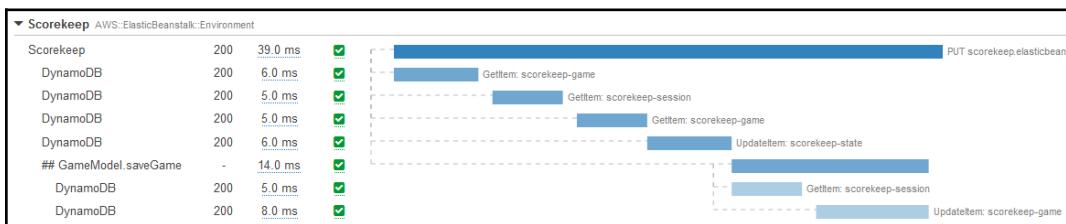
Looking at the preceding diagram, AWS services can be explained as follows:

- **AWS CodeCommit:** This is a fully managed version control source code repository to host private Git repositories. It is well suited for most of the version control needs in enterprises. It doesn't have any limitations on file size, file type, or repository size. More details on this AWS service are available in Chapter 28, *Exploring AWS CodeCommit*.
- **AWS CodeBuild:** This is a fully managed continuous integration service that compiles source code, runs tests, and produces software packages called artifacts. Artifacts are ready to deploy to production. It builds automatically to support running multiple builds concurrently. Each commit or merge to a master or configured branch will trigger an individual build process. More details on this AWS service are available in Chapter 20, *Working with AWS CodeBuild*.
- **AWS CodeBuild + third-party tools:** AWS CodeBuild can be used with third-party tools to satisfy custom build and test requirements.

- **AWS CodeDeploy:** This is a fully managed deployment service that automates software deployment to AWS compute services such as Amazon EC2, AWS Fargate, AWS Lambda, or on-premises servers. This service eases the frequent release of new application features by managing the complexity of updating applications. It eliminates the time-consuming and error-prone process of updating applications. It works seamlessly with various configuration management, source control, CI, continuous delivery, and continuous deployment tools. More details on this AWS service are available in Chapter 21, *Getting Started with AWS CodeDeploy*.
- **AWS CodeStar:** This combines various DevOps offerings from AWS web services under one roof, such as AWS CodeCommit, AWS CodeDeploy, AWS CodeBuild, and AWS CodePipeline. Each of these AWS web services can be used individually from their web console. But rather than jumping from one web console to another and dealing with individual resources in each of these AWS web services, AWS CodeStar provides a seamless experience by combining all the DevOps services in a single place as a project.

CodeStar provides a ready GUI-based step-by-step template to quickly and easily deploy DevOps for a new project. At the time of writing this chapter, CodeStar provides a sample template for the web application, web service, Alexa skills, Static website, and AWS Config Rule using C#, Go, HTML 5, Java, Node.js, PHP, Python, or Ruby to deploy on AWS Elastic Beanstalk, Amazon EC2, or AWS Lambda.

- **AWS X-Ray:** This service provides a platform to view, filter, and gain insights into the collected data to identify issues and optimization opportunities. AWS X-Ray helps developers to analyze and debug applications using distributed tracing. In a nutshell, this service helps us to understand the root cause of issues, errors, and bottlenecks in the application. The following screenshot is an example of application data collection and reporting:



- **Amazon CloudWatch:** This is a service that monitors AWS resources and customer applications running on AWS in real-time. It is automatically configured to provide some default metrics for each AWS service. These metrics vary for each AWS service and resource type. It collects monitoring and operational data in the form of logs, metrics, and events. More details on this AWS service are available in [Chapter 7, Monitoring with CloudWatch](#).
- **AWS Cloud9:** This is a rich cloud-based **Interactive Development Environment (IDE)** for writing, running, and debugging code in the web browser. It also has a built-in terminal preconfigured with AWS CLI. It saves you from installing and configuring an IDE on a developer's local machine. It also sets developers free to use the same IDE from any machine or device through a web browser. It is secured and also has collaboration features.

Summary

CI/CD is a mechanism that optimizes and automates the development life cycle. This chapter introduced you to CI/CD on AWS and described how you can use it with your AWS workloads.

The next chapter introduces you to serverless computing.

24

Serverless Computing

Serverless computing is a mechanism to run applications without provisioning, maintaining, and administering the compute or storage resources. You just need to worry about your application workload without worrying about the servers required to host your application and data.

Wait—*what?* Then *how does the application run without servers?* Technically, there are servers behind the scenes, but you don't need to manage these servers. The cloud service provider dynamically allocates and manages these servers to provide computing and data resources. When you use serverless computing, it reduces operational overhead and cost and increases agility. Developers do not need to worry about the compute and data resources required to run the workloads. An organization or team can focus on core solution development rather than worrying about infrastructure operations.

AWS provides a number of services that can help you build serverless applications. These AWS services are highly available and provide large-scale support for running enterprise-grade applications. Some of the AWS services that can be used in serverless architecture are AWS Lambda, Step Functions, S3, API Gateway, CodePipeline, CodeBuild, SNS, SQS, X-Ray, Fargate, DynamoDB, Aurora Serverless, Kinesis, Athena, IAM, and Cognito.

If you haven't already referred to some of these services in previous chapters, you may read about Lambda, S3, CodeBuild, CodePipeline, SNS, SQS, DynamoDB, and IAM in the previous chapters of this book.

The following topics will be covered in this chapter:

- Recapping AWS Lambda
- Overview of API Gateway
- Understanding step functions
- Amazon Cognito
- Amazon Cognito Sync

This chapter will also go through an example of building a serverless website using AWS services.

Recapping AWS Lambda

When we talk about serverless computing on AWS, the first service that we think of is Lambda. AWS Lambda is a **Function as a Service (FaaS)**. You can write your business application logic in one or more Lambda functions.

Lambda supports multiple programming languages that you can use to write functions. Lambda natively supports Node.js, Python, Java, Ruby, C#, Go, and PowerShell. You can write functions in either of these languages using Lambda. Though native languages in which you can write Lambda functions are limited, Lambda also provides Runtime APIs, which enables you to write functions in virtually any language and use them in Lambda. You can refer to Chapter 17, *Overview of AWS Lambda*, of this book to learn more about Lambda.

An overview of API Gateway

API Gateway is a fully managed service by Amazon that can be used to create secure REST and WebSocket APIs. Using API Gateway, developers can easily create and manage their APIs at any scale.

An API is a way to distribute business logic in a set of clearly defined methods. APIs simplify communication between two or more application components and enable the developers to achieve greater functionality. API Gateway can accept and process thousands of concurrent API calls. API Gateway can authorize users, handle traffic, and can also monitor the performance of the APIs.

Things that API Gateway can do for you

API Gateway can take care of the following things for you:

- **API creation and deployment:** You can easily create an API in API Gateway and point it to the underlined business logic either in a Lambda function or in an application endpoint hosted on EC2 instances, an application hosted on Elastic Beanstalk, or any other on-premises or cloud-based compute engine where the application functions are hosted.
- **Handle REST APIs and WebSocket APIs:** API Gateway allows you to create REST APIs that can handle HTTP requests or WebSocket APIs that can handle bidirectional communication between applications.
- **Request throttling:** API Gateway allows you to set rules for each of the HTTP methods that you define in your API and restrict the number of requests per second it can handle. For example, you can set a threshold on number of HTTP GET, HTTP POST, and HTTP PUT requests per second for any API created on API Gateway.
- **Caching:** API Gateway enables you to set caching for your APIs. It can cache the response from backend services with time-to-live in number of seconds. When API requests are handled by API Gateway Cache, your backend services are not overloaded with requests. It can re-cache the response after the cache expires based on the stipulated caching time in seconds.
- **API versioning:** API Gateway allows you to maintain multiple versions of the same API. This means you can easily support the backward compatibility of your application programs. While one set of users who would have migrated to latest version of your application, the user can use the latest version of your API and another set of users who may be still using older version, can be directed to older version of the API. This provides great level of flexibility to software developers.
- **Create and distribute your own SDK:** Using API Gateway, you can easily create your own **Software Development Kit (SDK)** and distribute it to developers who may want to use your APIs for development. With API Gateway, you can create client SDKs for Java, JavaScript, Java for Android, Objective-C/Swift for iOS, and Ruby.

- **API Monitoring:** API Gateway provides a dashboard that provides a visual representation of the calls made to your APIs. API Gateway is integrated with CloudWatch, which enables it to provide a number of metrics such as calls made to APIs, latency, and error rate. You can also create custom metrics for your APIs and use them as needed. API Gateway also uses CloudWatch to store call logs and execution errors.
- **Request authorization:** API Gateway can authorize API requests using the Signature Version 4 Signing Process. With Signature Version 4, you can use IAM access policy to authorize API requests. It also allows you to use the Lambda function to authorize bearer tokens such as JSON Web Tokens and/or SAML assertion.
- **API keys:** API Gateway provides API keys that can be distributed to third-party developers. You can create API keys for each of your APIs and control the access policies associated with them. Each API key can be associated with a throttling plan, which can restrict the key to use the APIs up to a limited number of requests.

How API Gateway works

Now we know what API Gateway is and what features it provides, but *how exactly does it work?* The following diagram shows an API Gateway architecture and how it works:



Figure 25.1: How API Gateway works

The working of API Gateway are explained as follows:

1. A user or an application initiates a request to an API endpoint. The request may be initiated by a web application, mobile application, or any other IoT device.
2. API Gateway receives the request, logs the call to CloudWatch, and either serves the request from the API Gateway Cache or directs the request to the respective application endpoint, such as a Lambda function, Amazon EC2 instance, or any other endpoint where the business logic is hosted. API Gateway can optionally authorize the request using a Signature Version 4 signing process and IAM access policies.
3. AWS Lambda, or wherever the business logic is hosted, serves the request and responds.

Understanding step functions

AWS Step Functions is a managed AWS service that enables you to create a set of functions using a visual workflow. Step Functions allows you to execute functions in a specific sequence or based on a specific condition. For example, you can execute functions *A*, *B*, *C*, and so on in a sequence. Alternatively, you can define a condition wherein function *B* can be executed depending on the outcome of function *A*. In short, Step Functions allows you to create a defined set of functions that execute in defined steps based on a sequence or outcome of functions.

There are many use cases of Step Functions. Some common use cases are as follows:

- Data consolidation, data processing, and report generation
- Automation of DevOps and maintenance tasks
- Automation of order fulfillment and order processing tasks in eCommerce

- Automation of user registration, subscription and payment workflow:

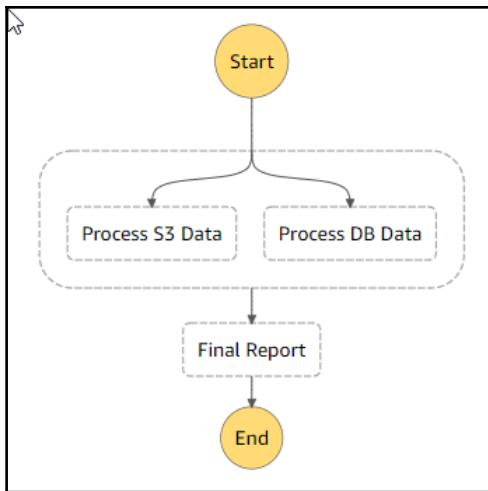


Figure 25.2: Step Functions use case

The difference between Step Functions and a Lambda function

In many use cases, when you have a set of application tasks to run in a serverless mode, you can create Lambda functions for each of the tasks. Step Functions can orchestrate these Lambda functions in a defined workflow. Step functions eliminates the need to write the logic to orchestrate the tasks.

The difference between Step Functions and SWF

Step functions can be differentiated from AWS SWF in the following ways:

- Step Functions is a fully managed service; tasks in Step Functions run in serverless services such as Lambda, AWS Batch, and ECS. You need not worry about the underlying infrastructure, whereas SWF can run on EC2 instances, on-premises servers, and other infrastructure that you need to manage.

- Step Functions flow logic is defined within the service, and respective tasks are executed in the integrated service, whereas in SWF, tasks are executed by the activity worker and flow logic is defined in the decider, that is a part of the application.
- Step Functions can be integrated with any process/application that has an HTTP endpoint and can provide input/output to Step Functions. SWF workflows can only be defined in application code, following a strict SWF coding requirement.
- You can define Step Functions workflow using a simplified GUI, whereas SWF workflows have to be defined in the application code only.

How Step functions works

To understand precisely how Step Functions works, we need to understand some of the important concepts of Step Functions, which are as follows:

- The Step Functions service is built on the idea of tasks and state machines.
- First, you define the state machine using Amazon State Language in JSON.
- Subsequently, when you define the structure of your state machine, it is displayed in a graphical view on the Step Functions console.
- Looking at the visual workflow on the console, you can validate your state machine's logic and can easily monitor its execution.

Understanding states

A **state machine** is a logical model that defines the flow of steps to be executed and stores the status of the tasks. In a state machine, the path of the execution is decided based on the conditions defined in the flow. It is represented using a state diagram. You can use **Amazon States Language (ASL)** to define a state machine in Step Functions.

There are two types of state machine, which are as follows:

- **Finite state machines:** A finite state machine comprises a finite number of states.
- **Infinite state machines:** There is no practical use for an infinite state machine.

In a state machine, different states can make different decisions based on what input is supplied to it. It performs defined actions and subsequently passes the output to other states in the machine.

A state in a state machine can do a number of things. A state's type is determined based on the activity it does:

State type	Activity
Task	Can perform a job in your state machine
Choice	Can decide the path of execution based on the logic
Fail or succeed	Can end the execution with a pass or fail state
Pass	Can directly pass its input as an output or add additional fixed data to it
Wait	Can delay the execution for a specified amount of time or until a specific time/date
Parallel	Can initiate execution of multiple branches together

The following code block shows an example state named `DemoState` that executes a Lambda function:

```
"DemoState": {
  "Type": "Task",
  "Resource": "arn:aws:lambda:us-
west-1:210987654321:function:DemoFunction",
  "Next": "AfterDemoState",
  "Comment": "Execute Demo Lambda function"
}
```

Commonly used state fields

Each state contains some fields that define the state. Some of the fields are given in the following table:

Field	Description
Type	It's mandatory for every state to have a <code>Type</code> field that indicates what type of state it is.
Resource	Indicates the resource to be executed.
Next	This is optional; it defines what state is to be executed next.
End	Instead of <code>Next</code> , a state can also use <code>End</code> with a value as <code>true</code> ; <code>End</code> indicates that there is no state to execute after this.
Comment	Optional comments on the state.

Tasks

The role of a task is to let state machines perform all the work. It performs work by executing a Lambda function or calling other APIs by passing parameters. Step functions can directly call a Lambda function from a state.

Step functions can integrate with some of the AWS services from within a task. It provides you with the ability to initiate actions with these services using **Amazon States Language (ASL)**.

The following are some of the examples of service integration with Step Functions:

- Invoke Lambda functions
- Manage Amazon DynamoDB tables
- Run an **Elastic Container Service (ECS)** operation
- Manage a job in SageMaker
- Call AWS Batch jobs
- Trigger a job in Glue
- Push a topic in SNS
- Pass a message to the SQS queue

In the previous example of `DemoState`, you can observe that the state type is `Task` and it's calling a Lambda function named `DemoFunction` with the following Resource field:

```
"Resource": "arn:aws:lambda:us-
west-1:210987654321:function:DemoFunction"
```

Creating a state machine

AWS Step Functions makes it easier to create a state machine using a GUI console and ASL. The following example shows how you can create a sample state machine:

1. Log into AWS Console, navigate to <https://console.aws.amazon.com/states/>, and click on **Get started**.

2. On the subsequent screen, you will see three options, as shown in the following screenshot. You can either create your own code snippet, use one of the sample projects, or choose from the existing templates to create a state machine of your choice. To follow this example, select **Author with code snippets**:

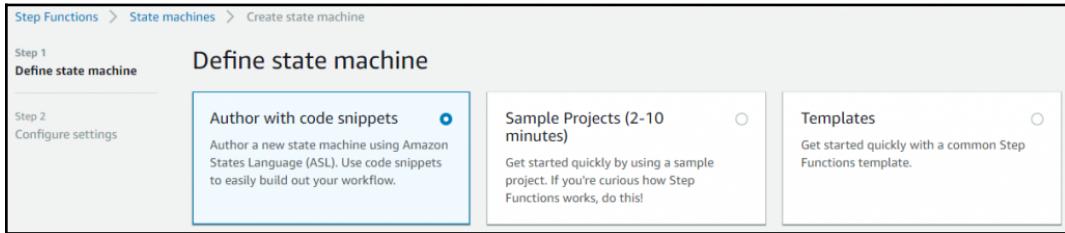


Figure 25.3: Defining the state machine

3. Remember to name the state machine in the **Name** field.
4. In the state machine definition, you can choose one of the existing code snippets from the drop-down menu for various service integrations. Alternatively, you can create a new code snippet from scratch. For this example, you can use the following code snippet:

```
{
  "StartAt": "DemoState",
  "States": {
    "DemoState": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-
west-1:210987654321:function:DemoFunction",
      "Next": "AfterDemoState",
      "Comment": "Execute Demo Lambda function"
    },
    "AfterDemoState": {
      "Type": "Pass",
      "Result": "Demo Completed",
      "End": true,
      "Comment": "End of demo state"
    }
  }
}
```

5. Click on the Refresh icon in the top right corner of the screen, and you will be able to see the following on the screen:

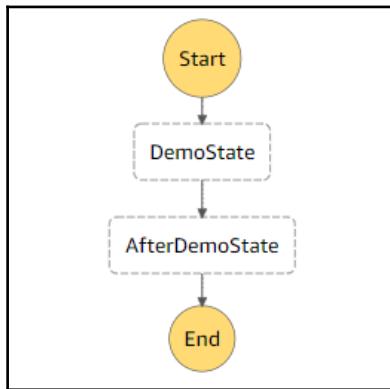


Figure 25.4: State machine definition – graph

6. Click on the **Next** button.
7. On the subsequent screen, you can either create a new IAM role, select an existing IAM role from the drop-down list, or enter the ARN of the IAM role.
8. Select **Create State Machine**.

Congratulations! You have just created your first state machine.

Amazon Cognito

When you start developing applications, one of the most critical aspects of your development is user management. You need a secure and effective user management, authentication, and authorization mechanism. To address this requirement for an application developer, Amazon has created a service called **Amazon Cognito**.

Amazon Cognito is a cloud-based, secure, and highly scalable authentication, authorization, and user management service for web and mobile apps. Cognito not only can authenticate and authorize users, it can also save user data such as user preferences, sign-in, session state, and so on. It can also synchronize the user data across a user's multiple devices to provide a consistent user experience.

Cognito primarily has two components:

- User pool
- Identity pool

Let's understand these components in subsequent points.

Cognito user pool

Cognito user pool makes it easier for a new user to sign-up and existing users to sign into mobile and web applications for authentication, authorization, resource access, and control. It creates and maintains a user directory. A user directory can be imagined as a database of users. The important features of a user pool are as follows:

- Provides sign-up and sign-in services for the mobile and web applications.
- Provides sign-in using any **Security Assertion Markup Language (SAML)** and **OpenID Connect (OIDC) Identity Providers (IdP)** such as Amazon, Google, and Facebook.
- Manages the user directory and user profile. Whether the user signs in directly or through a third-party, all members of the user pool have a directory profile. It can be accessed through the SDK.
- Embeds security features such as **Multi-Factor Authentication (MFA)** and phone and email verification to prevent account takeover.

