PEARSON IT CERTIFICATION

Practice Tests

Flash Cards

Review Exercises

Study Planner

Cert Guide

Advance your IT career with hands-on learning

AWS Certified Solutions Architect – Associate

(SAA-C02)

MARK WILKINS

AWS Certified Solutions Architect - Associate (SAA-C02) Cert Guide

Mark Wilkins

Pearson

AWS Certified Solutions Architect - Associate (SAA-C02) Cert Guide

ISBN-13: 978-0-13-732521-4

ISBN-10: 0-13-732521-5

Library of Congress Control Number: 2021937055

02 22

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Pearson IT Certification cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an “as is” basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

Special Sales

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact intlcs@pearson.com.

Editor-in-Chief
Mark Taub

Product Line Manager
Brett Bartow

Executive Editor
Nancy Davis

Development Editor
Christopher Cleveland

Managing Editor
Sandra Schroeder

Senior Project Editor
Tonya Simpson

Copy Editor
Kitty Wilson

Indexer
Timothy Wright

Proofreader
Betty Pessagno

Technical Editor
Akhil Behl

Publishing Coordinator
Cindy Teeters

Cover Designer
Chuti Prasertsith

Compositor
codeMantra

Contents at a Glance

Introduction xxvi

CHAPTER 1 Understanding the Foundations of AWS 3

CHAPTER 2 Designing Multi-tier Architecture Solutions 29

CHAPTER 3 Highly Available and Fault-Tolerant Architectures 85

CHAPTER 4 Decoupling Mechanisms Using AWS Services 135

CHAPTER 5 Choosing Resilient Storage 169

CHAPTER 6 Designing High-Performing Compute Architecture 213

CHAPTER 7 Choosing High-Performing and Scalable Storage Solutions 301

CHAPTER 8 Selecting Networking Solutions for Workloads 329

CHAPTER 9 Designing High-Performing Database Solutions 413

CHAPTER 10 Securing AWS Resources 453

CHAPTER 11 Securing Application Tiers 517

CHAPTER 12 Selecting Appropriate Data Security Options 531

CHAPTER 13 Designing Cost-Effective Solutions 551

CHAPTER 14 Final Preparation 603

Glossary of Key Terms 615

APPENDIX A Answers to the “Do I Know This Already?” Quizzes and Q&A Sections 623

APPENDIX B *AWS Certified Solutions Architect - Associate (SAA-C02) Cert Guide* Exam Updates 639

Index 641

Online Elements:

APPENDIX C Study Planner

Glossary of Key Terms

Table of Contents

Introduction xxvi

Chapter 1 Understanding the Foundations of AWS 3

Essential Characteristics of AWS Cloud Computing 4

On-Demand Self-Service 5

Broad Network Access 6

Resource Pooling 6

Rapid Elasticity 7

Measured Service 7

Moving to AWS 8

Infrastructure as a Service (IaaS) 10

Platform as a Service (PaaS) 12

Operational Benefits of AWS 15

Cloud Provider Limitations 17

Security at AWS 18

Network Security at AWS 19

Application Security at AWS 19

Playing in the AWS Sandbox 20

Determining What Problem Needs to Be Solved 21

Migrating Applications 22

Applications That Can Be Moved to AWS and Hosted on an EC2 Instance with No Changes 23

Applications with Many Local Dependencies That Cause Problems When Being Moved to the Cloud 23

Replacing an Existing Application with a SaaS Application Hosted by a Public Cloud Provider 24

Applications That Should Remain on Premises and Eventually Be Deprecated 24

The AWS Well-Architected Framework 24

The Well-Architected Tool 26

In Conclusion 27

Chapter 2 Designing Multi-Tier Architecture Solutions 29

“Do I Know This Already?” Quiz 30

Foundation Topics 33

Availability and Reliability 33
Availability in the Cloud 34
Reliability 35
AWS Regions and Availability Zones 36
Availability Zones 39
Availability Zone Distribution 40
Multiple Availability Zones 42
Choosing a Region 43
Compliance Rules 44
Understanding Compliance Rules at AWS 45
The Shared Responsibility Model 48
AWS and Compliance 48
HIPAA 50
NIST 51
GovCloud 52
Latency Concerns 53
Services Offered in Each AWS Region 54
Calculating Costs 55
Multi-Tier Architecture Solutions 56
Resilient VPC Architecture 56
Design Problems to Overcome 58
Multi-Tier Data Redundancy 61
Protecting Against Application Failure 62
AWS Global Services: Edge Locations 63
Services Located at the Edge 64
Route 53 64
AWS Shield (Standard and Advanced) 71
Web Application Firewall (WAF) 72
CloudFront 73
AWS Lambda@Edge 78
AWS Service Quotas 80
Exam Preparation Tasks 81
Review All Key Topics 81
Define Key Terms 82
Q&A 82

Chapter 3 Highly Available and Fault-Tolerant Architectures 85

"Do I Know This Already?" Quiz 85

Foundation Topics 88

Comparing Architecture Designs 88

Designing for High Availability 88

Adding Fault Tolerance 90

Removing Single Points of Failure 91

Disaster Recovery and Business Continuity 94

Backup and Restoration 95

Pilot Light Solution 95

Warm Standby Solution 96

Hot Site Solution 99

Multi-Region Active-Active Application Deployment 100

The AWS Service-Level Agreement (SLA) 102

Automating AWS Architecture 103

Automating Infrastructure with CloudFormation 105

CloudFormation Components 107

CloudFormation Templates 107

CloudFormation Stacks 109

CloudFormation Stack Sets 113

Third-Party Solutions 114

AWS Service Catalog 115

Elastic Beanstalk 117

Updating Elastic Beanstalk Applications 119

Deployment Methodologies 121

Rule 1: Use One Codebase That Is Tracked with Version Control to Allow Many Deployments 123