As shown in the following screenshot, the first step shows that the user provides credentials from a mobile or a web application to an Amazon Cognito user pool. On providing a valid credential, they receive a JSON token:

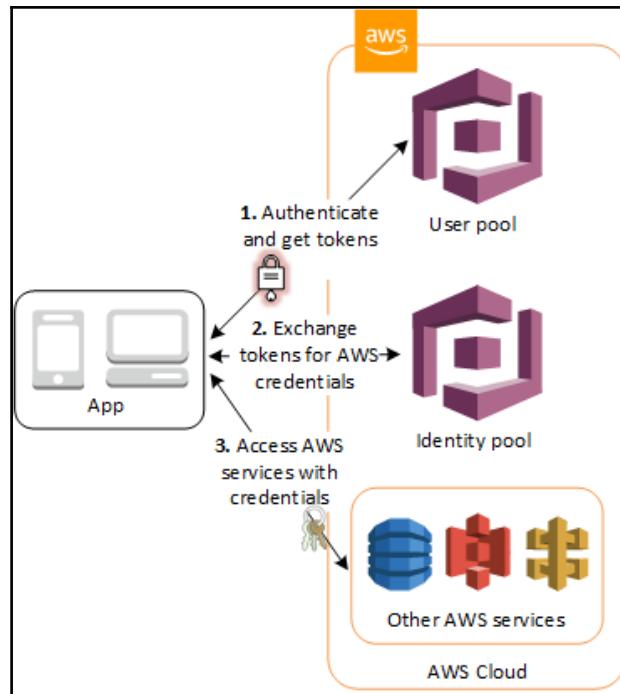


Figure 25.5: Amazon Cognito user pool

Cognito identity pool

Once a user has obtained a valid token from an Amazon Cognito user pool, it can be exchanged with Amazon Cognito identity pools to obtain temporary AWS credentials to access AWS services such as Amazon S3 buckets or Amazon DynamoDB tables. In the previous diagram, *step 2* shows the AWS credentials being obtained against exchanging the JSON token. Finally, it is possible for end users to access AWS resources in *step 3* as per the privileges they are granted. The Amazon Cognito identity pool also allows guest users. Supported users can get authenticated from the following:

- Amazon Cognito user pool
- OIDC
- SAML identity providers
- Developer authenticated identities

As per the project requirements, both the Amazon Cognito user pool and the Amazon Cognito identity pool can be used separately or together in the same mobile or web application. In a nutshell, the Amazon Cognito user pool manages user directories and provides sign-up and sign-in for mobile and web applications, while Amazon Cognito identity pool provides AWS credentials in exchange for a valid token to access other AWS services.

Common Amazon Cognito applications

The following are six common scenarios where Amazon Cognito can be used:

- **Authentication with a Amazon Cognito user pool:** As shown in the following diagram, for a mobile or a web application, users can be authenticated using the Amazon Cognito user pool. These users can be directly from a user pool, or federated through a third-party IdP such as Amazon, Google, Facebook, OIDC, or **Security Assertion Markup Language (SAML)**:

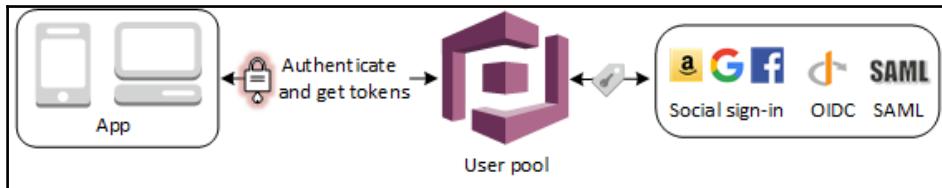


Figure 25.6: Authentication with a Amazon Cognito user pool

- **Access server-side resources with a user pool:** As shown in the following diagram, a user is authenticated when the user has a valid token from the Amazon Cognito user pool. If they are authorized, they can also access the server-side resources. User pool groups in an Amazon Cognito User pool can be created to easily manage the permissions:

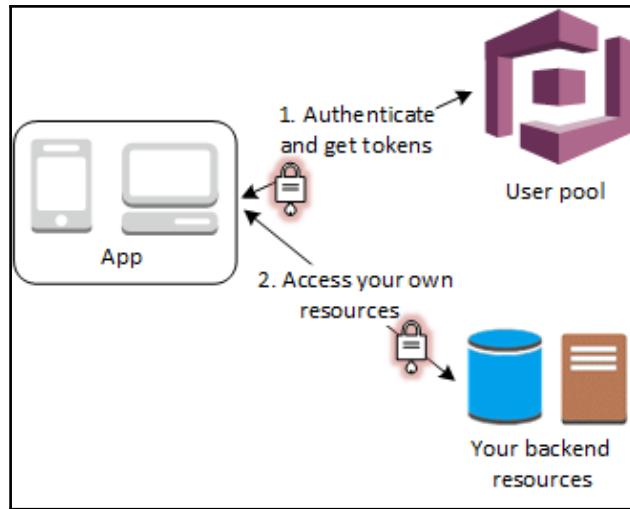


Figure 25.7: Access server-side resources with a user pool

- **Access resources with API Gateway and Lambda with a user pool:** As shown in the following diagram, once a user is authenticated and holds a valid token, they can access an API through an AWS API Gateway. AWS API Gateway can validate the user against a valid token and grant access to an API or a Lambda function:

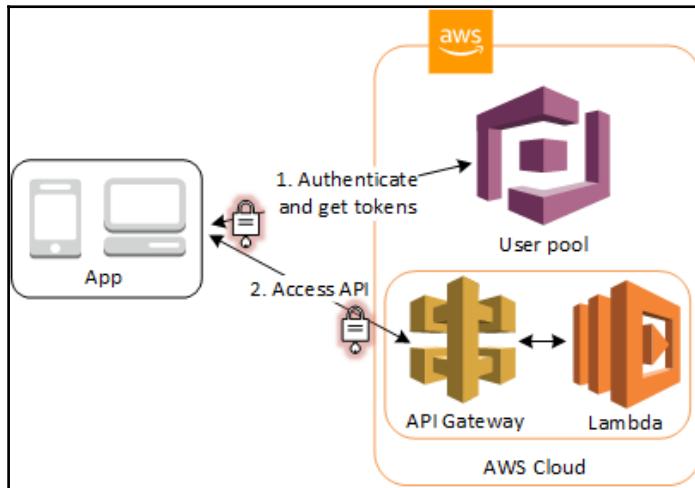


Figure 25.8: Access resources with API Gateway and Lambda with a user pool

- **Access AWS services with a user pool and an identity pool:** As shown in the following diagram, once a user has obtained a valid token from an Amazon Cognito user pool against valid credentials, it can be exchanged with an Amazon Cognito Identity pool to obtain a valid temporary AWS credential. With the help of this valid temporary credential, a user on a mobile or a web application can access various AWS resources:

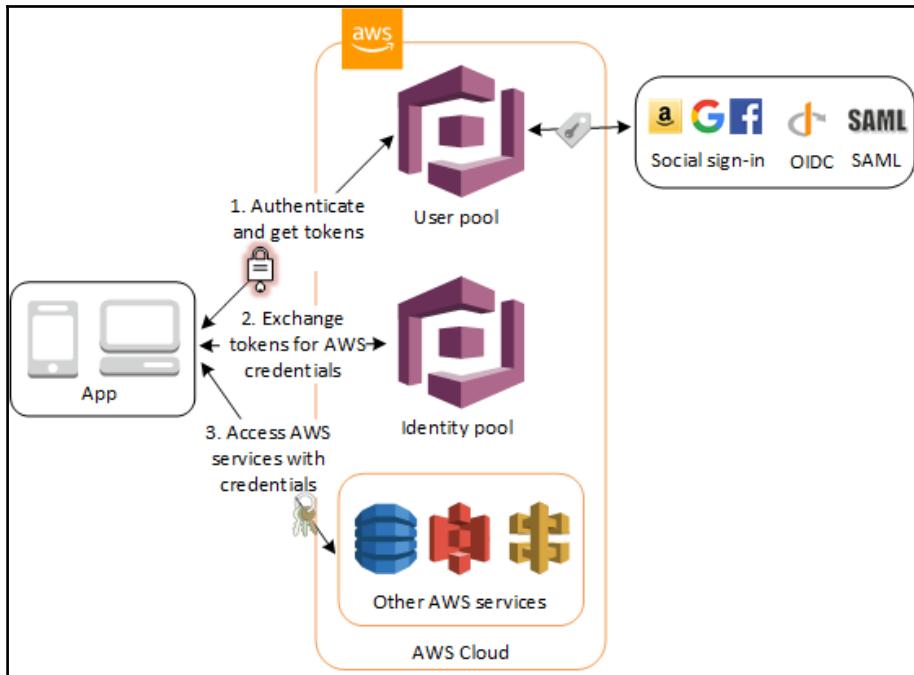


Figure 25.9: Access AWS services with a user pool and an identity pool

- **Authenticate with a third party and access AWS Services with an identity pool:** As shown in the following diagram, an Amazon Cognito user pool can also authenticate a user against a custom identity provider to be able to provide an IdP token. The same token can be exchanged with the identity pool for a temporary AWS credentials to access other AWS resources:

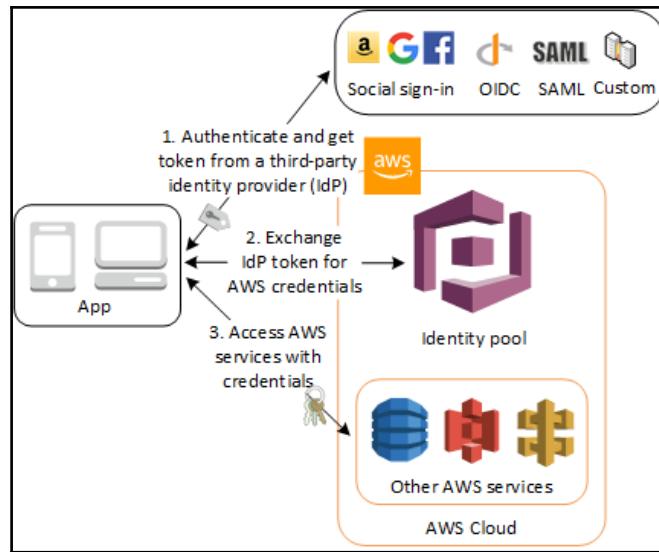


Figure 25.10: Authenticate with a third party and access AWS Services with an identity pool

- **Access AWS AppSync resources with Amazon Cognito:** As shown in the following diagram, against a valid token from a user pool, a user can access AWS AppSync resources:

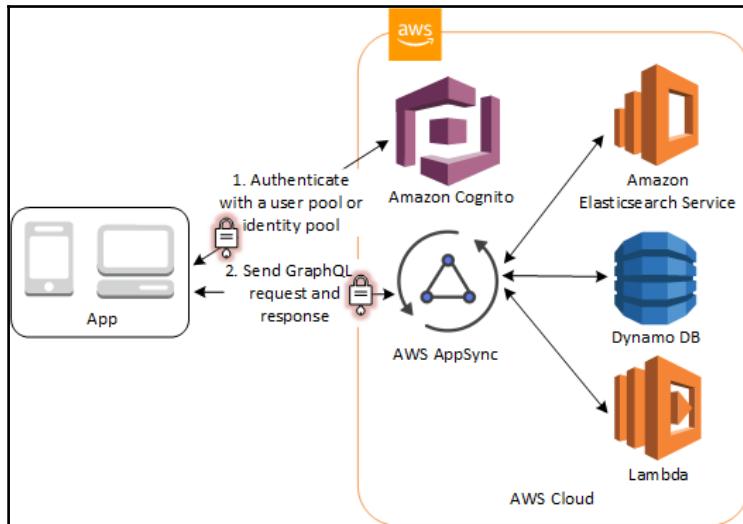


Figure 25.11: Access AWS AppSync resources with Amazon Cognito



More information on this can be obtained at the following URL: <https://docs.aws.amazon.com/appsync/latest/devguide/security.html>.

Amazon Cognito Sync

Amazon Cognito Sync is an AWS web service and client library that makes it possible for an application-related data to cross-device syncing. Developers do not need to implement a backend to synchronize user profile data across mobile devices and the web. A device's (mobile or web) connectivity status, such as online or offline, doesn't matter, as client libraries cache data locally on the device so that applications can read and write data easily. Once the devices are online, the data synchronization can be done manually or using a push sync, and they will automatically and immediately sync the devices once they are online.

Summary

Serverless computing is a mechanism that runs applications without provisioning, maintaining, and administering the computer or storage resources to run the applications. This chapter took you through serverless computing and a number of services that AWS provides to run your workloads in a serverless way. In the next chapter, we will learn about **Amazon Route 53**.

25

Amazon Route 53

Amazon Route 53 is a highly available and scalable cloud **Domain Name System (DNS)** service. This chapter provides an introduction to the service and describes its various components.

The following topics will be covered in this chapter:

- Introduction to Route 53
- Hosted zones
- DNS record types
- Routing policies
- Health checking

Introduction to Route 53

Amazon Route 53 is a reliable and scalable DNS web service. DNS is an internet protocol, and translates human-readable names, such as `www.example.com`, to numeric IP addresses (IPv4 or IPv6) that computers use to connect with each other. In other words, it helps end users (humans or programmatic) to route to the hosted applications over the internet.

At the time of writing, Amazon Route 53 mainly provides the following three features:

- Domain name registration (`www.example.com`)
- Resource health check (usually EC2 instances)
- Routing internet traffic to the applications hosted over the internet with public IPs (IPv4 or IPv6)

Working with Route 53

Before we start working with Amazon Route 53, an understanding of how DNS works is beneficial. You can easily understand how Amazon Route 53 helps in resolving a hostname to an IP address, illustrated in the following diagram. It explains how the service works at a very high level:

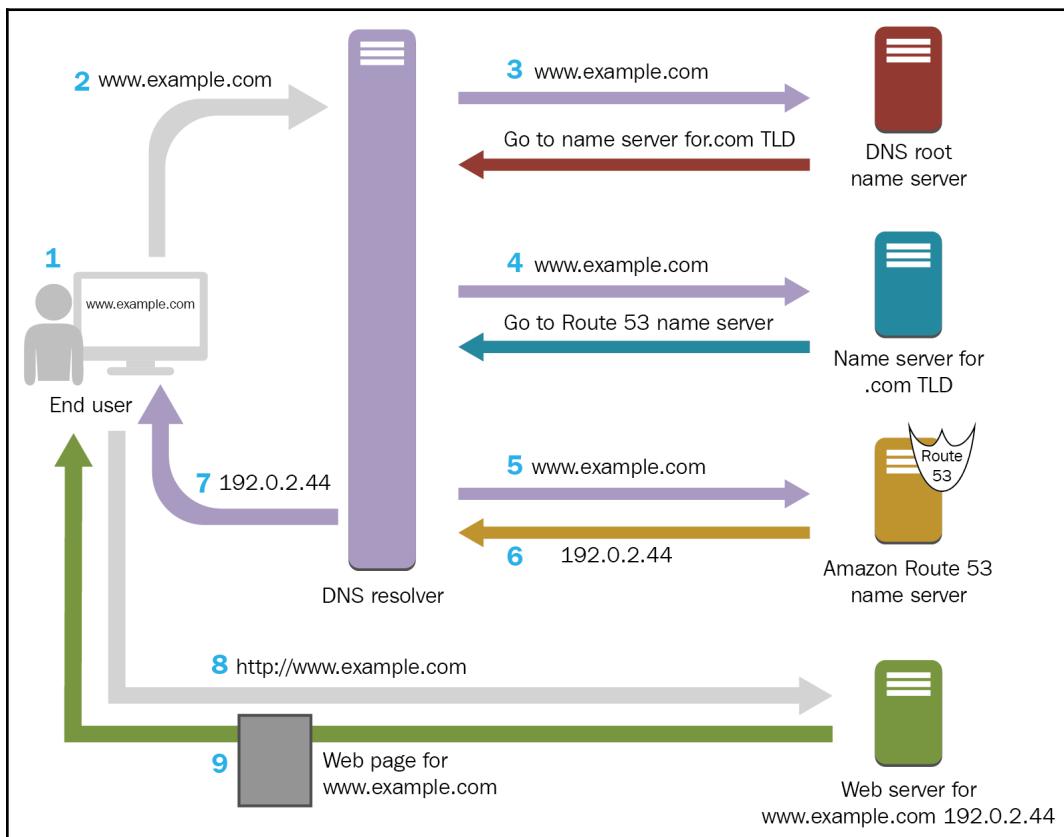


Figure 26.1: How Route 53 works

The preceding diagram is explained in detail in the following steps:

1. Usually, the user enters a website to visit in the address bar of the web browser (for example, `www.example.com`). As soon as the user hits *Enter*, the web browser checks its own cache to establish whether the user has already visited this website (that is, the domain). If yes, then there is no need to go elsewhere to find the IP address for the intended website or application to visit. If the IP address is not found in the web browser caching history, the web browser asks the OS, and the OS will check the `/etc/hosts` location to find the IP address for the user's intended website. The following table describes the location of host files in different operating systems:

Hosts' file path	OS
<code>/etc/hosts</code>	Linux
<code>/private/etc/hosts</code>	macOS
<code>C:\Windows\System32\drivers\etc\hosts</code>	Microsoft Windows



The `hosts` file size limit and path may vary according to the OS version and flavor.

- If it is able to obtain the IP address for the desired website, the web browser opens the site directly in the user's web browser.
2. If the web browser cannot resolve the IP address request for `www.example.com`, it requests to a DNS resolver, which is usually situated at the user's **Internet Service Provider (ISP)**. It may be cached in there.

3. If the DNS resolver at the ISP facility does not have the IP address in their records, then, with the help of the recursion process, the DNS resolver tries to resolve the IP address from the DNS root nameserver. Every DNS resolver server at the ISP knows how to contact the root nameserver. The following diagram describes a sample DNS hierarchy:

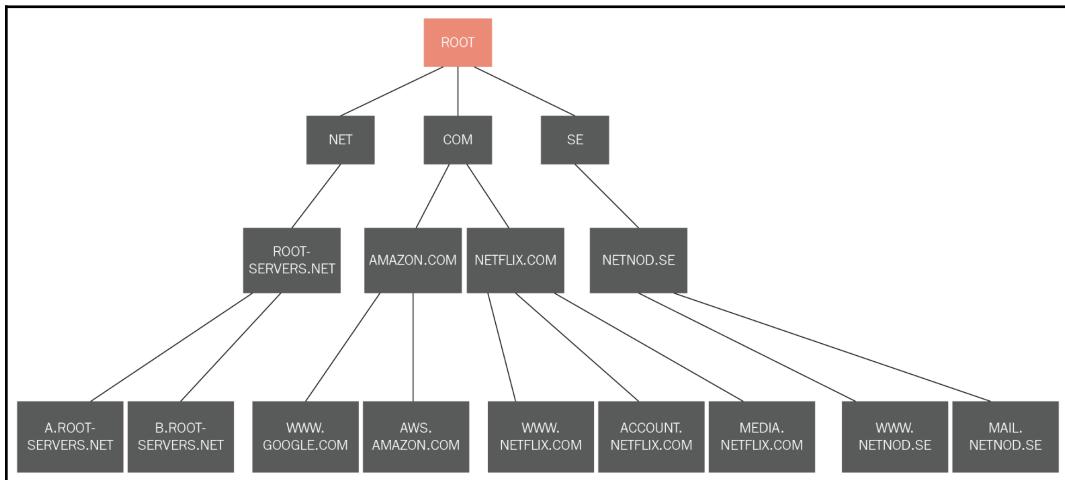


Figure 26.2: DNS hierarchy

As shown in the preceding diagram, the root nameservers are at the top-level DNS zone (also called the **root zone**) in the hierarchical namespace of the DNS. It contains a **root hints file**, which is the global list of top-level domains. The root hints file consists of the names and IP addresses of the authoritative nameservers for each of those top-level domains. These root nameservers are managed by 12 different organizations.

There are more than 750 root nameservers in six continents across the globe that are reachable using IP addresses. Local servers send quick responses to DNS queries that are sent to these IP addresses, as multiple servers around the world are assigned with them. As there are only 13 root server IP addresses, we can see only 13 root servers from a single location at any given point in time.



More details on root servers can be found at <https://www.netnod.se/i-root/what-are-root-name-servers>.

4. The root nameserver answers the request by returning the list of authoritative nameservers for the **Top Level Domain (TLD)** for .com. Authoritative nameservers are just the servers that hold actual DNS records, such as A, AAAA, and CNAME. Later in this chapter, we will look at record types in detail. Any server can be an authoritative server, as long as it holds actual DNS records. In this example, the user is intended to visit a website having a .com TLD. The root nameserver will return the list of .com TLD servers. Now, the ISP's DNS resolver will forward the request to the .com TLD server to obtain an IP address for www.example.com.
5. As the root nameserver guides the ISP's DNS resolver to contact the TLD, it goes to the TLDs to find the IP address for www.example.com. The TLD doesn't know the IP address for www.example.com. But the TLD does know who can provide the IP address for it. This server is called the domain nameserver. In this case, our domain nameserver is route 53. When we register our domain name with Amazon Route 53 or migrate an existing domain name from other domain registrars to Amazon Route 53, the entry in the TLD is updated.
6. The ISP's DNS resolver lands to Amazon Route 53 to get an IP address for the www.example.com domain. It checks the hosted zone for the IP address and gets the associated value. In our case, it will check for the IP address (A or AAAA for IPv4 or IPv6, respectively). For example, in our case, IPv4 is 192.1.3.45.
7. The DNS resolver has the IP address for www.example.com. It returns the IP address to the web browser.
8. The web browser sends a request to the received IP address to visit www.example.com. This IP may be pointing to the EC2 instance, on-premise server, or ELB (if the IP is pointing to the ELB, the case domain name resolver would send the CNAME record rather than the A or AAAA record).
9. These compute resources, such as an EC2 instance, return the web page for the www.example.com web application/website. Usually, all these things are done in the blink of an eye. The DNS concept can also be learned visually from <https://howdns.works>. Said website explains the DNS concept beautifully with less text and more graphics.

Hosted zones

At the Amazon Route 53 web console, under DNS management, we can create hosted zones. These are just logical containers for specific domain records, such as `www.example.com`, and sub-domains, such as `subdomain.example.com`. There are various types of records, and each record has a specific purpose. Later in this chapter, we will see all the supported record types in Amazon Route 53.

A hosted zone for a domain exists on a domain nameserver, usually with the company from where we have registered a domain name. If we have registered our domain name with another service provider, it is possible to migrate it to Amazon Route 53. If you do not wish to migrate your existing domain from an earlier register, you can still create your hosted zones on Amazon Route 53. Once you have created your hosted zones on Route 53, it will create four nameservers, storing all the records (that is, A, AAAA, and CNAME). You can get these NS server record details and register them with your earlier domain register.

In the following screenshot, we can see the basic details for a public hosted zone and nameserver for a domain:

Name	Type	Value	Evaluate Target Health	Health Check ID	TTL	Region
box.com.	NS	ns-652.awsdns-17.net. ns-1441.awsdns-52.org. ns-243.awsdns-30.com. ns-1781.awsdns-30.co.uk.	-	-	172800	-
box.com.	SOA	ns-652.awsdns-17.net. awsdns-hostmaster.amazon.	-	-	900	-

Figure 26.3: Public hosted zone

These basic details are populated by default. We have just created a hosted zone.

There are two types of hosted zones: **public** and **private**:

- **Public hosted zones:** These contain records set to route internet traffic from the end user to your web application, and emails
- **Private hosted zones:** These contain records set to route Amazon VPC traffic within an AWS account

DNS record types

Once a hosted zone with the same name as the domain name has been created, whether in public or in private, it is time to fill it up with the appropriate record types. These record types help traffic (internet or VPC, based in public or private, respectively) to forward the appropriate services. Let's look at each of them individually in detail.

A record type

This record stores IPv4 addresses with a dotted decimal notation. This record helps traffic to a specific web application server on EC2 or on-premises.

For example, an EC2 instance has an IPv4 IP, as follows. A record type for the given domain (`example.com`) can be configured with an IP address, as follows:

```
192.0.2.1
```

AAAA record type

This record stores IPv6 addresses in a colon-separated hexadecimal format. This record helps traffic to a specific web application server on EC2 or on-premises.

For example, an EC2 instance has the IPv6 IP, as follows. The AAAA record type for the given domain (`example.com`) can be configured with an IPv6 address, as follows:

```
2111:0db3:82a3:0:0:8a2e:0370:7334
```

CAA record type

This record helps us by specifying **certificate authorities (CAs)** that are authorized to issue certificates for a domain or subdomain. It helps to prevent a certificate being issued for your domain from the wrong CAs. It is essential to understand that this record type is not a substitute SSL certificate for your website.

For example, to authorize `ca.example.net` to issue a certificate for `example.com`, the CAA record for the `example.com` hosted zone can be created as follows:

```
0 issue "ca.example.net"
```

CNAME record type

CNAME stands for **canonical name**. The value for the CNAME record is the same as the domain name. It should never point directly to an IP address, but should always point to another domain name. It is possible that one CNAME points to another CNAME, but it is not recommended because of the performance issue.

To simplify, a record resolves traffic for the domain name to the IP address, while the CNAME resolves traffic for one domain name (`www.example.com`) to another domain name (`example.com`).

It is very common to find `example.com` and `www.example.com` hosted by the same server and pointing to the same application. In this case, we create the following in order to avoid logging two different records:

- An A record (for example, `.com`), pointing to the server IP address
- A CNAME record for `www.example.com`, pointing to `example.com`

Hence, `www.example.com` points to the same address through `example.com`, while `example.com` points to the server IP address. If the IP address changes, we just need to edit the A record (for example, `.com`) and then, `www.example.com` automatically reflects the changes.

MX record type

MX record stands for the **Mail Exchange** record. The mail server address responsible for accepting email messages is specified by this record type. The email messages are accepted on behalf of the domain name. Several MX records can be configured that point to an array of mail servers for load balancing and redundancy.

For example, each value for an MX record actually contains two values—**priority** and **domain name**. An integer value helps to identify which mail server you want to route emails first. For example, the value for the MX record can be shown as follows:

```
10 mail.example.com
```

In the preceding example, the `mail.example.com` server is considered to be the 10th priority. In the event of having only one record, it doesn't matter; we can give any number between 0 to 65,535.

NAPTR record type

The **Name Authority Pointer (NAPTR)** record type is mostly used by the **Dynamic Delegation Discovery System (DDDS)** application (internet telephony) for converting one value to another value. The NAPTR record consists of six separated values, which are as follows:

- **Order:** This helps to identify the sequence in cases where we have more than one NAPTR record. Valid values are from 0 to 65,535.
- **Preference:** This helps to identify the preference for the sequence to evaluate the record first in case we have two records with the same order. For example, when two or more records have an order of 1, the record with the lower preference evaluates the first.
- **Flags:** This is DDDS applications-specific, according to the RFC 2404. It has uppercase and lowercase letter A, P, S, and U, and the empty string “”. Flags are enclosed in quotation marks.
- **Service:** This is specific to a DDDS application and is enclosed in quotes.
- **Regexp:** The DDDS application uses a regular expression that is provided to convert an input value into an output value. The regular expression is enclosed in quotes. For example, a phone number that is entered by a user into a session can be converted into a **Session Initiation Protocol (SIP)** URI by an IP phone system using regular expressions.
- **Replacement:** This is called as a **fully qualified domain name (FQDN)** of the next domain name that you want the DNS query to be submitted by the DDDS application.

For example, consider the following code

```
100 50 "u" "E2U+sip" "!^(\\"+\+441632960083)$!sip:\\"1@example.com!" .
100 51 "u" "E2U+h323" "!^(\\"+\+441632960083$!h323:operator@example.com!" .
.
100 52 "u" "E2U+mailto" "!^.*$!mailto:info@example.com!" .
```

NS record type

This record is automatically created by Amazon Route 53, with appropriate values, when a hosted zone is created. It contains a domain name of a nameserver.

For example, Amazon Route 53 creates four nameservers, as shown here:

```
ns-652.awsdns-17.net.  
ns-1441.awsdns-52.org.  
ns-243.awsdns-30.com.  
ns-1781.awsdns-30.co.uk.
```

PTR record type

This is also called a **reverse DNS record**. The primary function of the DNS system is to convert a domain name to an IP address, but a PTR record does the opposite; it associates an IP with a domain name.

For example, this record contains a domain name as a value:

```
hostname.example.com
```

SOA record type

SOA stands for **Start of Authority**. When we create a hosted zone in Amazon Route 53, it also creates an SOA record along with an NS record. It identifies the base DNS information about the domain.

For example, consider the following code:

```
ns-652.awsdns-17.net. awsdns-hostmaster.amazon.com. 1 7200 900  
1209600 86400
```

SPF record type

This record helps in identifying the mail servers that are permitted to send an email on behalf of the domain name.

For example, consider the following code:

```
"v=spf1 ip4:192.168.0.1/16 -all"
```

SRV record type

This record type specifies a data-defining location of servers for specified service, such as the hostname and port number.

For example, consider the following code:

```
10 5 80 hostname.example.com
```

TXT record type

This record provides text information about sources that are outside the domain. It can be human or machine readable.

For example, consider the following code:

```
"This string includes \"quotation marks\"."  
"The last character in this string is an accented e specified in octal  
format: \351"  
"v=spf1 ip4:192.168.0.1/16 -all"
```

Routing policies

To resolve and return the IP address of the server that has the least latency to the client from the list of servers, while creating a record, it is essential to select an efficient routing policy. There are various routing policies available. Let's look at each of them:

- **Simple routing policy:** This is one of the most common and simple DNS routing policies. It can have a FQDN or IP address as a value. Usually, to implement this routing policy using an IP address, use an A or AAAA record (based on IPv4 or IPv6 IP, respectively) or CNAME when using an **Elastic Load Balancing (ELB)** endpoint as a point of contact.
- **Failover routing policy:** This routing policy serves traffic from one resource (primary server or location) when it is healthy and, if the primary resource is down, it will serve traffic from another resource (secondary server or location). It is very useful when implementing a disaster recovery site.

- **Geolocation routing policy:** This routing policy allows the accessing of web applications based on the visitor's geolocation. It allows you to specify geographic locations by continent, country, or states in the US. This routing policy can be very useful for multinational companies, such as online sellers, to serve an application that is compliance-enabled to match their local rules and regulations. It allows multiple records to be defined with the same FQDN.
- **Geoproximity routing policy:** This routing policy makes it possible to route traffic based on the physical distance between a visitor and hosted web application. It is also possible to route more or less web traffic to each of the resources by specifying a positive or negative bias. It allows you to define values, either in AWS regions (when using resources from the AWS cloud), or the latitude and longitude for each endpoint.
- **Latency routing policy:** When you have hosted your application in many AWS regions, to provide a minimum latency, this routing policy serves the IP address of the server that is closer to the visitors' location. For example, if two identical international news websites are hosted in the Oregon and London regions, visitors from Europe will be routed to the London region server, while visitors from the US will be routed to the server in the Oregon region.
- **Multivalue answer routing policy:** This routing policy is just like the **simple routing policy**, but rather than the IP address of one server, it can return multiple IP addresses. It configures more than one resource record for the same FQDN or sub-domain name.
- **Weighted routing policy:** This routing policy makes it possible to route traffic with the same IP address based on the weight assigned in terms of percentage to the record type. For example, you have two different sizes of EC2 instances. One is `c4.medium`, and another is `c4.large`. It is clear that `c4.large` will have double-capacity to serve the traffic compared to `c4.medium`. So, we can stipulate that 33 percent goes to `c4.medium`, and 67 percent of traffic to `c4.large`. On the other hand, it can be also useful in enabling blue-green deployment.

Health checking

For any web application, to create reliability and trust with end users, it is very important to constantly make sure that its resources (web servers) are healthy and able to respond to the visitor's request. Availability of the resource and performance can be monitored using Amazon Route 53's health check. It can perform the following types of monitoring activity:

- **The health of a resource, such as a web server:** It is possible to monitor an endpoint (web server) by specifying the IP or domain name, along with the health check configuration, such as the interval for health checks and the protocol to be used. Internally, Amazon route 53 submits an automated request on the specified endpoint to verify that the application, web server, or other resources are operational.
- **The status of other health checks:** This monitoring and alerting system offers the following two health check mechanisms to failover to alternate endpoints:
 - **Calculated health checks:** This combines the result of multiple Amazon Route 53 health checks into a single value using Boolean operations, such as **AND**, **OR**, and **NOT**. In other words, it creates a health check that combines the result of other checks in a useful way. For example, for a particular web application to meet your user request successfully requires three EC2 instances to be kept running at any given time. So, you can create an individual health check for each of these EC2 servers, and create a calculated health check to represent the overall health of the web application. It can be configured to get an email notification when any of these servers fails to pass the individual health check.
 - **Latency measure:** This makes it possible to measure and create CloudWatch metrics that affect latency, such as the time to the first byte, TCP connection time, and time to complete an SSL handshake. These latency measurements are executed as a part of the health check only.
- **The status of the Amazon CloudWatch alarm:** It is also possible to monitor the status of CloudWatch metrics, such as the number of throttled read events for an Amazon DynamoDB. It helps to improve resiliency and availability, as Route 53 doesn't wait for the CloudWatch alarm to go into the ALARM state.

The following diagram illustrates how the Amazon Route 53 health check works:

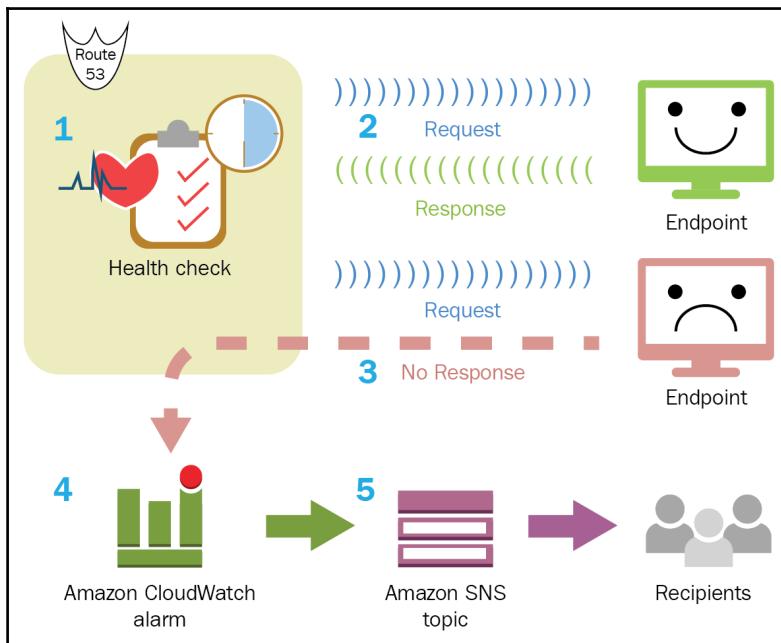


Figure 26.4: Amazon Route 53 health check workflow

The following points, as per the preceding diagram, explain how Route 53 handles health checks:

1. Create a health check with parameters, such as the IP address or domain name for the endpoint to monitor, protocol, health check request interval, failure threshold, and SNS topic to inform the end user regarding the failure of such health monitoring.
2. As soon as the health check is configured, Route 53 starts sending a health check request at the configured request interval. The endpoint is to be considered healthy when it responds to the health check request.
3. For each health check request, it will maintain the count of healthy and unhealthy requests in CloudWatch metrics. Now, when the unhealthy counts correspond to the defined counts for the unhealthy threshold, the endpoint is considered unhealthy. But, in-between, when unhealthy counts are continually increasing, and they are still below the defined unhealthy threshold counts, it will not contact you.

4. If the endpoint is considered unhealthy, it will raise a notification for CloudWatch.
5. Finally, with the configured notification for the health check, CloudWatch triggers an alarm and uses Amazon SNS to send a notification to the specified recipients.

Summary

- Amazon Route 53 is a reliable and scalable DNS web service. DNS is an internet protocol, and translates human-readable names, such as `www.example.com`, to numeric IP addresses (IPv4 or IPv6) that computers use to connect with each other.
- A TLD refers to the highest level in the hierarchical domain name system of the internet (that is, `.com`, `.net`, `.org`, and so on).
- Hosted zones are logical containers for specific domain records (that is, `www.example.com`), and sub-domains (`subdomain.example.com`).
- Public hosted zones contain records set to route internet traffic from the end user to your web application, and emails.
- Private hosted zones contain record sets to route Amazon VPC traffic within an AWS account.
- In Route 53, the A record type stores IPv4 addresses with a dotted decimal notation. This record helps traffic to a specific web application server on EC2 or on-premises.
- In Route 53, the AAAA record type stores IPv6 addresses in a colon-separated hexadecimal format. This record helps traffic to a specific web application server on EC2 or on-premises.
- The CAA record type helps us by specifying CAs that are authorized to issue certificates for a domain or subdomain.
- CNAME stands for canonical name. The value for the CNAME record is the same as the domain name. It should never point directly to an IP address, but should always point to another domain name.
- The MX record stands for the Mail Exchange record. The mail server address responsible for accepting email messages is specified by this record type
- NAPTR is mostly used by the DDDS application (internet telephony) for converting one value to another value.

- The NS record is automatically created by Amazon Route 53 with appropriate values, when a hosted zone is created. It contains a domain name of a nameserver.
- The PTR record is also called a reverse DNS record. The primary function of the DNS system is to convert a domain name to an IP address, but a PTR record does the opposite; it associates an IP with a domain name.
- SOA stands for Start of Authority. When we create a hosted zone in Amazon Route 53, it also creates an SOA record along with an NS record. It identifies the base DNS information about the domain.
- The SPF record helps in identifying which mail servers are permitted to send an email on behalf of the domain name.
- The SRV record type specifies data defining the location of servers for a specified service, such as the hostname and port number.
- The TXT record provides text information about sources that are outside the domain. It can be human or machine readable.
- Routing policy determines how Route 53 responds to queries for DNS records.
- The simple routing policy is one of the most common and simple DNS routing policies. It can have an FQDN or IP address as a value.
- The failover routing policy is used for routing the traffic to a primary resource, and traffic is routed to a failover resource in case there is any issue with a primary resource.
- The geolocation routing policy enables you to serve traffic from the geolocation of the users.
- The geoproximity routing policy routes the traffic based on the physical distance between a visitor and hosted application.
- The latency-based routing policy serves the traffic from a location that provides the least latency to the users.
- The multivalue answer routing policy is similar to the simple routing policy. However, unlike the simple routing policy, it can return multiple IP addresses.
- The weighted routing policy routes server traffic based on the weightage assigned to different endpoints/IPs. For example, you can route 70% of traffic to source A, 20% of traffic to source B, and 10% of traffic to source C.
- Availability of the resource and performance can be monitored using Amazon Route 53's health check.

26

ElastiCache Overview

In today's world, real-time web and mobile apps, such as gaming, media, social, and many other internet-scale applications, need high-throughput and low-latency performance, along with high concurrency. In order to achieve this for read-heavy applications, it is essential to store frequently used data on the memory rather than on a disk. Many a time when an application is hosted on EC2 or on-premises servers, we encounter memory (RAM) scarcity.

When you encounter memory issues, you can neither keep on adding more memory, nor can you keep scaling these servers vertically, as it would be expensive. Similarly, when we scale the environment horizontally, it adds operational overheads and may not necessarily resolve memory issues at the instance level.

This chapter elaborates on how ElastiCache can be helpful in handling more memory needs. The following topics will be covered in this chapter:

- Introduction to ElastiCache
- ElastiCache engine types
- Designing the right caching for your workload

Introduction to ElastiCache

Before we discuss Amazon ElastiCache, let's understand what an in-memory cache is, and where it can be used.

It is advantageous to have a cost-effective, separate, and dedicated mechanism, just to store and access frequently used data that supports redundancy and scalability without any downtime. It not only helps to achieve minimal latency for the end user, but also reduces the load on data storage.

The implementation of in-memory caching in the solution architecture is shown in *Figure 27.1*. In the diagram, in-memory caching sits in between the actual data store, such as the database or static content (S3). The end user initiates a web client/application request to the server through the load balancer. The web/app server routes the data read request to the cache cluster. If the data is already cached, the request response is sent back. If the data is not available in the cache, the cache cluster reads the data from the data source and caches it in the memory:

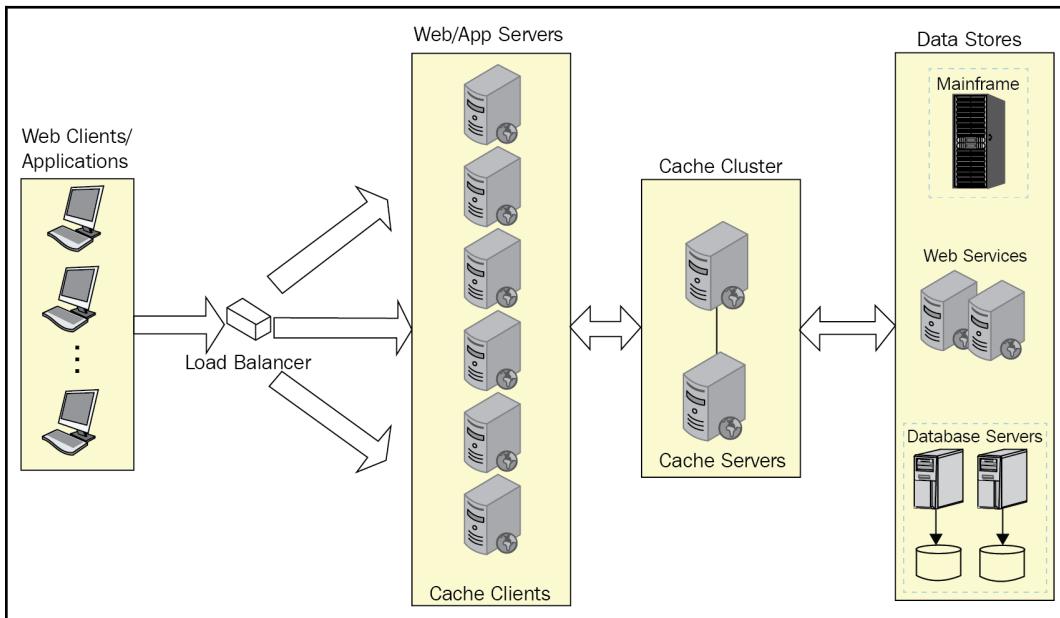


Figure 27.1: In-memory caching

Some of the scenarios where in-memory caching can be used are as follows:

- Unchanged data to be displayed frequently on each web page load or refresh.
- Frequently used database queries.
- Slower and expensive queries.
- Most frequently accessed data and access patterns. The maximum advantage of caching can be obtained for static and frequently accessed data. Caching such data can provide better access latency, and it is also more cost-effective.