AWS CodeCommit 124

Rule 2: Explicitly Declare and Isolate Dependencies 125

Rule 3: Store Configuration in the Environment 126

Rule 4: Treat Backing Services as Attached Resources 126

Rule 5: Separate the Build and Run Stages 127

Rule 6: Execute an App as One or More Stateless Processes 128

Rule 7: Export Services via Port Binding 129

Rule 8: Scale Out via the Process Model 129

Rule 9: Maximize Robustness with Fast Startup and Graceful Shutdown 130

Rule 10: Keep Development, Staging, and Production as Similar as Possible 130

Rule 11: Treat Logs as Event Streams 131

Rule 12: Run Admin/Management Tasks as One-Off Processes 131

Exam Preparation Tasks 131

Review All Key Topics 131

Define Key Terms 132

Q&A 132

Chapter 4 Decoupling Mechanisms Using AWS Services 135

"Do I Know This Already?" Quiz 135

Foundation Topics 138

Stateful Design 138

Changing User State Locations 140

User Session Management 142

Application Integration Services 144

Amazon SNS 144

SNS Cheat Sheet 147

Amazon SQS 147

SQS Cheat Sheet 149

AWS Step Functions 149

Lambda 152

Lambda Cheat Sheet 156

API Gateway 156

API Gateway Cheat Sheet 160

Building a Serverless Web App 160

Step 1: Create a Static Website 161

Step 2: Handle User Authentication 162

Step 3: Create the Serverless Backend Components 162

Step 4: Set Up the API Gateway 163

Step 5: Register for Conference 164

Exam Preparation Tasks 165

Review All Key Topics 165

Define Key Terms 166

Q&A 166

Chapter 5 Choosing Resilient Storage 169

Do I Know This Already? 170

Foundation Topics 174

AWS Storage Options 174

- Workload Storage Requirements 176

Elastic Block Storage (EBS) 177

- EBS Volume Types 178
- General-Purpose SSD (gp2/gp3) 180
- Elastic Volumes 182
- Attaching an EBS Volume 182
- EBS Cheat Sheet 183
- EBS Snapshots 184
- *Taking a Snapshot from a Linux Instance 184*
- *Taking a Snapshot from a Windows Instance 185*
- *Snapshot Administration 185*
- *Snapshot Cheat Sheet 186*
- Local Instance Storage 186

Elastic File System (EFS) 187

- EFS Performance Modes 188
- EFS Throughput Modes 188
- EFS Security 189
- EFS Lifecycle Management 190
- EFS DataSync 191
- EFS Cheat Sheet 191

FSx for Windows File Server 191

Simple Storage Service (S3) 194

- Buckets, Objects, and Keys 196
- S3 Data Consistency 198
- S3 Storage Classes 198
- S3 Management 200
- Versioning 204
- Amazon S3 Access Points 205
- S3 Cheat Sheet 206

S3 Glacier Storage Options 207

S3 Glacier 207

Vaults and Archives 208

S3 Glacier Deep Archive 208

Glacier Cheat Sheet 209

Exam Preparation Tasks 209

Review All Key Topics 210

Define Key Terms 211

Q&A 211

Chapter 6 Designing High-Performing Compute Architecture 213

Do I Know This Already? 213

Foundation Topics 217

EC2 Instances 217

Instance Families 218

What Is a vCPU? 219

EC2 Instance Choices 220

Micro Instances 221

General-Purpose Instances 221

Instances Designed to Burst 221

Compute-Optimized Instances 223

Memory-Optimized Instances 223

x1 Instances 223

High-Memory Instances 223

z1d Instances 224

Accelerated Computing Instances 224

g3 Instances 224

f1 Instances 224

Storage-Optimized Instances 224

Bare-Metal Instances 224

Dedicated Hosts 225

Dedicated Hosts Cheat Sheet 226

Dedicated Instances 226

EC2 Network Performance 226

Using Launch Templates 228

Changing the Current Instance Type 229

Amazon Lightsail 232

Amazon Machine Images (AMIs) 233

Choosing an AMI 235

AWS Linux AMIs 235

Windows AMIs 236

AMIs at AWS Marketplace 237

Creating a Custom AMI 237

Instance Store–Backed Windows or Linux AMIs 238

Custom Instance Store AMIs 239

AMI Build Considerations 240

Elastic Container Service (ECS) 242

Amazon Elastic Container Service (ECS) 244

AWS Fargate 245

AWS ECS for Kubernetes (EKS) 246

Monitoring with CloudWatch 247

CloudWatch Basic Monitoring 249

CloudWatch Logs 250

Collecting Data with the CloudWatch Agent 253

Planning for Monitoring 253

CloudWatch Integration 255

CloudWatch Terminology 257

Creating a CloudWatch Alarm 262

Additional Alarm and Action Settings 263

CloudWatch Cheat Sheet 264

Elastic Load Balancing (ELB) Services 264

Redundancy in Design 267

Health Checks 268

ELB Cheat Sheet 269

Classic Load Balancer (CLB) 269

Application Load Balancer (ALB) 270

Target Groups 271

ALB Cheat Sheet 271

Big-Picture Steps: ALB Creation 272

Rule Choices 274

HTTPS Listener Security Settings 276
Target Group Routing 277
Target Group Attributes 277
Maintaining User Sessions 278
Sticky Session Support 278
Configuring Health Checks 279
ALB Security 280
Monitoring Load Balancer Operation 283
CloudWatch 283
Access Logs 284
Network Load Balancer 284
NLB Cheat Sheet 285
EC2 Auto Scaling 285
EC2 Auto Scaling Cheat Sheet 289
EC2 Auto Scaling Operation 289
Launch Configuration 290
Launch Templates 290
Auto Scaling Groups (ASGs) 291
Scaling Options for Auto Scaling Groups 292
Cooldown Period 296
Termination Policy 296
Lifecycle Hooks 297
Exam Preparation Tasks 298
Review All Key Topics 298
Define Key Terms 299
Q&A 299
Chapter 7 Choosing High-Performing and Scalable Storage Solutions 301
Do I Know This Already? 301
Foundation Topics 304
Elastic Block Storage (Provisioned IOPS) 304
io1 and io2 304
Storage-Optimized EC2 Instances 305
Shared File Storage at AWS 306
Elastic File System (EFS) 307
EFS Performance Modes 309

Storage Classes 310
EFS Throughput Modes 311
EFS Security 312
EFS Cheat Sheet 312
Storage Options Compared 313
Amazon FSx for Windows File Server 315
FSx File System Performance 316
FSx Cheat Sheet 317
FSx for Lustre 317
FSx for Lustre Cheat Sheet 318
AWS Data Transfer Options 319
AWS Storage Gateway Family 322
AWS Storage Gateway Cheat Sheet 324
Exam Preparation Tasks 324
Review All Key Topics 324
Define Key Terms 325
Q&A 325
Chapter 8 Selecting Networking Solutions for Workloads 329
Do I Know This Already? 329
Foundation Topics 334
VPC Networking 334
Partnering with AWS 335
VPC Cheat Sheet 337
To Host or to Associate? 338
What's Behind the Networking Curtain? 339
It's All About Packet Flow 341
The Mapping Service 343
Creating a VPC 344
Using the Create VPC Wizard 344
Using the Launch VPC Wizard 345
Using the AWS CLI to Create a VPC 347
How Many VPCs Do You Need? 347
Creating the VPC CIDR Block 349
Planning Your Primary VPC CIDR Block 349