- Stale data storage. Stale data is nothing but the data that is not a recent version of committed data. For this, it is essential to identify whether the application is able to tolerate stale data or not, while also serving its prime purpose.

Amazon ElastiCache fits the bill exactly, and provides an in-memory cache that can serve the purposes mentioned in the preceding points.

Amazon ElastiCache is a scalable, reliable, and fully managed in-memory data store and caching service that improves the performance of your application by serving the data from the in-memory cache, instead of retrieving it from the source data store.

ElastiCache runs a cluster on an Amazon **Virtual Private Cloud (VPC)**, which helps to achieve compliance and the implementation of enterprise security standards. It supports two different engine types, which are described in the following section.

ElastiCache engine types

Amazon ElastiCache offers two open source caching mechanisms, **Redis** and **Memcached**. Existing applications working with open source versions of Redis and Memcached can run with Amazon ElastiCache with almost no modification. These both offer a level of high performance, but it is essential to understand the differences between them in order to choose the correct engine to meet our needs.

Amazon ElastiCache for Memcached

Compared to open source Memcached, Amazon ElastiCache for Memcached has several cutting-edge and enhanced reliability features, which make it more favorable to use in the cloud for a production load:

- It can automatically detect node failure and recover accordingly.
- It supports automatic node discovery, as it saves developers from making any changes to the application when a node has been added or removed in the cluster.
- For higher availability, and to avoid a single point of failure, nodes can be deployed in a multi-AZ.

- It can be easily and swiftly integrated with other AWS services, such as Amazon EC2, Amazon CloudWatch, AWS CloudTrail, and Amazon SNS, to design a secure, high-performance, and managed in-memory caching solution.

Let's have a look at some of the features and important components of ElastiCache for Memcached, which are as follows:

- **ElastiCache nodes:** These are the smallest possible elements in the ElastiCache cluster. They are nothing but a fixed-size chunk of secure network-attached RAM that run an instance of Memcached. They can be in relationships with other nodes, or they can be isolated. The number of nodes can be easily scaled up or down in the cluster, along with the DNS name and port, as they all are of the same instance type. There is a soft limit of 20 nodes in the Memcached cluster, or, in total, 100 nodes in the AWS region. One of the key features of the Memcached engine is that it supports **Auto Discovery**. As the application connects to the cache cluster endpoint, it gives a client application the ability to automatically identify the number of nodes in the cache cluster, and to initiate connections to all of these nodes.
- **ElastiCache for Memcached clusters:** These are a logical grouping of one or more ElastiCache nodes. Caching data is automatically partitioned across the nodes in a Memcached cluster. A Memcached cluster looks like the following diagram:

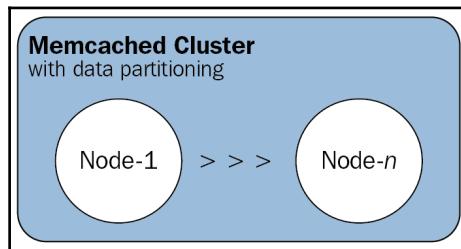


Figure 27.2: Memcached cluster

It is best practice to deploy a caching cluster in multi-AZ, in order to make it more fault-tolerant. It is highly recommended that we use a consistent hashing mechanism while partitioning the data.

- **ElastiCache for Memcached endpoints:** These are unique addresses (URL-CNAME) for the caching cluster. There is one configuration endpoint for a cluster, and there can be one or more node endpoints. In general, when Auto Discovery is enabled, and the application uses a configuration endpoint to transect data with the caching cluster automatically, it will come to know about all the nodes in the caching cluster. Also, for each node, individual node endpoints are also available.
- **ElastiCache parameter groups:** These provide engine-specific parameter settings to fine-tune the caching engine, such as controlled memory usage, eviction policies, and item sizes. These help to make sure that the configuration of each node in the caching cluster is exactly the same.
- **ElastiCache security:** By default, ElastiCache clusters are launched in an Amazon VPC. It shapes and separates network traffic only for the whitelisted applications (EC2 instances), in order to access the caching cluster with the help of subnets.
- **ElastiCache security groups:** These are only applicable to the clusters that are not running in an Amazon VPC. They act as a firewall and help to control the caching cluster access only from the whitelisted (desired) EC2 instances. By default, for security groups, the `AuthorizeCacheSecurityGroupIngress` parameter is disabled. To enable a security group to access the caching cluster (network), we need to use the appropriate API or CLI.
- **ElastiCache subnet groups:** When we create a caching cluster inside an Amazon VPC, it is essential to specify a cache subnet group. These are subnet groups (usually private subnets) where a caching cluster will be spanned.
- **ElastiCache for Memcached events:** When a major event occurs (such as a cache cluster being created, a node has been added, failure to add a node, or the modification of a security group), the ElastiCache cluster notifies a configured Amazon SNS topic.

Amazon ElastiCache for Redis

Let's have a look at some of the features and important components of ElastiCache for Redis, which are as follows:

- **ElastiCache nodes:** These are the smallest possible elements in the ElastiCache cluster. They are nothing but fixed-size chunks of secure network-attached RAM that run an instance of Redis. The number of nodes can easily be scaled up or down in the cluster, as they are all of same instance type, along with DNS name and port.
- **ElastiCache for Redis shards:** This is a grouping of related nodes. An Amazon ElastiCache Redis caching cluster can have a maximum of 90 shards. Each shard can have a minimum of one and a maximum of six nodes in a shard. In a multiple-node shard, one read/write primary node and between one and five replica nodes are present. Out of the box, when sharding is disabled, the caching cluster always has one shard.
- **ElastiCache for Redis clusters:** This is a logical grouping of one or more ElastiCache nodes. The caching data is partitioned automatically across the shards in a Redis cluster. As we have seen earlier, out of the box, the Amazon ElastiCache Redis cluster is a single shard. But, it can have a maximum of 90 shards, and each shard can have maximum of six nodes (that is, one read/write primary and up to five read-only). It is recommended to have more read-only nodes in a shard when the application is read-intensive, or to improve fault-tolerance. At any given time, the caching cluster can be scaled in or out.
- **ElastiCache for Redis replication:** For Redis replication, a shard must have two or more nodes. While one shard acts as a read/write primary, the remaining shards act as a read replica. ElastiCache for Redis allows a maximum of five read replicas. Replication can be implemented with the help of the API and CLI. It is called a **node group**. Each replica node asynchronously maintains a copy of the primary node's data. As a result, the application can read cached data from any node, but can only write to the primary node. The following diagram helps us to understand replication in a Redis cluster when the **cluster mode** is disabled and enabled:

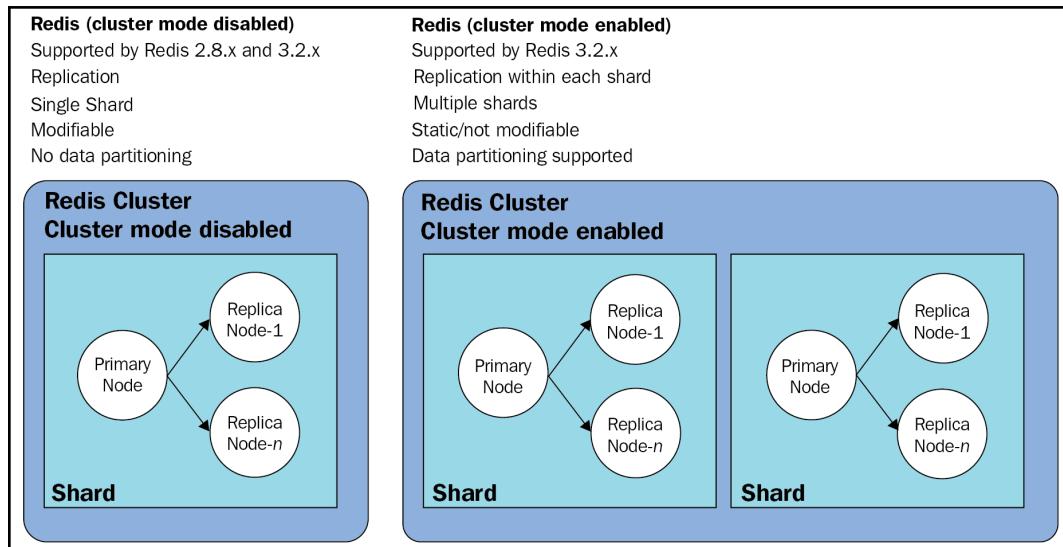


Figure 27.3: Redis cluster

Moving one step ahead, the table found at <https://docs.aws.amazon.com/AmazonElastiCache/latest/red-ug/WhatIs.Components.html> shows a crisp comparison between cluster modes that have been enabled and disabled for various cluster parameters.

As compared to open source Redis, Amazon ElastiCache for Redis has several cutting-edge enhanced reliability features, which make it more favorable to use in the cloud for a production load. For more details, please refer to <https://docs.aws.amazon.com/AmazonElastiCache/latest/red-ug/WhatIs.html>.

Some key points are as follows:



- If any primary has no replicas and the primary fails, you lose all that primary's data.
- You can use backup and restore to migrate to Redis (cluster mode enabled).
- You can use backup and restore to resize your Redis (cluster mode enabled) cluster.

It is important to understand that all the shards for a caching Redis cluster must be spanned within a single AWS region, but they can also be in multi-AZ:

- **ElastiCache for Redis endpoint:** This a unique address (URL-CNAME) for the caching cluster, based on the cluster configuration purpose of the endpoint. Let's see each of the Redis cluster configuration possibilities:
 - **Single-node Redis (cluster mode disabled) endpoint:** This endpoint can be used for both read/write purposes.
 - **Multi-node Redis (cluster mode disabled) endpoint:** This has two types of endpoints; one is the **primary endpoint** and the other is the **read endpoint**. The primary endpoint is always used to connect with the primary node in the cluster, in order to perform write operations to the caching cluster, even if the specific node in the primary role changes. For each replica node, there is an individual endpoint, which is called a read endpoint. These endpoints are used to read data from the caching cluster. Whether the caching cluster has scaled down or up manually in the application, these endpoints have to be updated in order to read data from the caching cluster.
- **Redis (cluster mode enabled) endpoint:** In this cluster configuration, the cluster has only one configuration endpoint. When you connect the configuration endpoint, each shard's primary and read endpoints in the cluster can be discovered by your application.
- **ElastiCache parameter groups:** The parameter groups in Redis are similar to those found in Memcached. Parameter groups provide engine-specific parameter settings, which are used to fine-tune the caching engine, such as controlled memory usage, eviction policies, item sizes, and more. It helps to make sure that all nodes in the caching cluster are configured in exactly the same way.
- **ElastiCache for Redis security:** By default, ElastiCache clusters are launched in an Amazon VPC. It is only possible to shape and separate network traffic to allow the whitelisted applications (EC2 instances) to access the caching cluster with the help of subnets.

- **ElastiCache security groups:** These are similar in both Redis and Memcached. Security groups are only applicable to the clusters that are not running in an Amazon VPC. They act as a firewall and help to control the caching cluster access only from whitelisted (desired) EC2 instances. By default, for security groups the `AuthorizeCacheSecurityGroupIngress` parameter is disabled. To enable a security group to access the caching cluster (that is, network), you need to enable this using the appropriate API or CLI.
- **ElastiCache subnet groups:** These are similar in both Redis and Memcached. When we create a caching cluster inside an Amazon VPC, it is essential to specify a cache subnet group; this is a subnet group (usually private subnets) where the caching cluster is spanned.
- **ElastiCache for Redis backups:** The Redis engine supports **snapshot**. This configuration takes a point-in-time copy of a Redis cluster. A new cluster can be seen or an existing cluster can be restored using these backups.
- **ElastiCache events:** These are similar in both Redis and Memcached. When major events occur (such as an icache cluster is created, a node has been added, failure to add a node, or the modification of a security group), the ElastiCache cluster sends a notification to a configured Amazon SNS topic.

Designing the right cache for your workload

In order to design a perfect cache, it is best practice to consider read/write speed, scaling, memory usage, and disk I/O. For a detailed comparison of these factors, please refer to the table at <https://aws.amazon.com/elasticache/redis-vs-memcached/>.

At the highest level, we can say that Memcached is generally used to store small and static data, such as HTML code pieces. As the Memcached core memory management is efficient and simple, it has a very small footprint of metadata; as a result, it consumes less memory for overheads. The disadvantage of Memcached is that it doesn't provide persistent storage options. If any node/cluster fails for any reason, all the data is lost. So, it is highly recommended to use Memcached with easily recoverable data.

Redis has five primary data structures, along with finely-grained control over data eviction. This makes it more suitable for large e-commerce applications, IoT applications, and real-time analytics. As compared to Memcached, the main advantage with Redis is that it provides a lightning fast in-memory caching performance, along with persistent storage with the help of read replica and primary replica.

We have seen that, at a high level, in-memory caching can help to minimize latency and improve throughput; but, for a first-time user, it could be a million-dollar puzzle as to how this mechanism is used in day-to-day programming or web applications. Choosing the right in-memory caching strategy is very important to minimize latency and improve throughput—it makes a big difference. Some of the industry-standard in-memory caching access patterns are as follows:

- **Cache miss:** This is a situation where the application requests some data (key) and it is not found in the in-memory cache. This may cause a delay in retrieving the information, as the requested information may need to be fetched from the actual data store, such as the database. With appropriate changes in the cache management approach, and the identification of the correct dataset to cache, such cache miss scenarios can be minimized.
- **Cache hit:** This is a situation where the application requests some data (key) and it is found in the in-memory cache. As it is found in the memory, the cache hit saves time for I/O, minimizes latency, and improves throughput.
- **Cache update:** This is a situation where the application requests some data (key) that may exist in the cache in a stalled state, or may not exist in the cache at all; therefore, it has to be written or updated in the cache.
- **Cache-aside (lazy loading):** This is one of the most widely used caching strategies. In this method, the application code can manually manage both the database and the cache information. The following diagram helps us to understand the scenario:

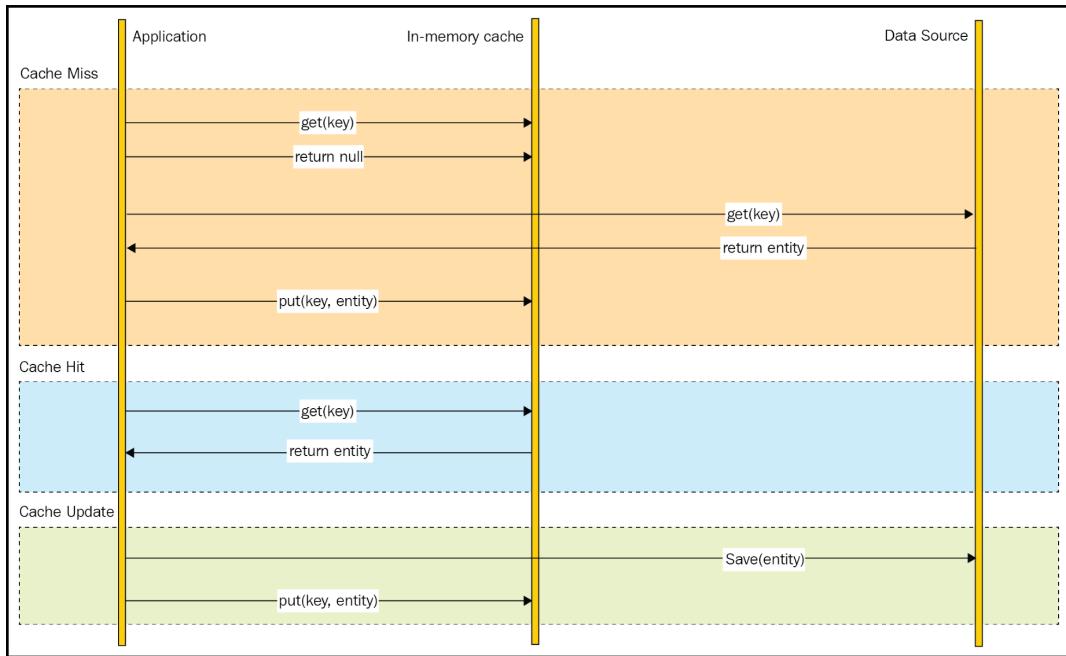


Figure 27.4: In-memory caching access patterns

Considering the preceding diagram, we can see the following:

1. The cache hit is blazing fast; as soon as the application sends a request for data, the valid data is found from the cache and is used directly.
2. The application sends a key retrieval request to the in-memory caching (Memcached or Redis), and if the data is not found, then it returns a NULL value. Immediately following this, the application sends a same-data retrieval request to the original data store. When these requested details are obtained from the original data source, it is updated in the in-memory caching, and is also used by the application for further usage, such as to display a web page on the end user's screen. This cache miss may add some latency, as it involves disk I/O.
3. The cache update occurs when the application has got new data from the main data store and it is updated in the in-memory caching.

The cache-aside caching strategy is also known as **Lazy Loading**. As the name suggests, the cache-aside caching strategy only loads data to the cache when it is required. In this way, only frequently used data remains in the cache by the implementation of the configured eviction policy. The cache-aside mechanism is suitable for read-heavy applications with Memcached and Redis in-memory caching. It is also robust in terms of in-memory cache cluster failures; in such situations, the applications still work by directly fetching data from the original data source. In the worst-case situation, when the database is not designed for such situations, the response time can be terrible, and it can sometimes even stop working.

- **Write-through cache:** This is a caching method whereby the data is written into both the cache and the relevant data store location at the same time. The following diagram helps us to understand the write-through cache:

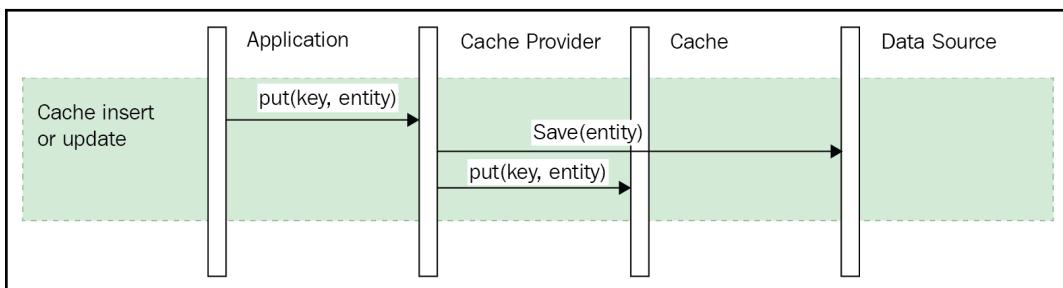


Figure 27.5: Write-through cache

As the data is stored in the cache, it is available for further retrieval and never gets stale. As new data comes in, it is first written into the cache. Another main advantage is that, in the worst cases, when cache clusters fail for any reason, data never gets lost, as it is also available in the main data store, such as the database. The application has to wait until the data has been inserted or updated at both of the data sources. As a result, it may incur a bit of latency while inserting or updating data using this strategy. This method is widely used where data loss tolerance is not acceptable. Along with this inserting or updating method, read-through is widely used to retrieve the data.

- **Write-around cache:** This is a caching method whereby the data is only written to the underlying data store without writing to the cache.
- **Write-back cache:** This is a caching method whereby the data is written to the cache and ElastiCache confirms the I/O completion. The data is subsequently stored to the underlined data store as a background process.

Summary

- Amazon ElastiCache is a scalable, reliable, and fully managed in-memory data store and caching service that improves the performance of your application, by serving the data from the in-memory cache instead of retrieving it from the source data store.
- Amazon ElastiCache offers two open source caching mechanisms, Redis and Memcached.
- As compared to open source Memcached, Amazon ElastiCache for Memcached has several cutting-edge and enhanced reliability features, which make it more favorable to use in the cloud for a production load.
- Amazon ElastiCache for Memcached can automatically detect node failure and recover accordingly.
- Amazon ElastiCache for Memcached supports automatic node discovery, as it saves developers from making any changes to the application when a node has been added or removed in the cluster.
- Amazon ElastiCache can provide high availability and avoid single point of failure by deploying nodes in a multi-AZ environment.
- Amazon ElastiCache for Memcached can be easily and swiftly integrated with other AWS services, such as Amazon EC2, Amazon CloudWatch, AWS CloudTrail, and Amazon SNS, in order to design a secure, high-performance, and managed in-memory caching solution.
- ElastiCache nodes are the smallest possible element in the ElastiCache cluster. They are nothing but a fixed-size chunk of secure network-attached RAM that run an instance of Memcached.
- There is a soft limit of 20 nodes in the Memcached cluster, or, in total, 100 nodes in the AWS region.
- One of the key features of the Memcached engine is its capability to Auto Discover nodes.
- The ElastiCache cluster is a logical grouping of one or more ElastiCache nodes.
- The ElastiCache endpoint is a unique address (URL-CNAME) for the caching cluster.
- ElastiCache parameter groups provide engine-specific parameter settings to fine-tune the caching engine, such as controlled memory usage, eviction policies, and item sizes.
- You can secure the ElastiCache cluster by restricting traffic to only whitelisted applications.

- When major events occur (such as a cache cluster being created, a node has been added, failure to add a node, or security group modification), the ElastiCache cluster sends notification to a configured Amazon SNS topic.
- ElastiCache for Redis shards is a group of related nodes.
- An Amazon ElastiCache Redis caching cluster can have a maximum of 90 shards.
- For Redis replication, a shard must have two or more nodes. While one shard acts as a read/write primary, the remaining shards act as read replicas.
- ElastiCache for Redis allows a maximum of five read replicas.
- If any primary has no replicas and the primary fails, you lose all that primary's data.
- You can use backup and restore to migrate to Redis (cluster mode enabled).
- You can use backup and restore to resize your Redis (cluster mode enabled) cluster.
- When you connect the configuration endpoint, each shard's primary and read endpoints in the cluster can be discovered by your application.
- To design a perfect caching, it is best practice to consider read/write speed, scaling, memory usage, and disk I/O.
- Cache miss is a situation where the application requests some data (key) and it is not found in the in-memory cache.
- Cache hit is a situation where the application requests some data (key) and it is found in the in-memory cache.
- Cache update is a situation where the application requests some data (key) that may exist in the cache in a stalled state, or it may not exist in the cache at all, and it has to be written or updated in the cache.
- The cache-aside caching strategy only loads data to the cache when it is required.
- The cache-aside caching strategy is also known as Lazy Loading.
- Write-through is a caching method whereby the data is written into the cache and the relevant data store location at the same time.
- Write-around is a caching method whereby the data is only written to the underline data store without writing to the cache.
- Write-back is a caching method whereby the data is written to the cache and ElastiCache confirms the I/O completion. The data is subsequently stored to the underlined data store as a background process.
- The Amazon ElastiCache in-memory caching cluster can be managed using a web console, CLI, SDK, or API.

27

Mock Tests

Mock test 1

- Packt Publishing has created the following IAM policy to control access on an S3 bucket. What does this policy do?

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:GetObject",  
                "s3:PutObject",  
                "s3:DeleteObject",  
                "s3>ListBucket"  
            ],  
            "Resource": "arn:aws:s3:::packtcontent/${aws:username}/*"  
        }  
    ]  
}
```

- A: Grants access to the packtcontent bucket to all the users.
- B: Grants access to a folder within the packtcontent bucket where the folder name dynamically matches with the username requesting the access.
- C: Grants access to the bucket named username.
- D: This policy is incorrect.

2. What is the significance of the IP address 169.254.169.254 in EC2 instances?
 - A: It can retrieve instance metadata, such as ami-id, local-ipv4, instance-id, hostname, and others, using the URL `http://169.254.169.254/latest/meta-data`.
 - B: It can retrieve instance user-data that is specified at the instance launch time, using the URL `http://169.254.169.254/latest/user-data`.
 - C: Both A and B.
 - D: None of the above.
3. You are assigned a task to build a real-time IoT device monitoring solution. Which of the following services would you use to gather and analyze the data coming from the devices?
 - A: CloudFront
 - B: Elastic Beanstalk
 - C: Kinesis
 - D: Data pipeline
4. Which of the following deployment type option(s) is/are provided by AWS CodeDeploy?
 - A: In-place deployment
 - B: Blue/green deployment
 - C: A/B testing
 - D: Rolling update
 - E: Both A and B
5. John is building a web application using Elastic Beanstalk. He needs to determine the right deployment policy for the application that ensures minimum impact in the event of failed deployment. Which Elastic Beanstalk deployment policy would you suggest to John?
 - A: All at once
 - B: Rolling
 - C: Rolling with additional batch
 - D: Immutable
 - E: None of the above

6. Packt Publishing uses a Java application in their Linux environment on AWS. Recently, when they released the new version of the application, the application started crashing out. The developers think that there is some memory issue and they want to monitor the memory utilization by the Java process. How can they monitor the memory utilization by the Java process?
- A: Create custom CloudWatch metrics for the Java process.
B: CloudWatch has in-built support for monitoring memory, so they can directly check the CloudWatch dashboard for the specific instance.
C: You cannot monitor memory utilization by a specific Java process in CloudWatch.
D: Create a shell script that monitors memory utilization by the Java process and sends the notification when a threshold is breached.
7. Where is your data encrypted when you use AWS KMS?
- A: You can use the KMS API and encrypt or decrypt the data using the CMK stored in KMS.
B: When you use KMS with AWS services, such as RDS and S3, they encrypt or decrypt the data using your KMS keys.
C: You can use the AWS Encryption SDK and use KMS keys to perform encryption or decryption in your application, whether on AWS or on an on-premises environment.
D: All of the above.
8. Packt Publishing has a CI/CD pipeline for their subscription portal in their on-premise environment. Currently, they use SVN for the code repository and Jenkins as the CI/CD engine. They are looking to migrate their workload to AWS-native services. What AWS services would you suggest for them on AWS?
- A: GitHub repository, Selenium on an EC2 instance for unit testing, and CodePipeline for automating the deployment process.
B: CodeCommit as repository, CodeBuild for compiling and packaging, and CodeDeploy and CodePipeline for unit testing and automating the deployment process.
C: The Bitbucket repository, Selenium for unit testing, and Jenkins on an EC2 instance for automating the deployment process.
D: The Bitbucket repository, CodePipeline for unit testing, and a Bamboo server on an EC2 instance for automating the deployment process.

9. What is the maximum size allowed for the Lambda Deployment Package?
- A: 25 MB zipped and 100 MB unzipped
 - B: 50 MB zipped and 250 MB unzipped
 - C: 100 MB zipped and 500 MB unzipped
 - D: 500 MB zipped and 1 GB unzipped
10. What is the default value for the maximum number of concurrent connections to an RDS instance?
- A: 45
 - B: 90
 - C: 1,000
 - D: Depends on the instance size
11. Which out of the following describes the features of step functions?
- A: Provides serverless orchestration for modern applications
 - B: Defines workflow as a state machine
 - C: Maintains the application state
 - D: Tracks the application workflow step
 - E: Stores an event log of the data passed between application components
 - F: All of the above
12. Which of the following recommendations should be considered while creating a partition key in a DynamoDB table?
- A: It is recommended to create a partition key with high-cardinality attributes with distinct values for each item, such as `email`, `employeeid`, `uuid`, `customerid`, `sessionid`, `orderid`, and so on.
 - B: It is recommended to create a partition key with composite attributes, such as `employeeid` and `departmentid` and create a sort key on the joining date.
 - C: It is recommended to add random numbers or digits from a predetermined range before or after a unique field for write-heavy use cases.
 - D: All of the above.

13. Packt Publishing has deployed a serverless application that provides access to over 7,000 practical ebooks and videos, with a subscription to the online library and learning platform for professional developers. The application uses S3 for the UI and storing static content, CloudFront distribution endpoints, DynamoDB, API Gateway, and Lambda functions to handle the workloads. How can Packt Publishing automate the deployment of Lambda functions and ensure that the Lambda function can be quickly rolled back if there is any issue in the deployment?
- A: Create a Lambda deployment package with the required libraries and deploy it using the Lambda API. Use Lambda versioning and point the latest version to an alias ARN with the name PROD. Perform automated tests and, in the event of any issue, use Lambda APIs to point the PROD alias ARN to previous version of the Lambda function and roll back.
- B: Use EC2 instances for writing business logic instead of using Lambda functions. Use Elastic Load Balancer and autoscaling with health checks. In the event of any issue, autoscaling will automatically terminate the instance with issue and replace it with a new instance.
- C: Develop a script using the AWS CLI to create and deploy a Lambda deployment package. In the event of any issue in the deployment, use another script to create the deployment package with a new function and replace the existing function code to rollback.
- D: Create a new Lambda function with every new deployment and modify the code to point to the latest Lambda function ARN during every deployment.
14. How do you enable unauthenticated identities (guest access) in Amazon Cognito?
- A: Cognito does not support unauthenticated identities.
- B: The Cognito console allows you to enable/disable unauthenticated identities access, in the Manage federated identities option for the required identity pool.
- C: Unauthenticated identities can be controlled from IAM.
- D: You can use STS for granting guest access to users.

15. What is the default visibility timeout for an Amazon SQS queue?

- A: 50 seconds
- B: 40 seconds
- C: 30 seconds
- D: 60 seconds

16. What API call is used to delete a message from an SQS queue?

- A: DeleteMessage
- B: EraseMessage
- C: TerminateMessage
- D: ClearMessage

17. You are assigned a task to create a serverless application for an organization. The application needs to address the travel requests of more than 500,000 employees. The application needs to authenticate the users and save the user data, such as user preferences, sign-in, session state, and more, and should be able to synchronize the user data across a user's multiple devices to provide consistent user experience. Which of the following AWS services is more suitable for such authentication requirements?

- A: IAM
- B: Active Directory Services
- C: Kinesis
- D: Cognito

18. What is true about Server Access Logging in S3?

- A: When Server Access Logging is enabled, all EC2 instance access logs are stored in the target bucket.
- B: When Server Access Logging is enabled, all EC2 instance application logs are stored in the target bucket.
- C: When Server Access Logging is enabled, S3 stores access logs of a source bucket to a target bucket configured by you.
- D: All of the above.
- E: None of the above.

19. Monthly billing for Lambda is based on which of the following?
- A: Number of uploaded functions in the AWS account per region.
 - B: Number of uploaded functions in the AWS account per region plus the execution time in minutes.
 - C: Execution request, and the time is rounded to the nearest 100 ms.
 - D: Execution request and time is rounded to the nearest 1 second.
20. Which of the following is a compulsory section in the CloudFormation template?
- A: Outputs and Resources
 - B: Resources
 - C: Parameters and Outputs
 - D: None of the above
21. If an EC2 instance with an instance store volume is stopped or terminated, any data on the instance store volume is lost:
- A: True
 - B: False
22. Which of the following statements is true about AWS regions and AZs?
- A: Every region is independent, consists of at least two or more AZs, and AZs within regions are interconnected through low-latency dedicated networks.
 - B: Each region is independent and has only one AZ.
 - C: You can create as many AZs as required from the AWS console.
 - D: Regions and AZs are only required when hosting legacy applications on the cloud.
23. Is it possible to stop an RDS instance?
- A: No, you cannot stop an RDS instance but you can terminate it.
 - B: Yes, you can stop an RDS instance for a maximum of 7 days if it's in a single AZ.
 - C: Yes, you can stop an RDS instance for a maximum of 7 days if it's in multi-AZ.
 - D: Yes, you can stop an RDS instance for a maximum of 7 days irrespective of single-AZ or multi-AZ, except for SQL Server multi-AZ RDS.

24. Which of the following statements are true? (Select two.)
- A: A group can contain many users, and a user can belong to multiple groups.
 - B: Groups can't be nested; they can contain only users, not other groups.
 - C: Both users and groups can be nested.
 - D: Groups can be nested but users cannot be nested.
25. What is the maximum size of an item in a DynamoDB table?
- A: 400 bytes
 - B: 400 KB
 - C: 400 MB
 - D: 400 GB
26. Which of the following statements is true?
- A: An NACL applies at the EC2 level and security groups apply at the network level.
 - B: Security groups apply at the EC2 level and NACL at the network level.
 - C: It can be implemented interchangeably depending on the project's requirements.
 - D: None of the above.
27. Which of the following AWS services supports infrastructure as a code?
- A: CloudFront
 - B: CloudFormation
 - C: CodeCommit
 - D: None of the above
28. When an RDS instance is configured in multi-AZ, what happens when the primary instance fails?
- A: The standby replica database automatically becomes the primary database.
 - B: You need to manually fail over control from the primary database to the secondary database in a different AZ.
 - C: The CNAME instance that is pointing to the primary database instance automatically changes to the standby database instance
 - D: All of the above.

29. An organization, Example Inc, runs their website on Amazon S3, which is named `https://www.example.com`. They have kept their corporate images in a separate S3 bucket, which is accessed at the endpoint `https://s3-us-east1.amazonaws.com/examplecorpimages`. While testing the website, Example Inc found that the images are blocked by the browser. In this scenario, what should the company do to resolve the issue so that the images are not blocked by the browser?
- A: Make the `examplecorpimages` bucket, where the images are stored publicly.
 - B: Enable versioning on the `examplecorpimages` bucket.
 - C: Create CORS configuration on the `examplecorpimages` bucket to allow cross-origin requests.
 - D: You can't do anything, as S3 does not allow you to host images in different buckets.
30. Which encryption method is supported by AWS EC2 and S3 by default?
- A: 256-bit Advanced Encryption Standard (AES-256)
 - B: RSA
 - C: 128-bit AES
 - D: DES
31. What is the purpose of the `cfn-init` helper script in CloudFormation?
- A: Installs and configures applications and packages on EC2.
 - B: Sends a signal to CloudFormation when the EC2 instance is successfully created.
 - C: Detects changes in resource metadata and runs user-specified actions when a change is detected.
 - D: All of the above.
32. A load balancer can span across ____.
- A: Multiple AZs
 - B: Multiple regions
 - C: Multiple AZs and, optionally, multiple regions
 - D: Depends on the region

33. AWS IAM is ____.
- A: Region-independent and free to use
 - B: Region-dependent and free to use
 - C: Region-dependent and the charges vary from region to region
 - D: None of the above
34. Which of the following CloudFormation template sections match a key to a corresponding value?
- A: Transform
 - B: Mappings
 - C: Metadata
 - D: Conditions
35. Which of the following statements is true?
- A: VPCs can span across multiple regions
 - B: VPCs can span across multiple AZs
 - C: VPCs can span across multiple AZs and multiple regions
 - D: VPCs span across AZs and, optionally, across regions
36. What happens when the password policy is changed or newly implemented?
- A: It is implemented immediately, but takes effect the next time an IAM user attempts to change the password.
 - B: It forces all the AWS IAM users whose passwords do not comply with the new password policy to change their passwords immediately.
 - C: It can be configured to apply only to a few IAM users.
 - D: None of the above.
37. Billing alerts are triggered by which AWS service?
- A: AWS Billing dashboard
 - B: CloudWatch
 - C: SES/SNS/SQS
 - D: All of the above

38. What is true about elasticity?

- A: Elasticity refers to the provisioning of new resources to match an increase in demand.
- B: Elasticity refers to automatically provisioning and deprovisioning resources to match the workload demand.
- C: Elasticity refers to the deprovisioning of resources due to a decrease in demand.
- D: None of the above.

39. How do you move/transfer an EC2 instance from one region to another?

- A: It is not possible to move an EC2 instance from one region to another
- B: It can be done only by a root user.
- C: Shut down the EC2, then take the AMI and copy it in another region to launch the new EC2 instance from the AMI.
- D: Raise a support request with AWS.
- E: None of the above.

40. Which of the following services are recommended for transferring petabytes of data between an on-premises data center and AWS?

- A: Snowball or Snowmobile
- B: S3 Transfer Accelerator
- C: S3 multipart upload
- D: S3 Import/Export
- E: Direct Connect

41. RRS stands for ____.

- A: Reduced Redundancy Storage
- B: Reduced Risk Storage
- C: Reduce Resource Storage
- D: None of the above

42. What can be used to provide internet connectivity to the resources residing in a private subnet?

- A: NAT instance or NAT gateway
- B: Internet gateway
- C: Virtual private gateway
- D: Elastic Load Balancer

43. Which of the following statements is true for CloudWatch metrics?
- A: They can be manually deleted when the CloudWatch alarm is no longer used.
 - B: They cannot be manually deleted.
 - C: They do not create any metrics for any custom CloudWatch alarms.
 - D: All of the above.
44. What subnet is usually recommended for hosting a database instance in RDS?
- A: Public subnet
 - B: Private subnet
 - C: A and B
 - D: None of the above
45. What is true about IAM policies?
- A: IAM policies cannot be modified.
 - B: When IAM policies are changed, it immediately reflects in the privileges of all users and groups.
 - C: IAM policies can be changed only by the AWS root account.
 - D: You need to raise a support request with AWS to change IAM policies.
46. Which of the following AWS services is suitable for data archiving?
- A: S3
 - B: EMR
 - C: Glacier
 - D. All of the above
47. What is true about deploying SSL on ELB?
- A: It is not possible.
 - B: It is not best practice to deploy SSL on ELB as it may increase the load on the EC2 instance.
 - C: It is suggested to deploy the SSL certificate on ELB to reduce the load on the EC2 instance.
 - D: Use of SSL is not required with ELB, as it automatically looks after encryption and decryption.

48. Which of the following is true about IP addressing in AWS?
- A: You can access an EC2 instance over the internet using a private IP address.
 - B: The public IP address of an EC2 instance does not change when you stop an instance and restart it.
 - C: Private IPs can be accessed only within the VPC.
 - D: None of the above.
49. Which of the following features is supported by DynamoDB?
- A: ELB
 - B: Autoscaling
 - C: A and B
 - D: None of the above
50. One EBS volume can be attached to which of the following?
- A: Only one EC2 instance.
 - B: Multiple EC2 instances.
 - C: Depends on the type of the EBS volume.
 - D: EBS cannot be attached to the EC2; it can only be attached to the RDS.
51. Data stored in an S3 bucket can be accessed from where?
- A: Within AWS
 - B: Within the same region
 - C: Within the same AZ
 - D: Anywhere across the internet
52. Which statement is true?
- A: Every region has at least two AZs and each AZ is isolated but interconnected with low-latency dedicated connectivity.
 - B: Every region may have one or more AZs and each AZ is isolated but interconnected with low-latency dedicated connectivity.
 - C: Every region has at least two AZs and each AZ is isolated but interconnected through the internet.
 - D: Every region consists of only one AZ that automatically provides highly available infrastructures.

53. Which of the following AWS services offers a NoSQL service?
- A: RDS
 - B: Simple database
 - C: EC2
 - D: EMR
 - E: DynamoDB
54. What can be done to reduce the cost of a mission-critical production application hosted on EC2 instances?
- A: Spot instances can be used.
 - B: A reserved instance can be used.
 - C: An on-demand instance can be used.
 - D: None of the above.
55. What is the largest individual object size supported by S3?
- A: 5 GB
 - B: 50 GB
 - C: 5 TB
 - D: Any size
56. Which AWS service helps to perform log analysis and resource monitoring?
- A: EC2
 - B: Lambda
 - C: CloudWatch
 - D: Any of the above
57. What is true about indexes in DynamoDB?
- A: A GSI can have a different partition key and sort key compared to its base table.
 - B: An LSI can have a different partition key and sort key compared to its base table.
 - C: A GSI should have the same partition key as its base table.
 - D: All of the above.

58. What happens when CloudFormation stack creation fails?
- A: It rolls back the stack and deletes any resources that have been created.
 - B: Skips that resource creation and continues.
 - C: Depends on the region in which the stack is being created.
 - D: None of the above.
59. What is session affinity in ELB?
- A: It ensures that ELB stops sending requests to instances that are deregistered or unhealthy, while existing connections are open.
 - B: It enables carrying source connection request information to the destination.
 - C: It enables ELB to bind a user's session to specific EC2 instances.
 - D: It distributes an incoming user's request evenly across registered AZs with ELB.
60. By default, which of the following metrics are not supported by CloudWatch?
- A: DiskRead/Write operations
 - B: NetworkIn/Out
 - C: CPU usage
 - D: Memory free/used
61. Which of the following statements is true for Lambda?
- A: Security groups and subnets can be assigned with Lambda functions.
 - B: Only subnets can be specified.
 - C: Only security groups can be specified.
 - D: Neither security groups nor subnets can be specified.
62. By default, a newly created object in S3 is what?
- A: Private and only accessible by an owner
 - B: Private and only accessible by IAM users
 - C: Public; anyone can access it
 - D: All of the above

63. Which of the following AWS services helps to use AWS storage as a local storage to the applications installed in the data centers?
- A: AWS storage gateway
 - B: Direct Connect
 - C: AWS Snowball
 - D: None of the above
64. Which of the following AWS services provides a relational database as a service?
- A: RDS
 - B: DynamoDB
 - C: RedShift
 - D: EMR
65. Which of the following queues in SQS can target messages that can't be processed (consumed) successfully?
- A: Fresh queue
 - B: Broken queue
 - C: Dead-letter queue
 - D: All of the above
66. What consistency model is used for scan operations in DynamoDB?
- A: Strongly consistent
 - B: Eventually consistent
 - C: Read after write consistency
 - D: All of the above
67. Which of the following AWS services can be used for various notification types?
- A: SES
 - B: SNS
 - C: SQS
 - D: All of the above

68. Which of the following platforms does not have in-built support for AWS Elastic Beanstalk?

- A: Java with Tomcat
- B: IIS
- C: Java SE
- D: Go
- E: C/C++
- F: Nginix

Mock test 2

1. An application hosted on an EC2 instance needs to write data into an S3 bucket, using the `PUT` method of the S3 API. What is the most secure way to authenticate `s3:PutObject` requests initiated by the application?

- A: Create an IAM user with the `s3:PutObject` permission and configure the access key and secret key of this user in the application to access the bucket for writing objects.
- B: Create an IAM role with the `s3:PutObject` permission and assign this role to the S3 bucket, where data needs to be stored.
- C: Create an IAM role with the `s3:PutObject` permission and assign this role to the EC2 instance where the application is hosted.
- D: Create a bucket policy that allows the `s3:PutObject` permission for everybody.

2. What is visibility timeout in an SQS queue?

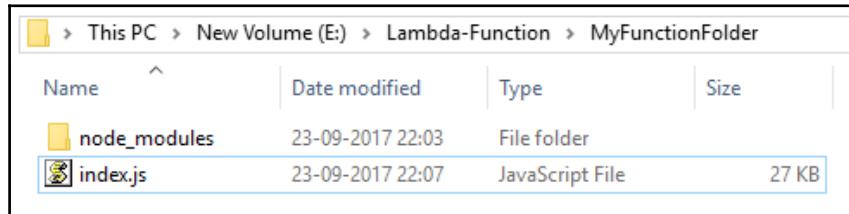
- A: When a message is sent to the queue by a producer, visibility timeout is triggered and the message is automatically deleted from the queue after the number of seconds configured in the visibility timeout.
- B: When a message is received by a subscriber from SQS, the subscriber processes the message and deletes it from the queue. If the message is not deleted within the visibility timeout, the messages reappears in the queue after the visibility timeout expires.
- C: Visibility timeout is the time in number of seconds that an entire queue remains available for processing. The entire SQS queue is automatically deleted after the visibility timeout expires.
- D: SQS sets the message size threshold in a queue. If a message producer application violates the message size threshold, the queue becomes inaccessible to the producer application for the number of seconds configured in the visibility timeout.

3. Which out of the following is a use case for S3 cross-region replication?

- A: You can comply with regulatory compliance requirements by replicating data in multiple regions.
- B: You can minimize access latency by replicating data across multiple regions if the users accessing the data are spread across regions.
- C: If the data stored in S3 is accessed by compute resources spread across multiple regions, you can increase operational efficiency by replicating the objects in multiple regions.
- D: It allows you to replicate data from one AWS account to another AWS account in the same or different region.
- E: All of the above

4. Packt Publishing wants to create a content portal. Their enterprise IT team wants to use S3 with JavaScript SDK for hosting the website and DynamoDB as the database engine. They are still contemplating the use of any AWS compute service where they can host business logic. Management is concerned about scaling, cost, and operational overhead. Which AWS compute service should they use that can reduce their operational cost and at the same time provide a scalable compute service to the portal?
- A: EC2 instances with autoscaling behind Elastic Load Balancer.
B: Elastic Beanstalk with autoscaling and Elastic Load Balancer.
C: Lambda.
D: Host the business logic in an on-premise server.
5. Packt Publishing is developing a reader collaboration application that stores session-state data in memory. During testing, it was discovered that the application is not able to scale. After contemplating the issue, it was decided to change the way the application handles session-state data. What approach would you suggest to Packt that can address the issue?
- A: Store session-state data in DynamoDB.
B: Create a separate session-state database server on an EC2 instance.
C: Store session-state data in a MySQL RDS instance.
D: Continue storing session-state data in memory and increase the memory size.
6. What is the efficient way of handling a database connection in Lambda?
- A: Define the database connection within the handler function and recreate the database connection with every function call.
B: Define the database connection in a global area before the handler function and re-use the connection.
C: You cannot create a database connection in Lambda.
D: Lambda automatically creates the RDS database connection for you; you do not need to explicitly create a database connection.
7. What type of keys does KMS include?
- A: AWS-Managed CMKs
B: Customer-Managed CMKs
C: AWS-Owned CMKs
D: All of the above

8. An online shopping portal uses a multi-AZ RDS database for storing their product and user data. The application is read-heavy and its read-to-write ratio for the RDS database is 7:3. Recently, the application started experiencing latency in reading and writing data on the RDS database. What would you suggest to address this issue?
- A: Use a Multi-AZ RDS instance to minimize the latency.
B: Fail over the database from primary AZ to secondary AZ.
C: Create an RDS read replica and use it for read operations.
D: Enable autoscaling on the RDS instance to automatically scale the number of instances based on traffic.
9. What operations should you perform to troubleshoot a Lambda function?
- A: Check the stack trace in the log and troubleshoot the error referred in the log.
B: Check for the permission denied error in the log and ensure that the Lambda execution role has sufficient permission to perform the operations.
C: Check for timeout errors. If a timeout error occurs, your function is probably not able to finish execution in time. Troubleshoot and adjust the timeout settings.
D: If you see a *memory exceeded* error, ensure that you have provisioned sufficient memory for the function.
E: All of the above.
10. Jennifer is building a serverless application using Lambda. She has created the following folder structure for her Lambda functions:



The screenshot shows a Windows File Explorer window with the following path: This PC > New Volume (E:) > Lambda-Function > MyFunctionFolder. Inside 'MyFunctionFolder', there are two items: 'node_modules' (File folder) and 'index.js' (JavaScript File, 27 KB). The table below represents this structure.

Name	Date modified	Type	Size
node_modules	23-09-2017 22:03	File folder	
index.js	23-09-2017 22:07	JavaScript File	27 KB

How can she package this Lambda function and deploy it on Lambda?

- A: Zip the entire `Lambda-Function` folder and deploy it on Lambda.
- B: Zip the content of the `Lambda-Function` folder and deploy it on Lambda.
- C: Zip the entire `MyFunctionFolde` folder and deploy it on Lambda.
- D: Zip the content of the `MyFunctionFolder` folder and deploy it on Lambda.

11. What does an immutable deployment policy do in Elastic Beanstalk?

- A: It launches a new version of the application in a separate environment alongside the old version. If the new version of the application does not pass the health checks, it terminates it without disturbing the existing version.
- B: It launches a new version of the application in the same instances. If the new version deployment fails, it retracts the new version automatically.
- C: It launches a new version of the application alongside the existing version and you can distribute the traffic between the old and new application version.
- D: It overwrites the existing application version with the new version. In the event of failed deployment, you need to manually redeploy the older version of the application.

12. What is true about AWS CodeDeploy?

- A: You can deploy any type of application using CodeDeploy.
- B: You can specify files to copy and script to run against each of the instances during deployment.
- C: AWS CodeDeploy is programming language-agnostic.
- D: You can use CodeDeploy for deployment on on-premises instances.
- E: All of the above.

13. Which of the following **Multi-Factor Authentication (MFA)** mechanism(s) is/are supported by Amazon Cognito, apart from the user ID and password?

- A: SMS text message
- B: Time-based **One-Time Password (OTP)**
- C: Both A and B
- D: None of the above

14. Packt Publishing has deployed a serverless application for managing their subscription service provided to readers. This application uses DynamoDB and Cognito, as well as other AWS serverless services. The DynamoDB table named `subscription` contains the Cognito user ID as the partition key. The developer has created the following IAM policy for DynamoDB access. Determine what can this policy do:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "dynamodb>DeleteItem",  
                "dynamodb>GetItem",  
                "dynamodb>PutItem",  
                "dynamodb>Query",  
                "dynamodb>UpdateItem"  
            ],  
            "Resource": [  
                "arn:aws:dynamodb:*.*:table/subscription"  
            ],  
            "Condition": {  
                "ForAllValues:StringEquals": {  
                    "dynamodb>LeadingKeys": [  
                        "${cognito-  
identity.amazonaws.com:sub}"  
                    ]  
                }  
            }  
        }  
    ]  
}
```