Adding a Secondary CIDR Block 351

The Default VPC 352

Revisiting Availability Zones 353

Subnets 354

Subnet Cheat Sheet 355

Route Tables 356

The Main Route Table 357

Custom Route Tables 357

Route Table Summary 360

Route Table Cheat Sheet 360

IP Address Types 361

Private IPv4 Addresses 361

Private IP Address Summary 362

Public IPv4 Addresses 362

Elastic IP Addresses 364

Public IPv4 Address Cheat Sheet 366

Inbound and Outbound Traffic Charges 367

Bring-Your-Own IP (BYOIP) 368

The BYOIP Process 369

IPv6 Addresses 370

Security Groups 371

Security Group Cheat Sheet 374

Custom Security Groups 374

Application Server Inbound Ports 375

Database Server Inbound Ports 376

Administration Access 376

Pinging an EC2 Instance 377

Elastic Load Balancing (ELB) 377

Security Group Planning 378

Network ACLs 379

Network ACL Implementation Details 380

Network ACL Cheat Sheet 380

Network ACL Rule Processing 381

Understanding Ephemeral Ports 383

Network ACL Planning 385
VPC Flow Logs 385
Connectivity Options 387
Peering VPCs 387
Establishing a Peering Connection 388
Endpoints 390
Gateway Endpoints 391
Interface Endpoints 392
Endpoint Services with PrivateLink 393
External Connections 395
Internet Gateway: The Public Door 396
Internet Gateway Cheat Sheet 397
Egress-Only Internet Gateway 398
NAT 399
NAT Gateway 399
NAT Gateway Cheat Sheet 400
Transit Gateway 401
VPN Connections 401
Virtual Private Gateway 404
Customer Gateway 404
VPN CloudHub 406
Understanding Route Propagation 406
Direct Connect 407
Direct Connect Cheat Sheet 408
Exam Preparation Tasks 409
Review All Key Topics 410
Define Key Terms 411
Q&A 411
Chapter 9 Designing High-Performing Database Solutions 413
Do I Know This Already? 413
Foundation Topics 417
Relational Database Service (RDS) 417
RDS Database Instances 418
Database Instance Class Types 420

High-Availability Design for RDS 420
Multi-AZ RDS Deployments 423
Big-Picture RDS Installation Steps 423
Monitoring Database Performance 425
Best Practices for RDS 425
RDS Cheat Sheet 426
Amazon Aurora 427
Aurora Storage 429
Communicating with Aurora 432
Aurora Cheat Sheet 433
DynamoDB 433
Database Design 101 435
DynamoDB Tables 437
Provisioning Table Capacity 438
Adaptive Capacity 439
Data Consistency 441
ACID and DynamoDB 442
Global Tables 443
DynamoDB Accelerator (DAX) 444
Backup and Restoration 445
ElastiCache 445
Memcached Cheat Sheet 447
Redis Cheat Sheet 447
Amazon Redshift 447
Exam Preparation Tasks 449
Review All Key Topics 449
Define Key Terms 450
Q&A 450
Chapter 10 Securing AWS Resources 453
Do I Know This Already? 454
Foundation Topics 457
Identity and Access Management (IAM) 457
IAM Policy Definitions 460
IAM Authentication 461
Requesting Access to AWS Resources 464

The Authorization Process 465

Actions 466

IAM Users and Groups 467

The Root User 468

The IAM User 470

Creating an IAM User 471

IAM User Access Keys 472

IAM Groups 474

Signing In as an IAM User 474

IAM Account Details 475

Creating a Password Policy 476

Rotating Access Keys 477

Using Multifactor Authentication (MFA) 479

Creating IAM Policies 479

IAM Policy Types 479

Identity-Based Policies 480

Resource-Based Policies 482

In-Line Policies 483

IAM Policy Creation 484

Policy Elements 484

Reading a Simple JSON Policy 486

Policy Actions 488

Additional Policy Control Options 489

Reviewing Policy Permissions 492

IAM Policy Versions 493

Using Conditional Elements 494

Using Tags with IAM Identities 495

IAM Roles 496

When to Use Roles 497

Using Roles When AWS Services Perform Actions on Your Behalf 497

Using Roles for EC2 Instances Hosting Applications That Need Access to AWS Resources 497

Using Roles with Mobile Applications 499

Cross-Account Access to AWS Resources 499

AWS Security Token Service (STS) 501

IAM Cheat Sheet 503

Identity Federation 503

IAM Best Practices 506

IAM Security Tools 507

AWS Organizations 509

AWS Organizations Cheat Sheet 510

AWS Resource Access Manager (RAM) 511

Exam Preparation Tasks 513

Review All Key Topics 513

Define Key Terms 514

Q&A 514

Chapter 11 Securing Application Tiers 517

Do I Know This Already? 518

Foundation Topics 520

AWS CloudTrail 520

Creating a CloudWatch Trail 521

CloudTrail Cheat Sheet 522

Essential AWS Management Tools 523

AWS Secrets Manager 523

GuardDuty 524

Amazon Inspector 525

AWS Trusted Advisor 526

Exam Preparation Tasks 528

Review All Key Topics 528

Define Key Terms 529

Q&A 529

Chapter 12 Selecting Appropriate Data Security Options 531

Do I Know This Already? 532

Foundation Topics 535

EBS Encryption 535

S3 Bucket Security 538

S3 Storage at Rest 540

Object Lock Policies 542

Legal Hold 542

S3 Glacier Storage at Rest 543

Key Management Service (KMS) 543

Envelope Encryption 544

KMS Cheat Sheet 545

CloudHSM 545

Amazon Certificate Manager (ACM) 546

Exam Preparation Tasks 547

Review All Key Topics 547

Define Key Terms 547

Q&A 548

Chapter 13 Designing Cost-Effective Solutions 551

Do I Know This Already? 551

Foundation Topics 555

Calculating AWS Costs 555

Management Service Costs 556

Understanding Tiered Pricing at AWS 557

Compute Costs 558

EC2 Pricing 559

On-Demand Instance Limits 560

Reserved Instances (RI) 562

Term Commitment 563

Payment Options 564

EC2 Reserved Instance Types 564

Scheduled Reserved EC2 Instances 565

Regional and Zonal Reserved Instances 565

Savings Plans 567

Spot Instances 568

Spot Fleet Optimization Strategies 571

Spot Capacity Pools 572

EC2 Fleet 573

EC2 Pricing Cheat Sheet 575

Storage Costs 575

Tagging EBS Volumes and Snapshots 578

Cost Allocation Tags 579

Storage Performance Comparison 579

Database Costs 581

Database Design Solutions 582

Networking Costs 584

Network Design Solutions 587

Public Versus Private Traffic Charges 588

Data Transfer Costs Cheat Sheet 590

Management Tool Pricing Example: AWS Config 590

AWS Config Results 591

AWS Billing Costs 592

AWS Cost Explorer 593

AWS Budgets 595

Cost Explorer 597

Cost and Usage Report 599

Managing Costs Cheat Sheet 599

Exam Preparation Tasks 600

Review All Key Topics 600

Define Key Terms 601

Q&A 601

Chapter 14 Final Preparation 603

Exam Information 603

Tips for Getting Ready for the Exam 606

Scheduling Your Exam 608

Tools for Final Preparation 609

Pearson Test Prep Practice Test Software and Questions on the Website 609

Accessing the Pearson Test Prep Software Online 609

Accessing the Pearson Test Prep Software Offline 610

Customizing Your Exams 611

Updating Your Exams 612

Premium Edition 612

Chapter-Ending Review Tools 613

Suggested Plan for Final Review/Study 613

Summary 613

Glossary of Key Terms 615

Appendix A Answers to the "Do I Know This Already?" Quizzes and Q&A Sections 623

Appendix B *AWS Certified Solutions Architect - Associate (SAA-C02) Cert Guide* Exam Updates 639

Index 641

Online Elements:

Appendix C Study Planner

Glossary of Key Terms

About the Author

Mark Wilkins is an electronic engineering technologist with a wealth of experience in designing, deploying, and supporting software and hardware technology in the corporate and small business world. Since 2013, Mark has focused on supporting and designing cloud service solutions with Amazon Web Services, Microsoft Azure, and the IBM Cloud. He is certified in Amazon Web Services (Architecture and SysOps). Mark is also a Microsoft Certified Trainer (MCT) and holds certifications in MCTS, MCSA, Server Virtualization with Windows Server Hyper-V, and Azure Cloud Services.

Mark worked as a technical evangelist for IBM SoftLayer from 2013 through 2016 and taught both SoftLayer fundamentals and SoftLayer design classes to many Fortune 500 companies in Canada, the United States, Europe, and Australia. As former course director for Global Knowledge, Mark developed and taught many technical seminars, including Configuring Active Directory Services, Configuring Group Policy, and Cloud and Virtualization Essentials. Mark currently develops AWS curriculum on technical aspects of AWS architecture for O'Reilly Media, Pluralsight, and LinkedIn Learning. To read and discuss all things Mark finds interesting about the cloud visit The Cloud Thingy, at https://thecloudthingy.substack.com/.

Mark's published books include *Windows 2003 Registry for Dummies*, *Administering SMS 3.0*, *Administering Active Directory*, and *Learning Amazon Web Services (AWS): A Hands-On Guide to the Fundamentals of AWS Cloud*.

Dedication

I would like to dedicate this book to my life partner, Jan Wilkins. And to Bruce, one of our cats, for making me take breaks when he wanted.

Acknowledgments

This manuscript was made truly great by the incredible technical review of Akhil Behl.

I would also like to express my gratitude to Chris Cleveland, the development editor of this book. I was lucky to work with him on this text. Chris helped make this book several cuts above the rest.

Finally, thanks so much to Nancy Davis, my tireless acquisitions editor. Nancy very patiently made this book a reality.

About the Technical Reviewer

Akhil Behl, CCIE No. 19564, is a passionate IT executive with a key focus on cloud and security. He has more than 16 years of experience in the IT industry, working in several leadership, advisory, consultancy, and business development roles with various organizations. His technology and business specializations include cloud, security, infrastructure, data center, and business communication technologies.

Akhil is a published author. Over the span of the past few years, Akhil has authored multiple titles on security and business communication technologies. He has contributed as technical editor for more than a dozen books on security, networking, and information technology. He has published several research papers in national and international journals, including *IEEE Xplore*, and presented at various IEEE conferences, as well as other prominent ICT, security, and telecom events. Writing and mentoring are his passions.

Akhil holds CCIE (Collaboration and Security), CCSK, CHFI, PMP, ITIL, VCP, TOGAF, CEH, ISM, CCDP, and many other industry certifications. He has a bachelor's degree in technology and a master's degree in business administration.

We Want to Hear from You!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

We welcome your comments. You can email or write to let us know what you did or didn't like about this book—as well as what we can do to make our books better.

Please note that we cannot help you with technical problems related to the topic of this book.

When you write, please be sure to include this book's title and author as well as your name and email address. We will carefully review your comments and share them with the author and editors who worked on the book.

Email: community@informit.com

Introduction

There are many reasons to get certified in AWS technology. First of all, AWS certifications validate your AWS cloud knowledge. As a starting point for understanding the AWS cloud, the AWS Certified Solutions Architect - Associate (SAA-C02) exam is one great place to start. However, there are other certifications that may be a better fit, depending on your technical level, your current knowledge of cloud concepts, and your current and future jobs with AWS technologies and services. Certifications are broken down into Foundational, Associate, Professional, and Specialty certifications. Full details can be found at https://aws.amazon.com/certification/. AWS frequently adds new certification tracks, but the following are the certifications that are currently available:

- **Foundational:** There is one Foundational certification: Cloud Practitioner. The recommendation is to have at least 6 months of fundamental AWS cloud knowledge before attempting this certification exam. You might be closer to this certification than you think, depending on your current level of technical skills. One advantage of taking the Cloud Practitioner exam first is that it helps you get used to answering multiple-choice test questions and learn about the foundational AWS cloud services.
- **Associate:** There are several Associate certifications:
  - **AWS Certified Solutions Architect - Associate:** For individuals working as solutions architects, designing AWS solutions using AWS services
  - **AWS Certified SysOps Administrator - Associate:** For individuals working as system administrators, managing and operating AWS services
  - **AWS Certified Developer - Associate:** For individuals working as developers, deploying and debugging cloud-based applications hosted at AWS

Each certification exam expects that you know how the AWS service that you are being tested on works. Each Associate certification has a specific focus:

- **Architect:** The best design possible, based on the question and scenario
- **SysOps:** The administration steps required to carry out a particular task
- **Developer:** How to best use the service for the hosted application you are writing

For example, the three Associate exams would test different aspects of CloudWatch logs:

- **Architect:** The main focus of this exam is on how CloudWatch logs work and the main design features to consider based on specific needs—that is, design knowledge related to using CloudWatch logs for a variety of solutions.
- **SysOps:** The main focus of this exam is on how to configure CloudWatch logs based on specific needs—that is, configuration and deployment of CloudWatch logs using operational knowledge.
- **Developer:** The main focus of this exam is on what CloudWatch logs are useful for when developing applications for tracking performance of an application hosted on an EC2 instance—that is, knowledge of how a particular AWS service can help in the development and testing process with applications.