- A: The policy allows access to the subscription table, only from Cognito.
B: The policy allows row-level access to the subscription table based on the Cognito user ID.
C: The policy allows the DynamoDB service to manage Cognito users.
D: The policy allows access to subscription table for a specific Cognito user ID named `sub`.

15. What statement is true about API Gateway REST APIs?
- A: REST APIs are HTTP-based.
 - B: REST APIs follow the REST protocol, which enables stateless client-server communication.
 - C: REST APIs implement standard HTTP methods, such as GET, POST, PUT, PATCH and DELETE.
 - D: All of the above.
16. Which statement is true about API Gateway WebSocket APIs?
- A: WebSocket APIs follow the WebSocket protocol.
 - B: WebSocket APIs enable stateful communication between the client and server.
 - C: WebSocket APIs enable full-duplex communication between the client and server.
 - D: WebSocket APIs route messages based on message content.
 - E: All of the above.
17. What access-control mechanisms are supported by API Gateway?
- A: Resource policies wherein the access is restricted on the source IP and VPC endpoint.
 - B: IAM roles and policies for defining access permission on an entire API or specific method.
 - C: **Cross-Origin Resource Sharing (CORS)** for controlling cross-domain access.
 - D: Amazon Cognito user pools and OpenID identity brokers, such as Amazon, Google, and Facebook.
 - E: Usage plans, based on which you can track and limit the usage for each API key.
 - F: AWS WAF for protection against common web application exploits.
 - G: Lambda authorizers using bearer token authentication.
 - H: All of the above.
18. Which service(s) support server-side encryption using KMS in AWS?
- A: S3
 - B: RDS
 - C: EBS
 - D: Kinesis
 - E: All of the above

19. What does AWS X-Ray do?

- A: AWS X-Ray helps you to analyze and debug applications.
- B: Creates a service map of the services used by your application.
- C: Identifies bugs and errors in your application and automatically highlights them.
- D: Enables you to build your own analysis and visualization apps.
- E: All of the above.

20. What is true about the X-Ray daemon?

- A: The X-Ray daemon is an application that listens for traffic on the UDP port.
- B: The X-Ray daemon is an open source project.
- C: Lambda and Elastic Beanstalk can use the X-Ray daemon.
- D: You can run the X-Ray daemon on AWS as well as in an on-premises environment.
- E: All of the above.

21. What is the maximum execution time limit for a Lambda function?

- A: 1 minute
- B: 3 minutes
- C: 15 minutes
- D: Depends on the complexity of the function

22. Which of the following AWS services is used along with S3 to enable S3 Transfer Acceleration?

- A: EMR
- B: SES
- C: SQS
- D: CloudFront

23. Which of the following features is supported by DynamoDB?

- A: Supports relational databases
- B: Supports nested JSON
- C: Supports A and B
- D: None of the above

24. How many IPs are reserved in a VPC CIDR range?
- A: 10
 - B: 5
 - C: 2
 - D: 1
25. Which of the following programming languages is not currently supported by Lambda?
- A: Node.js
 - B: Java
 - C: Python
 - D: Scala
26. Which of the following is the efficient way to serve high-volume read-only application traffic in RDS?
- A: Create one or more read replicas to serve read-only application traffic.
 - B: Deploy the database in multi-AZ.
 - C: Create a memory-enhanced RDS instance type.
 - D: None of the above.
27. What is the primary purpose of using AWS SQS?
- A: Decoupling application components.
 - B: Building serverless architecture.
 - C: Building microservice architecture.
 - D: Building monolithic applications.
28. The AWS EC2 instance store volume provides which of the following:
- A: Permanent block storage.
 - B: Temporary block storage.
 - C: A and B.
 - D: None of the above.

29. Which of the following statement is true about DynamoDB?
- A: DynamoDB uses a pessimistic locking model.
 - B: DynamoDB uses conditional writes for consistency.
 - C: DynamoDB restricts item access during reads.
 - D: DynamoDB restricts item access during writes.
30. Which of the following AWS storage services supports object storage?
- A: EBS
 - B: EFS
 - C: S3
 - D: All of the above
31. Which of the following helper scripts cannot be used while bootstrapping an instance during stack creation in a CFT?
- A: cfn-get-metadata
 - B: cfn-put-metadata
 - C: cfn-init
 - D: cfn-hup
32. Which of the following terms attracts CEOs and CTOs to select cloud computing rather than on-premises infrastructure?
- A: They adopt cloud computing because everyone is doing so.
 - B: The cloud converts CapEx to OpEx and gives better ROI.
 - C: Cloud computing does not require hardware and software.
 - D: Cloud is more secure than an on-premises environment.
33. Which of the following statements about SWF is true?
- A: SWF tasks are assigned once and never duplicated.
 - B: SWF needs to be associated with an S3 bucket to start the workflow.
 - C: SWF uses SNS for sending mails and SES for sending notifications.
 - D: SWF requires at least one EC2 instance per domain.
34. Which of the following is the default consistency model in DynamoDB?
- A: Strongly consistent
 - B: Eventually consistent
 - C: Occasionally consistent
 - D: All of the above

35. Which of the following engines is not supported by RDS?
- A: Cassandra
 - B: MySQL
 - C: Oracle
 - D: MS SQL
36. What is the default interval for CloudWatch metrics?
- A: 10 minutes
 - B: 1 minute
 - C: 5 minutes
 - D: 2 minutes
37. Which of the following notification mechanisms is not supported by SNS?
- A: Pager
 - B: Email
 - C: SMS
 - D: All of the above
38. Which of the following is an example of a good DynamoDB hash key?
- A: Unique user ID, where the application has many users.
 - B: Status code, where there are only a few possible status code.
 - C: Item creation date, rounded to the nearest time period.
 - D: Employee name, where there are many employees with similar names.
39. By default, what privileges does a newly created AWS IAM user get?
- A: Inherits all root privileges.
 - B: Inherits administrator privileges.
 - C: Depends on the AWS region where the user is created.
 - D: Does not have any privileges.
40. What is the purpose of the `cfn-init` helper script in CloudFormation?
- A: Downloads and installs the packages and files described in the template.
 - B: Retrieves metadata attached to the template.
 - C: Initializes the bootstrapping command.
 - D: Signals that the stack is ready.

41. How many objects can be stored in each bucket?

- A: 500 objects/bucket
- B: 50,000 objects/bucket
- C: Virtually unlimited
- D: None of the above

42. What is true about EC2 instances?

- A: An EC2 instance can be part of only one target group.
- B: An EC2 instance can be part of multiple target groups.
- C: An EC2 instance cannot be a part of any target group.
- D: All of the above.

43. How do you obtain AWS security compliance documents?

- A: You can ask to initiate a third-party inspection at the AWS data center.
- B: Schedule a visit to the AWS data center and ask for the document.
- C: Obtain the compliance documents from the AWS compliance reporting service.
- D: Every compliance agency across the globe is automatically sent the AWS compliance report.

44. Packt Publishing stores periodic log data from its high-traffic book subscription portal to S3. Which object naming convention would give optimal performance on S3 for storing the log data?

- A: instanceID-log-HH-DD-MM-YYYY
- B: instanceID-log-YYYY-MM-DD-HH
- C: HH-DD-MM-YYYY-log_instanceID
- D: YYYY-MM-DD-HH-log_instanceID

45. Which of the following AWS services provides a schemaless database?

- A: RDS
- B: Simple database
- C: DynamoDB
- D: All of the above

46. What can be done to avoid a single point of failure in AWS?
- A: Create a multi-AZ architecture with ELB.
 - B: Create a multi-region architecture with ELB.
 - C: Create a multi-AZ with multi-region architecture with ELB and auto scaling.
 - D: AWS automatically takes care of single-point failure.
47. What is the maximum size of an SQS message?
- A: 1 MB
 - B: 512 KB
 - C: 128 KB
 - D: 256 KB
48. Each datapoint in CloudWatch needs to have a valid timestamp. This timestamp can be ____.
- A: 2 weeks in the past and up to 2 hours into the future.
 - B: 2 hours in the past and up to 2 weeks into the future.
 - C: 2 hours in the past and up to 2 hours into the future.
 - D: 2 weeks in the past and up to 2 weeks into the future
49. Which of the following statements is true for an internet-facing web application?
- A: Web server should be hosted in a private subnet and the associated database should be hosted in a public subnet.
 - B: Web server should be hosted in a public subnet and the associated database should be hosted in a private subnet.
 - C: Web server and database server should both be hosted in a public subnet.
 - D: Web server and database server should both be hosted in a private subnet.

50. What is the secure way to access an RDS database from an application deployed on EC2?
- A: Embed the access key and secret in the application source code.
 - B: Use an IAM role with the EC2 instance.
 - C: Directly specify the database user ID and password in the application configuration file.
 - D: Store the credential file on an encrypted object in S3 and access it from the application for the user ID and password.
51. Swathy Mohan, working for Packt Publishing, is repeatedly trying to launch an EC2 instance in the organization's AWS account. However, the instance gets terminated as soon as it is launched. What could be the possible reason behind this issue?
- A: The snapshot used for launching the instance is corrupt.
 - B: The AMI used for launching the instance is corrupt.
 - C: The AWS account has reached the maximum EC2 instance limit.
 - D: The user does not have permission to launch an EC2 instance.
52. Abhishek is unable to SSH an EC2 instance that is launched in Packt Publishing's AWS account. What could be the possible solution to resolve the accessibility issue?
- A: Modify the EC2 instance's security group to allow incoming TCP traffic on port 443.
 - B: Modify the EC2 instance's security group to allow incoming ICMP packets from your IP.
 - C: Modify the EC2 instance's security group to allow incoming TCP traffic on port 22.
 - D: Apply the most recently released operating system security patches on the EC2 instance.
53. Which AWS service triggers scale-in and scale-out in auto scaling?
- A: SES
 - B: SNS
 - C: SQS
 - D: CloudWatch
 - E: All of the above

54. How do you change the EC2 instance type?
- A: Directly change it, using the EC2 console, API, or CLI.
 - B: Stop the EC2 instance and change the instance type, using the EC2 console, API, or CLI.
 - C: AWS does not allow you to change the EC2 instance type once it is created.
 - D: The EC2 instance type can be changed only for a specific instance type family.
55. Which AWS service can be used to store archival data in a cost-effective way?
- A: AWS RRS
 - B: AWS S3
 - C: AWS RDS
 - D: AWS Glacier
56. Packt Publishing needs to process a big chunk of data stored in S3 using a data analytics process. This process needs to run periodically on an EC2 instance and does not need to have the EC2 instance running after the data is processed. The processed data should be stored back in S3. Which of the following instance types would be the most cost-effective for running these processes?
- A: On-demand EC2 instance.
 - B: Reserved EC2 instance with full upfront payment.
 - C: Reserved EC2 instance with no upfront payment.
 - D: Spot EC2 instance.
57. Which of the following AWS-managed services makes it easy to coordinate work across distributed application components?
- A: EMR
 - B: SWF
 - C: RedShift
 - D: SNS

58. Which of the following AWS services supports a volume that can be mounted on multiple EC2 instances at a time?
- A: EFS
 - B: EBS
 - C: Glacier
 - D: S3
59. Which of the following AWS services is not supported as an AWS Lambda event source?
- A: Amazon Alexa
 - B: AWS Config
 - C: Amazon Lex
 - D: EMR
60. What is true about Amazon SQS queues?
- A: Standard Queue guarantees at-least-once delivery and FIFO queue guarantees exactly-once delivery.
 - B: Standard Queue guarantees exactly-once delivery and FIFO queue guarantees at-least-once delivery.
 - C: Standard Queue and FIFO queues do not provide message delivery guarantees.
 - D: None of the above.
61. Which of the following multi-AZ data replications is implemented by RDS?
- A: Dynamic
 - B: Static
 - C: Asynchronous
 - D: Synchronous
62. What is the difference between the default VPC and custom VPC?
- A: A default VPC is only available in a few regions and AZs.
 - B: A default VPC is more secure than a custom VPC.
 - C: A default VPC is available in each region, while a custom VPC can be created as per requirement.
 - D: A custom VPC is more secure than a default VPC.