Before you attempt one of the Associate certifications, AWS recommends that you have at least 1 year of experience solving problems and implementing solutions using AWS services. AWS really wants to ensure that you have hands-on experience solving problems.

- **Professional:** These certifications include the AWS Certified Solutions Architect - Professional and the AWS Certified DevOps Engineer - Professional. Professional certifications are not where you normally start your certification journey. AWS recommends that you have at least 2 years of hands-on experience before taking these tough Professional exams.
- **Specialty:** The Specialty certifications for Advanced Networking, Security, Machine Learning, Data Analytics, and Database require advanced knowledge of the subject matter. AWS recommends that you have an Associate certification before you attempt one of these certifications.

NOTE The AWS Certified Solutions Architect - Associate (SAA-C02) certification is globally recognized and does an excellent job of demonstrating that the holder has knowledge and skills across a broad range of AWS topics.

The Goals of the AWS Certified Solutions Architect - Associate Certification

The AWS Certified Solutions Architect - Associate certification is intended for individuals who perform in a solutions architect role. This exam validates a candidate's ability to effectively demonstrate knowledge of how to architect and deploy secure and robust applications on AWS technologies. It validates a candidate's ability to

- Define a solution using architectural design principles based on customer requirements
- Provide implementation guidance based on best practices to an organization throughout the lifecycle of a project

Recommended Prerequisite Skills

While this book provides you with the information required to pass the Certified Solutions Architect - Associate (SAA-C02) exam, Amazon considers ideal candidates to be those who possess the following:

- At least 1 year of hands-on experience designing highly available, cost-effective, fault-tolerant, and scalable distributed systems on AWS
- Hands-on experience using compute, networking, storage, and database AWS services
- Hands-on experience with AWS deployment and management services
- Ability to identify and define technical requirements for an AWS-based application
- Ability to identify which AWS services meet a given technical requirement
- Knowledge of recommended best practices for building secure and reliable applications on the AWS platform
- An understanding of the basic architectural principles involved in building in AWS Cloud
- An understanding of the AWS global infrastructure
- An understanding of network technologies related to AWS
- An understanding of security features and tools that AWS provides and how they relate to traditional services

The Exam Objectives (Domains)

The AWS Certified Solutions Architect - Associate (SAA-C02) exam is broken down into five major domains. This book covers each of the domains and the subtopics included in them.

The following table lists the breakdown of the domains represented on the exam:

Domain	Percentage of Representation in Exam
Domain 1: Design Resilient Architectures	30%
Domain 2: Design High-Performing Architectures	28%
Domain 3: Design Secure Applications and Architectures	24%
Domain 4: Design Cost-Optimized Architectures	18%
	Total 100%

Each domain includes a number of objectives:

Domain 1: Design Resilient Architectures: This domain is covered in Chapters 2–5

1.1 Design a multi-tier architecture solution

1.2 Design highly available and/or fault-tolerant architectures

1.3 Design decoupling mechanisms using AWS services

1.4 Choose appropriate resilient storage

Domain 2: Design High-Performing Architectures: This domain is covered in Chapters 6–9

2.1 Identify elastic and scalable compute solutions for a workload

2.2 Select high-performing and scalable storage solutions for a workload

2.3 Select high-performing networking solutions for a workload

2.4 Choose high-performing database solutions for a workload

Domain 3: Design Secure Applications and Architectures: This domain is covered in Chapters 10–12

3.1 Design secure access to AWS resources

3.2 Design secure application tiers

3.3 Select appropriate data security options

Domain 4: Design Cost-Optimized Architectures: This domain is covered in Chapter 13

4.1 Identify cost-effective storage solutions

4.2 Identify cost-effective compute and database services

4.3 Design cost-optimized network architectures

Steps to Becoming an AWS Certified Solutions Architect - Associate

To become an AWS Certified Solutions Architect - Associate, an exam candidate must meet certain prerequisites and follow specific procedures. Exam candidates must ensure that they have the necessary background and technical experience for the exam and then sign up for the exam.

Signing Up for the Exam

The steps required to sign up for the AWS Certified Solutions Architect - Associate exam are as follows:

Step 1. Create an AWS Certification account at https://www.aws.training/Certification and schedule your exam from the home page by clicking **Schedule New Exam**.

Step 2. Select a testing provider, either Pearson VUE or PSI, and select whether you want to take the exam at a local testing center or online from your home or office. If you choose to take an online exam, you will have to agree to the online testing policies.

Step 3. Complete the examination signup by selecting the preferred language and the date of your exam.

Step 4. Submit the examination fee.

TIP Refer to the AWS Certification site at https://aws.amazon.com/certification/ for more information regarding this and other AWS certifications.

What This Book Covers

This book maps directly to the domains of the AWS Certified Solutions Architect - Associate (SAA-C02) exam and includes a number of features that help you understand the topics and prepare for the exam.

Objectives and Methods

This book uses several key methodologies to help you discover the exam topics on which you need more review, to help you fully understand and remember those details, and to help you ensure that you have retained your knowledge of those topics. This book does not try to help you pass the exam only by memorization; it seeks to help you truly learn and understand the topics. This book is designed to help you pass the AWS Certified Solutions Architect - Associate (SAA-C02) exam by using the following methods:

- Helping you discover which exam topics you have not mastered
- Providing explanations and information to fill in your knowledge gaps
- Supplying exercises that enhance your ability to recall and deduce the answers to test questions
- Providing practice exercises on the topics and the testing process via test questions on the companion website

Book Features

To help you customize your study time using this book, the core chapters have several features that help you make the best use of your time:

- **Foundation Topics:** The sections under “Foundation Topics” describe the core topics of each chapter.
- **Exam Preparation Tasks:** The “Exam Preparation Tasks” section lists a series of study activities that you should do at the end of each chapter:
  - **Review All Key Topics:** The Key Topic icon appears next to the most important items in the “Foundation Topics” section of the chapter. The “Review All Key Topics” activity lists the key topics from the chapter, along with the number of the page where you can find more information about each one. Although the contents of the entire chapter could be tested on the exam, you should definitely know the information listed in each key topic, so you should review these.
  - **Define Key Terms:** Although the AWS Certified Solutions Architect - Associate (SAA-C02) exam may be unlikely to word a question “Define this term,” the exam does require that you learn and know a lot of terminology. This section lists the most important terms from the chapter and asks you to write a short definition and compare your answer to the glossary at the end of the book.
  - **Q&A:** Confirm that you understand the content that you just covered by answering these questions and reading the answer explanations.

- **Web-based practice exam:** The companion website includes the Pearson Test Prep practice test engine, which enables you to take practice exam questions. Use it to prepare with a sample exam and to pinpoint topics where you need more study.

How This Book Is Organized

This book contains 12 core chapters—Chapters 2 through 13. Chapter 1 introduces the foundations of AWS, and Chapter 14 provides preparation tips and suggestions for how to approach the exam. Each core chapter covers a subset of the topics on the AWS Certified Solutions Architect - Associate (SAA-C02) exam:

- Chapters 2–5 cover the topics in Domain 1.
- Chapters 6–9 cover the topics in Domain 2.
- Chapters 10–12 cover the topics in Domain 3.
- Chapters 13 covers all the topics in Domain 4.

Companion Website

Register this book to get access to the Pearson Test Prep practice test software and other study materials plus additional bonus content. Check this site regularly for new and updated postings written by the author that provide further insight into the more troublesome topics on the exam. Be sure to check the box indicating that you would like to hear from us to receive updates and exclusive discounts on future editions of this product or related products.

To access this companion website, follow these steps:

Step 1. Go to **www.pearsonitcertification.com/register** and log in or create a new account.

Step 2. Enter the ISBN **9780137325214**.

Step 3. Answer the challenge question as proof of purchase.

Step 4. Click the **Access Bonus Content** link in the Registered Products section of your account page to be taken to the page where your downloadable content is available.

Please note that many of our companion content files can be very large, especially image and video files.

If you are unable to locate the files for this title by following these steps, please visit www.pearsonITcertification.com/contact and select the Site Problems/Comments option. Our customer service representatives will assist you.

Pearson Test Prep Practice Test Software

As noted earlier, the Pearson Test Prep practice test software comes with two full practice exams. These practice tests are available to you either online or as an offline Windows application. To access the practice exams that were developed with this book, see the instructions in the card inserted in the sleeve at the back of the book. This card includes a unique access code that enables you to activate your exams in the Pearson Test Prep practice test software. For more information about the practice tests and more tools for exam preparation, see Chapter 14.

Figure Credits

Cover image © whiteMocca/Shutterstock

Chapter opener art © Charlie Edwards/Photodisc/Getty Images

Figure 1-2, screenshot of the AWS Management Console © 2021, Amazon Web Services, Inc

Figure 1-4, screenshot of using AWS budgets to track and alert when costs are over budget © 2021, Amazon Web Services, Inc

Figure 1-5, screenshot of Infrastructure as a Service at AWS © 2021, Amazon Web Services, Inc

Figure 1-7, screenshot of platform options at AWS © 2021, Amazon Web Services, Inc

Figure 1-8, screenshot of encrypting S3 buckets using AES-256 or AWS-KMS managed keys © 2021, Amazon Web Services, Inc

Figure 1-9, screenshot of encrypted traffic flow at AWS © 2021, Amazon Web Services, Inc

Figure 1-10, screenshot of Cloud9 IDE at AWS © 2021, Amazon Web Services, Inc

Figure 1-11, screenshot of using the Well-Architected Framework tool © 2021, Amazon Web Services, Inc

Figure 1-12, screenshot of recommended improvements using the Well-Architected Framework tool review © 2021, Amazon Web Services, Inc

Figure 2-1, screenshot of AWS regions and geographic locations © 2021, Amazon Web Services, Inc

Figure 2-3, screenshot of visualizing an AWS region and supporting services © 2021, Amazon Web Services, Inc

Figure 2-5, screenshot of AZ balancing and distribution of resources © 2021, Amazon Web Services, Inc

Figure 2-6, private network wiring for AWS regions and AZs courtesy of Amazon Web Services, Inc

Figure 2-7, failover possibilities of availability zones for application stacks courtesy of Amazon Web Services, Inc

Figure 2-8, screenshot of reviewing audit reports in the AWS Management Console © 2021, Amazon Web Services, Inc

Figure 2-9, screenshot of authenticating to the GovCloud region © 2021, Amazon Web Services, Inc

Figure 2-14, screenshot of Route 53 traffic flow policies © 2021, Amazon Web Services, Inc

Figure 2-16, forwarding rules courtesy of Amazon Web Services, Inc

Figure 2-17, conditional forwarding rules courtesy of Amazon Web Services, Inc

Figure 2-19, CloudFront operation courtesy of Amazon Web Services, Inc

Figure 2-20, CloudFront regional cache architecture courtesy of Amazon Web Services, Inc

Figure 2-23, screenshot of trusted advisor service limits summary © 2021, Amazon Web Services, Inc

Figure 3-12, screenshot of Aurora DB cluster with multiple writes © 2021, Amazon Web Services, Inc

Figure 3-13, screenshot of the CloudFormation console © 2021, Amazon Web Services, Inc

Figure 3-14, screenshot of AWS sample stacks and CloudFormation templates © 2021, Amazon Web Services, Inc

Figure 3-17, screenshot of portfolios in service catalog © 2021, Amazon Web Services, Inc

Figure 3-18, screenshot of IAM group constraints controlled by service catalog © 2021, Amazon Web Services, Inc

Figure 3-19, screenshot of Elastic Beanstalk creating infrastructure and installing an application © 2021, Amazon Web Services, Inc

Figure 3-20, screenshot of modifying the capacity of the Elastic Beanstalk application infrastructure © 2021, Amazon Web Services, Inc

Figure 3-21, screenshot of apply rolling updates to an Elastic Beanstalk application © 2021, Amazon Web Services, Inc

Figure 3-23, screenshot of a CodeCommit repository © 2021, Amazon Web Services, Inc