63. Which of the following AWS services supports the Content Delivery Network (CDN)?
- A: CloudFormation
 - B: CloudFront
 - C: VPC
 - D: All of the above
64. Which of the following storage options can create a persistent storage volume in EC2?
- A: Ephemeral
 - B: EBS
 - C: Both Ephemeral and EBS
 - D: None of the above
65. Which AWS service helps to connect a data center to AWS directly, bypassing the internet?
- A: Direct Connect
 - B: Storage gateway
 - C: VPC
 - D: Data pipeline
66. Which of the following services empowers ElasticBeanstalk? (Select three.)
- A: ELB
 - B: Auto Scaling Group
 - C: EC2
 - D: EFS
 - E: Cognito
67. What is the default limit for a number of S3 buckets that can be created in an AWS account?
- A: 100 per account
 - B: 200 per account
 - C: 100 per IAM User
 - D: Maximum 100 per region
 - E: There is no limit on the number of buckets in S3

68. A security group ____.

- A: Acts as a firewall at the EC2 level.
- B: Acts as a firewall at the subnet level.
- C: Acts as a firewall at the EC2 and the subnet level.
- D: None of the above.

69. ELB deletion protection helps to ____.

- A: Prevent the user's connection from terminating.
- B: Prevent an EC2 instance from being auto-terminated.
- C: Prevent accidental deletion of ELB.
- D: All of the above.

70. Which of the following statements is true?

- A: A user must have the required privileges to successfully create a stack from a template.
- B: Sufficient privileges should be with the user who has written the template.
- C: By default, every user gets a privilege to create any stack using any template.
- D: User privilege requirement depends upon the region where the CFT is executed.

Assessments

Mock Test 1

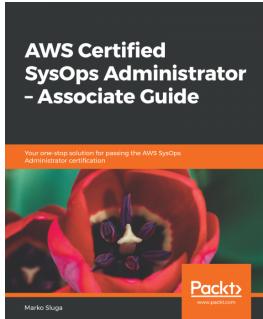
1. B	2. C	3. C	4. E	5. C	6. A	7. D	8. B	9. B
10. D	11. F	12. D	13. A	14. B	15. C	16. A	17. D	18. C
19. C	20. B	21. A	22. A	23. D	24. A	25. B	26. B	27. B
28. C	29. C	30. A	31. A	32. A	33. A	34. B	35. B	36. A
37. B	38. B	39. C	40. A	41. A	42. A	43. B	44. B	45. B
46. C	47. C	48. C	49. B	50. A	51. D	52. A	53. E	54. B
55. C	56. C	57. A	58. A	59. C	60. D	61. A	62. A	63. A
64. A	65. C	66. B	67. B	68. E				

Mock Test 2

1. B	2. B	3. E	4. C	5. A	6. B	7. D	8. C	9. E
10. D	11. A	12. E	13. C	14. B	15. D	16. E	17. H	18. E
19. E	20. E	21. C	22. D	23. B	24. B	25. D	26. A	27. A
28. B	29. B	30. C	31. B	32. B	33. A	34. B	35. A	36. C
37. A	38. A	39. D	40. A	41. C	42. B	43. C	44. C	45. C
46. A	47. D	48. A	49. B	50. B	51. C	52. C	53. D	54. B
55. D	56. D	57. B	58. A	59. D	60. A	61. D	62. C	63. B
64. B	65. A	66. A, B, and C	67. A	68. A	69. C	70. A		

Another Book You May Enjoy

If you enjoyed this book, you may be interested in the another book by Packt:



AWS Certified SysOps Administrator - Associate Guide

Marko Sluga

ISBN: 978-1-78899-077-6

- Create and manage users, groups, and permissions using AWS IAM services
- Create a secure VPC with public and private subnets, Network Access Control, and security groups
- Get started with launching your first EC2 instance, and working with it
- Handle application traffic with ELB and monitor AWS resources with CloudWatch
- Work with S3, Glacier, and CloudFront
- Work across distributed application components using SWF
- Understand event-based processing with Lambda and messaging SQS and SNS in AWS
- Get familiar with AWS deployment concepts and tools including Elastic Beanstalk, CloudFormation and AWS OpsWorks

Leave a review - let other readers know what you think

Please share your thoughts on this book with others by leaving a review on the site that you bought it from. If you purchased the book from Amazon, please leave us an honest review on this book's Amazon page. This is vital so that other potential readers can see and use your unbiased opinion to make purchasing decisions, we can understand what our customers think about our products, and our authors can see your feedback on the title that they have worked with Packt to create. It will only take a few minutes of your time, but is valuable to other potential customers, our authors, and Packt. Thank you!

Index

A

A record type 693
AAAA record type 693
Access Control List (ACL) 91, 262, 267
access request evaluation logic 473, 475
access
 controlling 142
Accumulo database 366
Active Directory (AD) 40, 82
Active Directory Federation Service (AD FS)
 about 96
 integrating, with AWS console 97, 98
actors, Amazon SWF
 about 489, 490
 activity worker 491
 decider 491
 workflow starter 490
administrators of key
 updating 593
Advanced Encryption Standard (AES) 195, 581
Aerospike database 364
AllegroGraph 365
Amazon Aurora DB cluster
 connecting to 357
Amazon Aurora DB
 about 326, 331
 features 333
Amazon CloudFront 311
Amazon CloudWatch 668
Amazon CodePipeline
 high-level view, of input artifacts at each stage 652
 high-level view, of output artifacts at each stage 652
 higher level view 651, 652

Amazon Cognito Sync 686
Amazon Cognito, application
 about 682
 AWS AppSync resources, accessing 685
 AWS services, accessing with identity pool 684
 AWS Services, accessing with identity pool 684
 AWS services, accessing with user pool 684
 resources, accessing with API Gateway 683
 server-side resources, accessing with user pool 682
 third party, authentication 684
 user pool, authentication 682
Amazon Cognito
 about 679, 680
 identity pool 681, 682
 user pool 680
Amazon EBS 309
Amazon EBS-backed AMI 174
Amazon EC2 instance store 309
Amazon EFS, modes
 General Purpose mode 312
 Max I/O mode 312
Amazon Elastic File System (EFS) 312, 313
Amazon ElastiCache
 engine types 705
 for Memcached 705
 for Redis 708
Amazon Glacier 308
Amazon Lambda 561
Amazon Machine Image (AMI)
 about 172, 173
 EC2 instance virtualization types 175
 placement groups 191
 root device types 174
Amazon Machine Images (AMIs) 161

Amazon RDS Aurora
 versus Amazon RDS MySQL 334

Amazon RDS components
 about 328
 availability zone (AZ) 329
 DB instances 328, 329
 DB option groups 330
 DB parameter groups 330
 region 329
 security groups 329

Amazon RDS DB instance
 connecting to 356

Amazon RDS instance
 monitoring 349, 350

Amazon RDS MySQL DB instance
 creating 341, 342, 343, 345, 346, 347, 348

Amazon RDS MySQL
 versus Amazon RDS Aurora 334

Amazon RDS, engine types
 Amazon Aurora DB 331, 333
 MariaDB 335
 Microsoft SQL Server 336, 337
 MySQL 337, 338
 Oracle 339, 340
 PostgreSQL 340

Amazon RDS
 about 355, 356
 engine types 330

Amazon Resource Name (ARN) 93, 410, 570

Amazon S3, terminologies
 bucket 252
 key 253
 region 253
 S3 data consistency model 254

Amazon S3
 about 252, 255, 307
 expiration actions 281
 Glacier 279
 IA storage 277
 Intelligent-Tiering 278
 lifecycle configuration 281
 lifecycle management 281
 lifecycle policy, defining for bucket 282, 285, 287
 One Zone-IA 278

RRS object 278
static website, hosting 288, 289
storage classes 276
storage classes, versus Glacier 280
transition actions 281

Amazon SNS fanout 452, 453

Amazon SNS messages
 sending, to Amazon SQS queues 477, 481

Amazon SNS topics
 access control, using 469
 access, managing 469
 creating 454, 457
 key concepts 470, 471

Amazon States Language (ASL) 675, 677

Amazon SWF
 SWF domains 492
 task lists 493
 tasks, pulling for 498
 using 487
 versus AWS Step Functions 674
 workflow 487, 488
 workflow execution closure 494, 495
 workflow execution life cycle 495, 497
 workflow history 488, 489

API Gateway
 about 670, 671
 API keys 672
 API, creating 671
 API, Monitoring 672
 API, versioning 671
 caching 671
 Handle REST APIs 671
 overview 670
 request authorization 672
 request throttling 671
 SDK, creating 671
 SDK, distributing 671
 WebSocket APIs 671
 working 672, 673

Apple Push Notification Service (APNS) 463

application alert
 about 453
 mobile device push notifications 453
 text messaging 453

Application Load Balancer (ALB)

about 212, 233
working 223, 224

applications, building with AWS Lambda
about 573
event source mapping, for AWS services 573, 574
event source mapping, for AWS stream-based services 575
event source mapping, for custom applications 576

AppSpec file
about 642
reference 642

architectural concepts, Elastic Beanstalk
about 537
web server environment tier 537, 538
worker environment tier 539

artifact bucket 652

aspects, DR plan
Recovery Point Objective (RPO) 51
Recovery Time Objective (RTO) 51

Aurora Replica 331

Auto Scaling groups
about 204
rolling updates 531

Automatic Speech Recognition (ASR) 43

availability zones (AZs) 22, 111, 163, 204, 278, 329, 367

AWS account alias 101, 102

AWS account ID 101

AWS account
creating 29, 31
deleting 33

AWS analytics services 41

AWS application integration services 44

AWS backup services 307, 309, 311

AWS business productivity services 45

AWS CLI help
obtaining 77

AWS CLI
about 75
configuring 76

DynamoDB Query, using with 400
installing 75, 76
reference 386

syntax 77
used, for creating IAM roles 90
used, for creating IAM user 77

AWS Cloud9 668

AWS cloud
versus on-premises data centers 19

AWS CloudFormation Designer
reference 506

AWS CodeBuild + third-party tools 666

AWS CodeBuild, build project
Artifacts configuration 635, 636
Buildspec configuration 633, 634
configuration 623, 624, 631, 632
environment 625, 627, 629, 630
Logs configuration 637, 638
source code 625

AWS CodeBuild
about 619, 620, 621, 666
working with 621, 622, 623

AWS CodeCommit 666

AWS CodeDeploy, components
about 642
application 643
blue/green, on Amazon ECS compute platform 646
blue/green, on AWS Lambda compute platform 646
compute platform 643
deployment configuration 642, 644
deployment group 642, 644
deployment type 645
IAM instance profile 646
revision 642, 646
service role 647
target version 647

AWS CodeDeploy
about 640, 641, 667
need for 640, 641

AWS CodePipeline
concepts 654
Continuous Delivery (CD), using 654
Continuous Integration (CI), using 654
usages 651
working with 655, 656, 658, 659

AWS CodeStar 667

AWS Command-Line Interface (CLI) 21
AWS compute services 36
AWS Console
 reference 677
AWS customer engagement services 46
AWS dashboard
 about 34
 components 34, 35
AWS Data Pipeline 313
AWS database services 37
AWS desktop and app streaming services 45
AWS developer tools 38
AWS Elastic File System (EFS) 308
AWS game development services 43
AWS IAM 60
AWS IoT services 43
AWS Lambda function, invocation types
 asynchronous 564
 synchronous 564
AWS Lambda function
 aliases, creating 570, 571
 versioning 570, 571
 writing 565
AWS Lambda
 about 371, 562
 all-at-once 648
 best practices 577, 578
 canary 647
 linear 647
 recapping 670
 used, for building application 573
AWS machine learning services 42
AWS Management Console
 about 21
 reference 266
 user access, controlling 102, 103
AWS management tools 39
AWS media services 46
AWS migration services 38
AWS mobile services 44
AWS networking and content delivery services
 37
AWS resource type
 reference 527
AWS root user 61
AWS security, identity, and compliance
 services 40
AWS services
 accessing 21
 architectural overview 472
 AWS Command-Line Interface (CLI) 21
 AWS Management Console 21
 AWS Software Development Kits (SDKs) 21
 KMS, using 595
 Query APIs 21
 reference 84, 530
 regular AWS services 573
 roles, creating 86, 89
 stream-based services 573
AWS Simple Notification Service (SNS)
 about 449, 450, 451
 best practices 484
 monitoring, with CloudWatch 482, 484
AWS Snowball 310, 322, 323
AWS Snowmobile 311, 323
AWS Software Development Kits (SDKs) 21
AWS Step Functions
 about 673
 state machine 675, 676
 state machine, creating 677, 678, 679
 tasks 677
 use cases 673
 versus Amazon SWF 674
 versus Lambda function 674
 working 675
AWS Storage Gateway 310, 314
AWS Storage Gateway, solutions
 file gateways 315
 tape-based storage solutions 319
 volume gateways 315
AWS storage services 36, 307, 309, 311
AWS tools
 for CI/CD 664, 665
AWS user access key
 obtaining 76
AWS X-Ray 667
AWS' free tier 31
AWS' global infrastructure
 about 22
 availability zones (AZs) 22, 24

regions 22, 24
AWS, using over traditional data center
 benefits 18, 19
AWS-managed CMKs 585
AWS
 about 18
 DR, using 50, 52
 overview 22
 services 666
 soft limits 49, 50
AWSALB 218

B

backup and restore DR model 52, 53
badge 624
bare-metal virtualization type 26
base table 371
billing alerts 241, 242
binary large object (BLOB) 605
binary translation 27
block storage 249
blue and green 645
Border Gateway Protocol (BGP) 130
Bring Your Own License (BYOL) model 167,
 331
bucket, policy
 about 262
 action 262
 effect 262
 principal 262
 resource 262
 statement ID 262
bucket
 about 252
 access control 261
 creating 256, 260
 limitation 261
 Requester Pays model 266
 restriction 261
 transfer acceleration 264
 user policies 263
Build badge 624
build process 640
Business Associate Agreement (BAA) 422

C

C#
 Lambda function handler 568, 569
canonical name (CNAME) 694
Cassandra database 366
certificate authorities (CAs) 693
CIDR to IPv4 conversion
 reference 122
Classic Load Balancer (CLB)
 about 205, 233
 creating 205, 207, 209, 211
 working 221, 222
ClassicLink 156, 157
Classless Inter-Domain Routing (CIDR) 112,
 113
cloud computing, models
 Infrastructure as a Service (IaaS) 24
 Platform as a Service (PaaS) 24
 Software as a Service (SaaS) 24
cloud computing
 evolution 16, 17
Cloud Security Alliance (CSA) 49
cloud services, type
 hybrid cloud 15
 private cloud 14
 public cloud 15
cloud services
 examples 14
CloudFormation condition
 reference 523
CloudFormation Templates (CFTs) 503, 530
CloudFormation
 best practices 531, 532
CloudFormer 530
CloudFront
 about 299, 301
 content, setting up 302
 regional edge cache 301
 use cases 303
CloudWatch alarm
 creating 234, 236, 238, 240, 241
CloudWatch dashboards 243
CloudWatch metrics 214
CloudWatch monitoring, types
 about 244, 245

basic monitoring 244
detailed monitoring 244

CloudWatch, elements
about 229
alarms 233, 234
dimensions 231
metrics 230
namespaces 229
percentile 233
statistics 232

CloudWatch
about 227
best practices 246, 247
used, for monitoring EBS volumes 195
working 228, 229

CNAME record type 694

CodePipeline 660
about 650
workflow 650

Cold HDD (sc1) 194

column-store database 365

Command-Line Interface (CLI)
about 386
used, for creating IAM group 81

Common Internet File System (CIFS) 250

components, ElastiCache for Memcached
about 706
ElastiCache for Memcached clusters 706
ElastiCache for Memcached endpoints 707
ElastiCache for Memcached events 707
ElastiCache nodes 706
ElastiCache parameter groups 707
ElastiCache security 707
ElastiCache security groups 707
ElastiCache subnet groups 707

components, ElastiCache for Redis
about 708
ElastiCache events 711
ElastiCache for Redis backups 711
ElastiCache for Redis clusters 708
ElastiCache for Redis endpoint 710
ElastiCache for Redis replication 708
ElastiCache for Redis security 710
ElastiCache for Redis shards 708
ElastiCache nodes 708

ElastiCache parameter groups 710
ElastiCache security groups 710
ElastiCache subnet groups 710
Redis (cluster mode enabled) endpoints 710

composite primary key 369

compute platform
Amazon ECS 644
AWS Lambda 643
EC2/On-premises 643

conditional writes 405

consumer APIs 601, 602, 603

content delivery network (CDN) 299, 311

Content Distribution Network (CDN) 37

Continuous Delivery (CD)
about 662
using, with AWS CodePipeline 654

Continuous Deployment (CD) 649, 663

Continuous Integration (CI)
about 620, 649, 661, 662
using, with AWS CodePipeline 654

Continuous Integration/Continuous Delivery (CI/CD)
about 660, 661
AWS tools 664, 665

Coordinated Universal Time (UTC) 230

core AWS services
about 35
AWS analytics services 41
AWS application integration services 44
AWS business productivity services 45
AWS compute services 36
AWS customer engagement services 46
AWS database services 37
AWS desktop and app streaming services 45
AWS developer tools 38
AWS game development services 43
AWS IoT services 43
AWS machine learning services 42
AWS management tools 39
AWS media services 46
AWS migration services 38
AWS mobile services 44
AWS networking and content delivery services 37
AWS security, identity, and compliance

services 40
AWS storage services 36
Criminal Justice Information Services (CJIS) 49
cron job 562
Cross-Origin Resource Sharing (CORS)
about 289, 290
configuring, on bucket 291
enabling, on bucket 292, 293
used, in scenarios 290
cross-region replication
about 293
enabling 294, 297
requisites 294
Customer Gateway (CGW) 132
Customer Master Key (CMK), types
about 586
AWS-managed CMKs 585
customer-managed CMKs 584
Customer Master Key (CMK)
about 195, 385, 428, 580, 584
creating 587, 589, 591
customer-managed CMKs 584
customer-provided key 195

D

data distribution 392, 393
Data Encryption Standard (DES) 581
data keys 584
data types
about 374, 375
document data type 377
scalar data type 376
set data type 378
DB instance identifiers 328
DB option groups 330
DB parameter groups 330
DB security groups 329
DB
restoring, from snapshot 352, 353
Dead Letter Queue (DLQ) 423
delay time 421
deployment type
blue/green deployment 645
blue/green on EC2 645
blue/green, on premises compute platform 645
in-place deployment 645
DevOps 660
DHCP option set 152
Digital Rights Management (DRM) 47
Direct-Attached Storage (DAS) 319
Disaster Recovery (DR)
about 307, 503
using, with AWS 50, 52
Distributed Denial of Service (DDoS) attack 40
DNS record types
A record type 693
AAAA record type 693
about 693
CAA record type 693
CNAME record type 694
MX record type 694
NAPTR record type 695
NS record type 695
PTR record type 696
SOA record type 696
SRV record type 697
TXT record type 697
document data type
List 377
Map 377
Document Object Model (DOM) 290
Domain Name Server (DNS) 152, 153, 154
Domain Name System (DNS) 687
Dynamic Delegation Discovery System (DDDS) 695
Dynamic Host Configuration Protocol (DHCP) 110, 115
dynamic reference to specify template values 512
DynamoDB API
permissions 411
DynamoDB provisioned throughput
about 389
read capacity units 389
table throughput, calculating 390
throughput calculation 390
write capacity units 389
DynamoDB Query
about 398, 399, 400
using, with AWS CLI 400

- DynamoDB Scan 401, 402
DynamoDB Streams 371, 372
DynamoDB table
 Auto Scaling 383, 384
 creating 378, 379, 380
 creating, advanced settings used 381
 encryption at rest 385
 item, reading from 402
 item, writing to 403
 Provisioned capacity 383
 read capacity modes 383
 secondary indexes, creating 382
 sort key, adding 380
 write capacity mode 383
DynamoDB, accessing methods
 CLI 386, 387, 388
 console 385, 386
 working, with API's 388
DynamoDB, components
 DynamoDB Streams 371, 372
 primary key 369, 370
 secondary indexes 370, 371
DynamoDB
 about 361, 364, 367
 access control 409, 410
 accessing, methods 385
 best practices 412
 components 367, 368
 policies, managing 410
 user authentication 409, 410
- E**
- EBS snapshot 196, 197
EBS volumes
 monitoring, with CloudWatch 195
EBS, types
 about 192
 Hard Disk Drive (HDD) 192
 previous generation volume 192
 Solid State Drive (SSD) 192
EBS-optimized EC2 instances 198
EC2 instance life cycle
 about 168
 instance launch 169
 instance reboot 170
 instance retirement 170
 instance start 169
 instance stop 169
 instance termination 170, 172
EC2 instance type
 changing 182, 183
EC2 instance virtualization types
 about 175
 Hardware Virtual Machine (HVM) 175
 Paravirtual (PV) 175
EC2 instance
 connecting to 183
 connecting, with PuTTY session 186, 188
 creating 176, 177, 179, 181
 metadata 189
 user data 189
EC2 pricing
 about 163
 dedicated hosts 167, 168
 on-demand 164
 reserved instances 165, 166
 scheduled reserved instances 166
 spot instances 164, 165
EC2/on-premises
 reference 646
EC2
 about 162, 163
 best practices 198, 199
Edge Location 299
Elastic Beanstalk application
 configuration, changing 551, 552
 creating 542, 544, 547
 environment information, viewing 548, 549
 load balancer changes, verifying 553
 version, deploying 549, 551
Elastic Beanstalk-supported platforms 540
Elastic Beanstalk
 about 535
 AWS account, signing into 541
 best practices 558
 cleaning up 554
 components 536
 new version 554
 web application, creating 541
 web applications, deploying 555

working with 541

Elastic Block Store (EBS) 49, 161, 162, 191

Elastic Compute Cloud (EC2) 17

Elastic Container Service (ECS) 677

Elastic File System (EFS) 250, 306

Elastic IP addresses 117, 118, 119, 163

Elastic Load Balancer (ELB), features

- about 212
- access logs 214
- back-end server encryption 218
- CloudWatch metrics 214
- configurable idle connection timeout 217
- connection draining 215
- cross-zone load balancing 217
- elastic IP address 219
- health checks 214
- high availability 214
- host-based routing 216
- IP addresses as targets 216
- load balancer deletion protection 216
- native HTTP/2 support 217
- path-based routing 216
- platforms 213
- preserve source IP Address 219
- protocols 213
- redirects and fixed response 220
- resource-based IAM permissions 220
- route to multiple ports on single instance 215

Server Name Indication (SNI) 218

slow start 220

SSL offloading 217

static IP 219

sticky sessions 218

tag-based IAM permissions 220

user authentication 220

WebSockets 215

Elastic Load Balancer (ELB), types

- about 205
- Application Load Balancer (ALB) 212
- Classic Load Balancer (CLB) 205
- Network Load Balancer 212

Elastic Load Balancer (ELB)

- about 51, 202, 203, 645, 697
- benefits 204
- best practices 225

working 220

Elastic MapReduce (EMR) 193

elastic network interface (ENI) 110, 143, 144, 216, 572

ElastiCache 703, 705

elasticity

- about 28
- versus scalability 28

elements, Identity and Access Management (IAM)

- about 62
- access key ID 64
- multi-factor authentication (MFA) 66
- password policy 65
- secret key 64
- security token-based MFA 67
- users 62, 64

encoding 580

encrypted EBS 194, 195

encryption 580

encryption at rest 385

encryption, types

- asymmetric encryption 581
- symmetric encryption 581

Enterprise Resource Planning (ERP) 18

environment variables 571, 572

eth0 116

eventual consistency 254

existing CMK

- modifying 592
- viewing 592

Extract, Transform, and Load (ETL) 42

F

failover routing policy 697

Family Educational Rights and Privacy Act (FERPA) 49

Fast-Moving consumer Goods (FMG) 395

Fiber Channel (FC) 250

file gateways 315

file storage 250

finite state machines 675

First In First Out (FIFO) queue 421, 422

flow logs 135, 140, 141

fully qualified domain name (FQDN) 695

Function as a Service (FaaS) 670

G

gateway-VTL, components
 virtual tape 320
 VTL 321
 VTS 321
General Purpose SSD (gp2) 192, 329, 508
geolocation routing policy 698
geoproximity routing policy 698
Glacier
 about 279
 versus storage classes 280
Global Secondary Index (GSI)
 about 370, 396
 versus Local Secondary Index (LSIs) 397
Global Transaction Identifiers (GTIDs) 338
Google Cloud Messaging (GCM) 463

H

Hard Disk Drive (HDD)
 about 192
 Cold HDD (sc1) 194
 Throughput optimized HDD (st1) 193
Hardware Security Module (HSM) 41
Hardware Virtual Machine (HVM) 27, 175
HBase database 366
health check 699, 700
Health Insurance Portability and Accountability
 Act (HIPAA) 49
HeidiSQL 357
high-resolution metrics 231
hosted hypervisor 26
hosted zones 692
hosted zones, types
 about 692
 private hosted zones 692
 public hosted zones 692
HTTP Live Streaming (HLS) 603
hybrid cloud 15
Hypertable database 366
hypervisor 26

I

IA storage
 about 277
 features 277
IAM groups
 about 78, 80
 creating 80
 creating, with CLI 81
 existing users, adding 81
IAM policy simulator 95
IAM roles
 about 81, 82
 creating, with AWS CLI 90
 reference 84
IAM user
 creating, with AWS CLI 77
IBM graph 365
Identity and Access Management (IAM)
 about 32, 60, 409, 486
 access, managing with 499
 best practices 103, 105
Identity Providers (IdP) 82, 680
identity-based policies 410
infinite state machines 675
Infrastructure as a Service (IaaS) 17, 24
Infrequently Accessed (IA) 276
inline policies 92
instance 162
instance store 162
Instance store-backed AMI 174
instance types 162
Integrated Development Environment (IDE) 39, 668
Intelligent-Tiering
 about 278
 characteristics 279
Internet Assigned Numbers Authority (IANA)
 140
Internet gateway (IGW) 110
 about 146
 egress-only IGWs 147
internet gateways
 egress-only IGWs 148
Internet of Things (IoT) 161
Internet Service Provider (ISP) 689

Internet Small Computer System Interface (iSCSI) 315
intrinsic function 517
invoke operation 564
IP addressing
 about 115
 Elastic IP addresses 118, 119
 Private IPs 115, 116
 Public IPs 116, 117
IT Vendor (ITV) 395
item, writing to DynamoDB table
 conditional writes 405, 406, 407, 408, 409
 DeleteItem 405
 PutItem 403, 404
 UpdateItem 404, 405

J

Java applications
 Kinesis Data Analytics for 614, 615, 616
Java Database Connectivity (JDBC) 37
Java
 Lambda function handler 566

K

Key Management Service (KMS)
 about 48, 195, 385, 428, 572, 580
 using, with AWS services 595
 working 582, 583

keys, types
 about 584
 Customer Master Key (CMK) 584
 data keys 584
keys
 disabling 594
 enabling 594
 tagging 594

Kinesis Client Library (KCL) 603

Kinesis Data Analytics
 about 612
 for Java applications 614, 615, 616
 for SQL applications 612, 613, 614

Kinesis Data Firehose
 about 608
 data flow 610, 611
 key concepts 609

Kinesis Data Streams
 about 603
 architecture 604
 terminology 605, 606, 607, 608
Kinesis Video Streams API
 about 600
 consumer APIs 601, 602, 603
 producer API 600
Kinesis Video Streams
 about 597, 598, 599, 600
 advantages 598

L

lab mode 333
Lambda function handler
 in C# 568, 569
 in Java 566, 567
 in Node.js 565, 566
 in Python 567
Lambda function
 about 561, 562, 564
 deploying 569, 570
 invoking, with SNS notification 475, 476
 versus AWS Step Functions 674

Lambda functions
 over VPC 572
 tagging 572
latency routing policy 698
Lazy Loading 714
license included 331
Linux EC2 instance
 connecting to, from Microsoft Windows system 184
Local Secondary Index (LSI)
 about 370, 396
 versus Global Secondary Index (GSIs) 397

M

Magic Quadrant (MQ) 18
magnetic storage type 329
managed policies, types
 about 92
 AWS-managed policies 92
 customer-managed policies 92
MariaDB 326, 330, 335

MariaDB instance
connecting to 357

media access control (MAC) 144

Memcached 705, 711, 712, 714

message queue 416

Microsoft SQL Server 326, 330, 336, 337

Motion Picture Association of America (MPAA)
49

Multi-Factor Authentication (MFA) 62, 680

multi-site DR model 57, 58

MultiLangDaemon 608

multivalue answer routing policy 698

MySQL 326, 330, 337, 338

MySQL instance
connecting to 358

N

Name Authority Pointer (NAPTR) 695

naming rules
about 374, 375
for DynamoDB 374

NAT instance
about 148
versus NAT gateway 151

Neo4j 365

network 110

Network Access Control List (NACL)
about 119, 135, 139
versus security groups 140

Network Address Translation (NAT) 110, 113,
148, 149, 150, 151, 573

Network Basic Input/Output System (NetBIOS)
152

Network File System (NFS) protocol 250, 314

Network File System Version 4.1 (NFSv4.1)
protocol 308

Network Load Balancer 212

Network Time Protocol (NTP) 153

node group 708

Node.js
Lambda function handler 565, 566

Non-Relational database 363

non-root user
versus root user 32

Non-SQL database 363

NoSQL database, types
document databases 364
graph databases 365
key-value pair databases 364
wide column databases 365, 366

NoSQL databases
about 363
types 363
using 366
versus SQL 366

Not-only-SQL database 363

NS record type 695

O

object identifiers, Amazon SWF
about 493
activity tasks 493
activity type 493
decision tasks 493
workflow type 493

object storage 250

object, elements
access control information 268
key 267
metadata 267
subresources 267
value 267
version ID 267

object, metadata type
system metadata 271
user-defined metadata 273

object
about 267, 302
key naming guide 268, 270
keys 268
metadata 271
tagging 275
version, enabling on bucket 274
versioning 273

on-demand invocation 564

on-premises data centers
versus AWS cloud 19

on-premises instances
reference 644

Open Database Connectivity (ODBC) 37

OpenID Connect (OIDC) 82, 680
OpenStack 17
operating system (OS) 17
Oracle 326, 330, 339, 340
Oracle instance
 connecting to 358
organizational unit (OU) 91
Origin Access Identity (OAI) 303

P

Para Virtualization Mode (PVM) 27
parallel execution
 reference 615
Parameters section, template structure
 AWS-specific parameters 512, 515, 517, 520
Paravirtual (PV) 175
partition key 393, 394, 395
partitions 392, 393
PEM file
 converting, to private key (PPK) 185, 186
permissions 91
persistent storage 309
pilot light DR model 53, 55
placement groups 191
Plain Old Java Object (POJO) class 567
Platform as a Service (PaaS) 17, 24
Point of Presence (PoP) 299
policy simulator
 reference 95
policy
 about 90
 Active Directory Federation Service (AD FS)
 96
 IAM policy simulator 95
 inline policies 92
 managed policies 92
 resource-based policies 93, 94, 95
 web identity federation 98, 100
PostgreSQL 326, 340
previous generation volume 192
primary instance 331
primary key
 about 369, 370
 types 370

private cloud 14
producer 599
producer API 600
protected health information (PHI) 422
Provisioned capacity 383
Provisioned IOPS SSD (io1) 193, 329
PTR record type 696
public cloud 15
Public IPs 116, 117
public subnets 114, 115
Public-Key Cryptography Standards (PKCS)
 581
push model 573
put-get-delete paradigm 419
Python 3
 installing, on Linux 75
Python
 Lambda function handler 567

Q

Query APIs 21
queue
 attributes 423
 CloudWatch metrics, for SQS 443
 creating 425, 427
 Dead Letter Queue (DLQ) 423
 deleting 436
 FIFO queue 422
 logging 443
 message, deleting 432
 message, sending 429
 message, viewing 432
 monitoring 443
 operations in 425
 purging 434
 SQS API actions, logging 445
 standard queue 422
 subscribing, to topic 437
 user permissions, adding 439
 working 420

R

range key 369
RDS instance type
 modifying 353, 354

Read capacity checkbox 383
read capacity unit 389
read consistency models
 about 373
 types 373
read endpoint 710
read-after-write consistency 254
Recovery Point Objective (RPO) 51
Recovery Time Objective (RTO) 51
Redis 705, 708, 711, 712, 714
Reduced Redundancy Storage (RRS) 276
region 329
Relational Database Management System (RDBMS) 331, 361, 362
Relational Database Service (RDS)
 about 227, 326, 327, 328
 best practices 359
Requester Pays
 about 266
 enabling, on bucket 266
 model 266
resource-based policies 93, 262, 410
return on investment (ROI)
 about 21
 versus total cost of ownership (TCO) 20
reverse DNS record 696
revision 656
RFC 4632 standards
 reference link 113
Riak database 364
Rivest Cipher 4 (RC4) 581
root account credentials 61
root device 174
root device types
 about 174
 Amazon EBS-backed AMI 174
 Instance store-backed AMI 174
root name server
 reference 690
root user
 versus non-root user 32
Route 53
 about 687
 working 689, 691
 working with 688
route table 144, 145, 146
routing policies
 about 697
 failover routing policy 697
 geolocation routing policy 698
 geoproximity routing policy 698
 latency routing policy 698
 multivalue answer routing policy 698
 simple routing policy 697
 weighted routing policy 698
RRS object
 about 278
 characteristics 278
RunInstances API 164

S

S3 console
 reference 256
S3 Standard storage 276
scalability
 about 28
 versus elasticity 28
scalar data type 376
scale out 28
secondary indexes, types
 about 370
 Global Secondary Index (GSI) 370
 Local Secondary Index (LSI) 370
secondary indexes
 about 370, 371
 advantages 396
Secure Sockets Layer (SSL) 40
security 135, 136
Security Assertion Markup Language (SAML)
 about 82, 680, 682
 reference 98
security groups
 about 135, 137, 138, 329
 versus Network Access Control List (NACL)
 140
Security Token Service (STS) 100
Server Message Block (SMB) 250
Server Name Indication (SNI) 213
Server Side Encryption (SSE) 317, 428, 446
Serverless Application Model (SAM) 526

Serverless computing 669
service control policy (SCP) 91
Service Level Agreement (SLA) 277, 605
Service Vendor (SRV) 395
Session Initiation Protocol (SIP) 695
set data type 378
shared security responsibility model
 about 47, 48
 reference 49
Simple Email Service (SES) 372
Simple Notification Service (SNS) 437
Simple Queue Service (SQS)
 about 416
 benefits 421
 features 421
 limitations 442
 use cases 418, 419
 using 417
simple routing policy 697
Simple Storage Service (S3) 17
Simple Workflow Service (SWF) 486
single sign-on (SSO) 96
snapshot
 creating 351, 352
 DB, restoring from 352, 353
SNS topic
 deleting 467, 469
 message, pushing to 460, 463
 subscribing to 458, 459
SOA record type 696
Software as a Service (SaaS) 17, 24, 218
Software Development Kit (SDK) 671
software virtualization (hypervisor)
 binary translation 27
 Hardware Virtual Machine (HVM) 27
 Para Virtualization Mode (PVM) 27
Solid State Drive (SSD)
 about 192, 367
 General Purpose SSD (gp2) 192
 Provisioned IOPS SSD (io1) 193
sort key 395
SPF record type 696
SQL applications
 Kinesis Data Analytics for 612, 613, 614
SQS security

about 446
authentication 446
SRV record type 697
SSH connection issues
 troubleshooting 189
stack 506, 508
standard queue 422
Standard storage
 features 277
standard-resolution metrics 231
Start of Authority (SOA) 696
state fields
 using 676
state machine
 about 675, 676
 finite state machines 675
 infinite state machines 675
statement ID 262
static website
 hosting, on Amazon S3 288, 289
Storage Area Network (SAN) 319
storage classes
 versus Glacier 280
Structured Query Language (SQL)
 about 361, 363
 versus, NoSQL databases 366
subnets
 about 109
 private subnets 113
 public subnets 114, 115
subnetwork 113
subresources, types
 Access Control List (ACL) 267
 torrent 268
sweepers 412
SWF actions
 allowing, IAM policy 499
SWF endpoints 498
system alerts
 about 453
 push email 453

T

Table of Contents (TOC) 197
table partitioning 392

- table throughput
 calculating 390
- table-style database 365
- tape-based storage solutions 319
- tape-based storage solutions, types
 Virtual Tape Library (VTL) 319
- tasks, Amazon SWF
 activity task 492
 decision task 492
 lambda task 492
- template structure
 about 508
- AWSTemplateFormatVersion section 510
- Conditions section 523, 524
- Description section 510
- Mappings section 521, 523
- Metadata section 511
- Outputs section 528, 529
- Parameters section 511
- Resources section 527
- Transform section 526
- templates 503, 504, 506
- temporary security credentials
 reference 101
- termination protection 171
- terminologies, IAM role
 AWS service role 83
- AWS service role, for an EC2 instance 83
- AWS service-linked role 84
- delegation 84
- federation 85
- permissions boundary 85
- policy 85
- principal 86
- role 83
- role chaining 84
- role for cross-account access 86
- Test Pyramid 619
- throughput calculation
 about 390
- example 390, 391, 392
- Throughput optimized HDD (st1) 193
- Time To Live (TTL) 385
- timeout 563
- Top Level Domain (TLD) 691
- torrent 267
- total cost of ownership (TCO)
 about 21
- versus return on investment (ROI) 20
- transactions per second (TPS) 422
- Transfer Acceleration
 about 264
- enabling 265
- transition 658
- Transport Layer Security (TLS) 40, 598
- TXT record type 697

U

- user policies 262, 263
- users of key
 updating 593
- users, Identity and Access Management (IAM)
 66

V

- vaults 308
- virtual machines (VMs) 17
- virtual MFA device
 enabling, for user 67, 70, 71
- Virtual Private Clouds (VPCs)
 about 109, 110, 111, 112, 113, 163, 204, 355, 705
- best practices 157, 158
- ClassicLink 156, 157
- creating 119
- endpoints 155, 156
- hardware VPN access 129, 130, 132
- IP addressing 115
- Lambda functions 572
- subnets 113
- with hardware VPN access 132, 135
- with private subnet 132, 135
- with private subnets 126, 127, 128
- with public and private subnets 129, 130, 132
- with public subnets 126, 127, 128
- with single public subnet 120, 121, 122, 125
- Virtual Private Gateway (VGW) 132
- Virtual Private Network (VPN) 17, 204
- Virtual Private Servers (VPS) 36

Virtual Tape Library (VTL) 319
Virtual Tape Shelf (VTS) 319
Virtual Tape Shelf (VTS), storage
 cache storage 321
 upload buffer 322
virtualization 25
virtualization production types
 class 1 type 26
 class 2 type 26
virtualization types, based on virtualization methods
 about 26
 hardware emulation 27
 OS-level virtualization 26
 software virtualization (hypervisor) 27
virtualization types
 based on virtualization software 26
volume gateways, types
 cached volumes 316, 317
 gateway-stored volumes 317, 319
volume gateways
 about 315
 types 315
VPC networking components
 about 143
DHCP option set 152
Domain Name Server (DNS) 153, 154
elastic network interface (ENI) 143, 144
Internet gateway (IGW) 146, 147
Network Address Translation (NAT) 148
 route table 144, 145, 146
VPC peering 154, 155
VPC security groups 329

W

warm standby DR model 56
web applications, Elastic Beanstalk
 monitoring 557
web applications
 about 541
 deploying, to Elastic Beanstalk 555
web server environment tier 537, 538
Web Services Description Language (WSDL)
 510
WebSocket handshaking 215
weighted routing policy 698
worker environment tier 539
Write capacity checkbox 383
write capacity units 389
WS IAM user
 creating, with AWS dashboard 72, 74
WS user secret key
 obtaining 76