Figure 3-24, screenshot of using SQS queues to provide stateless memory-resident storage for applications © 2021, Amazon Web Services, Inc

Figure 4-5, SNS Publisher and subscriber push options courtesy of Amazon Web Services, Inc

Figure 4-6, screenshot of creating a notification topic © 2021, Amazon Web Services, Inc

Figure 4-7, SQS and SNS working together courtesy of Amazon Web Services, Inc

Figure 4-8, response and request queues decoupling application state information courtesy of Amazon Web Services, Inc

Figure 4-10, screenshot of creating a custom Lambda function for a specific task © 2021, Amazon Web Services, Inc

Figure 4-13, screenshot of Lambda function execution settings © 2021, Amazon Web Services, Inc

Figure 4-15, screenshot of choosing the API protocol to use © 2021, Amazon Web Services, Inc

Figure 4-16, screenshot of selecting an authorizer for the API gateway © 2021, Amazon Web Services, Inc

Figure 4-18, screenshot of using an S3 bucket for static website hosting © 2021, Amazon Web Services, Inc

Figure 4-19, screenshot of creating an authentication pool using Cognito © 2021, Amazon Web Services, Inc

Figure 4-20, screenshot of creating a DynamoDB table © 2021, Amazon Web Services, Inc

Figure 4-21, screenshot of registering the RESTful API with the API gateway © 2021, Amazon Web Services, Inc

Figure 5-3, optimizing MySQL running on Amazon EC2 using Amazon EBS, September 2017 © Amazon Web Services, Inc

Figure 5-4, screenshot of modifying EBS volumes © 2021, Amazon Web Services, Inc

Figure 5-5, screenshot of creating a snapshot © 2021, Amazon Web Services, Inc

Figure 5-6, screenshot of instance storage architecture © 2021, Amazon Web Services, Inc

Figure 5-7, screenshot of selecting EFS performance © 2021, Amazon Web Services, Inc

Figure 5-8, screenshot of EFS security © 2021, Amazon Web Services, Inc

Figure 5-9, screenshot of EFS lifecycle policy © 2021, Amazon Web Services, Inc

Figure 5-10, screenshot of FSx setup © 2021, Amazon Web Services, Inc

Figure 5-11, screenshot of S3 object details © 2021, Amazon Web Services, Inc

Figure 5-12, screenshot of creating an S3 inventory report © 2021, Amazon Web Services, Inc

Figure 5-13, screenshot of enabling object lock on an S3 bucket © 2021, Amazon Web Services, Inc

Figure 5-14, screenshot of enabling cross-region replication © 2021, Amazon Web Services, Inc

Figure 5-15, screenshot of lifecycle rules © 2021, Amazon Web Services, Inc

Figure 5-16, screenshot of creating S3 Glacier vaults © 2021, Amazon Web Services, Inc

Figure 6-4, screenshot of AWS Instance choices © 2021, Amazon Web Services, Inc

Figure 6-5, screenshot of t2 unlimited mode © 2021, Amazon Web Services, Inc

Figure 6-6, screenshot of creating a launch template © 2021, Amazon Web Services, Inc

Figure 6-7, screenshot of changing the current instance type © 2021, Amazon Web Services, Inc

Figure 6-8, screenshot of default instance volumes © 2021, Amazon Web Services, Inc

Figure 6-9, screenshot of using Lightsail to order an application stack © 2021, Amazon Web Services, Inc

Figure 6-11, screenshot of Amazon Linux 2 in the EC2 console © 2021, Amazon Web Services, Inc

Figure 6-12, screenshot of AWS Marketplace AMI choices © 2021, Amazon Web Services, Inc

Figure 6-13, screenshot of AMI creation © 2021, Amazon Web Services, Inc

Figure 6-17, screenshot of task definition options © 2021, Amazon Web Services, Inc

Figure 6-18, screenshot of CloudWatch metrics © 2021, Amazon Web Services, Inc

Figure 6-19, screenshot of basic metrics for S3 buckets displayed in CloudWatch © 2021, Amazon Web Services, Inc

Figure 6-20, screenshot of CloudWatch metric choices © 2021, Amazon Web Services, Inc

Figure 6-21, screenshot of VPC Flow Logs integrated with CloudTrail © 2021, Amazon Web Services, Inc

Figure 6-22, screenshot of CloudWatch log data export options © 2021, Amazon Web Services, Inc

Figure 6-23, screenshot of EBS CloudWatch metrics © 2021, Amazon Web Services, Inc

Figure 6-24, screenshot of EC2 instance metrics © 2021, Amazon Web Services, Inc

Figure 6-25, screenshot of defining the metric evaluation period © 2021, Amazon Web Services, Inc

Figure 6-26, screenshot of data points summarized every 5 minutes © 2021, Amazon Web Services, Inc

Figure 6-27, screenshot of defining a CloudWatch metric filter © 2021, Amazon Web Services, Inc

Figure 6-28, screenshot of CloudWatch and EC2 auto scaling © 2021, Amazon Web Services, Inc

Figure 6-29, screenshot of defining metric behaviors © 2021, Amazon Web Services, Inc

Figure 6-30, screenshot of setting CloudWatch alarm details © 2021, Amazon Web Services, Inc

Figure 6-31, screenshot of ELB load balancer choices © 2021, Amazon Web Services, Inc

Figure 6-32, screenshot of load balancer health check settings © 2021, Amazon Web Services, Inc

Figure 6-33, screenshot of adding a target group to ALB © 2021, Amazon Web Services, Inc

Figure 6-34, screenshot of initial configuration of ALB © 2021, Amazon Web Services, Inc

Figure 6-35, screenshot of choosing AZs and subnets © 2021, Amazon Web Services, Inc

Figure 6-36, screenshot of default ALB rule for forwarding traffic © 2021, Amazon Web Services, Inc

Figure 6-37, screenshot of host- and path-based rules defined for precise traffic flow © 2021, Amazon Web Services, Inc

Figure 6-38, screenshot of choosing a security certificate for ALB © 2021, Amazon Web Services, Inc

Figure 6-39, screenshot of target group attributes © 2021, Amazon Web Services, Inc

Figure 6-40, screenshot of managing SSL/TLS certificates with AWS Certificate Manager © 2021, Amazon Web Services, Inc

Figure 6-41, screenshot of connection draining to deregister instances from target groups © 2021, Amazon Web Services, Inc

Figure 6-42, screenshot of launch templates © 2021, Amazon Web Services, Inc

Figure 6-43, screenshot of auto scaling group settings and options © 2021, Amazon Web Services, Inc

Figure 6-44, screenshot of health check options © 2021, Amazon Web Services, Inc

Figure 6-45, screenshot of target tracking variables for increasing group size © 2021, Amazon Web Services, Inc

Figure 6-46, screenshot of simple scaling parameters © 2021, Amazon Web Services, Inc

Figure 6-47, screenshot of step scaling parameters © 2021, Amazon Web Services, Inc

Figure 7-1, screenshot of changing IOPS values on the fly © 2021, Amazon Web Services, Inc

Figure 7-2, EFS Architecture courtesy of Amazon Web Services, Inc

Figure 7-3, screenshot of EFS Performance mode architecture © 2021, Amazon Web Services, Inc

Figure 7-4, file system components that affect performance courtesy of Amazon Web Services, Inc

Figure 7-5, FSx for Lustre architecture courtesy of Amazon Web Services, Inc

Figure 7-6, screenshot of creating a snowball job © 2021, Amazon Web Services, Inc

Figure 8-1, screenshot of the VPC console © 2021, Amazon Web Services, Inc

Figure 8-7, screenshot of using the Create VPC Wizard © 2021, Amazon Web Services, Inc

Figure 8-8, screenshot of VPC starting design choices © 2021, Amazon Web Services, Inc

Figure 8-9, screenshot of creating a VPC by using the CLI © 2021, Amazon Web Services, Inc

Figure 8-11, screenshot of the default VPC © 2021, Amazon Web Services, Inc

Figure 8-13, screenshot of the main route table © 2021, Amazon Web Services, Inc

Figure 8-15, screenshot of using custom route tables © 2021, Amazon Web Services, Inc

Figure 8-16, screenshot of the AWS public IP address pool © 2021, Amazon Web Services, Inc

Figure 8-17, screenshot of elastic IP addresses © 2021, Amazon Web Services, Inc

Figure 8-18, screenshot of assigned IP addresses and DNS hostnames © 2021, Amazon Web Services, Inc

Figure 8-20, screenshot of BYOIP address architecture at AWS © 2021, Amazon Web Services, Inc

Figure 8-21, screenshot of an advertised BYOIP address range © 2021, Amazon Web Services, Inc

Figure 8-22, screenshot of security group details © 2021, Amazon Web Services, Inc

Figure 8-23, screenshot of default security group in a newly created VPC © 2021, Amazon Web Services, Inc

Figure 8-25, screenshot of flow log storage location choices © 2021, Amazon Web Services, Inc

Figure 8-26, screenshot of peering traffic flow © 2021, Amazon Web Services, Inc

Figure 8-32, screenshot of creating a NAT gateway © 2021, Amazon Web Services, Inc

Figure 8-33, screenshot of transit gateway attachments © 2021, Amazon Web Services, Inc

Figure 9-1, screenshot of changing database instance parameters © 2021, Amazon Web Services, Inc

Figure 9-4, screenshot of Aurora deployment options © 2021, Amazon Web Services, Inc

Figure 9-10, screenshot of a DynamoDB table © 2021, Amazon Web Services, Inc

Figure 9-11, screenshot of adjusting table capacity © 2021, Amazon Web Services, Inc

Figure 9-12, screenshot of DynamoDB auto scaling settings © 2021, Amazon Web Services, Inc

Figure 9-15, screenshot of DynamoDB global tables © 2021, Amazon Web Services, Inc

Figure 9-18, screenshot of creating a Redshift cluster © 2021, Amazon Web Services, Inc

Figure 10-1, screenshot of the IAM console © 2021, Amazon Web Services, Inc

Figure 10-2, screenshot of using service-linked roles in IAM © 2021, Amazon Web Services, Inc

Figure 10-3, screenshot of IAM user account access keys © 2021, Amazon Web Services, Inc

Figure 10-4, screenshot of an ARN © 2021, Amazon Web Services, Inc

Figure 10-7, screenshot of actions approved by IAM © 2021, Amazon Web Services, Inc

Figure 10-8, screenshot of root user logon © 2021, Amazon Web Services, Inc

Figure 10-9, screenshot of creating an IAM user © 2021, Amazon Web Services, Inc

Figure 10-10, screenshot of access keys required for CLI operation © 2021, Amazon Web Services, Inc

Figure 10-11, screenshot of IAM user account creation options © 2021, Amazon Web Services, Inc

Figure 10-12, screenshot of using custom URL for IAM users © 2021, Amazon Web Services, Inc

Figure 10-13, screenshot of user account summary information © 2021, Amazon Web Services, Inc

Figure 10-14, screenshot of password policy options © 2021, Amazon Web Services, Inc

Figure 10-15, screenshot of creating an additional access key manually © 2021, Amazon Web Services, Inc

Figure 10-16, screenshot of policy settings for rotating access keys © 2021, Amazon Web Services, Inc

Figure 10-17, screenshot of managed policies © 2021, Amazon Web Services, Inc

Figure 10-18, screenshot of job functions policies © 2021, Amazon Web Services, Inc

Figure 10-20, screenshot of the JSON editor © 2021, Amazon Web Services, Inc

Figure 10-21, screenshot of version information © 2021, Amazon Web Services, Inc

Figure 10-22, screenshot of adding a permission boundary to a user account © 2021, Amazon Web Services, Inc

Figure 10-25, screenshot of policy summary tables © 2021, Amazon Web Services, Inc

Figure 10-26, screenshot of viewing versions of IAM policies © 2021, Amazon Web Services, Inc

Figure 10-27, screenshot of attaching an IAM role to an EC2 instance © 2021, Amazon Web Services, Inc

Figure 10-28, screenshot of using Cognito for mobile user authentication © 2021, Amazon Web Services, Inc

Figure 10-29, screenshot of using the switch role option for cross-account access © 2021, Amazon Web Services, Inc

Figure 10-30, screenshot of changing the validity time frame for temporary credentials © 2021, Amazon Web Services, Inc

Figure 10-31, screenshot of adding a metadata XML file © 2021, Amazon Web Services, Inc

Figure 10-32, screenshot of selecting policies for an IAM role © 2021, Amazon Web Services, Inc

Figure 10-33, screenshot of a properly set up root account © 2021, Amazon Web Services, Inc

Figure 10-34, screenshot of credential report © 2021, Amazon Web Services, Inc

Figure 10-35, screenshot of access advisor details © 2021, Amazon Web Services, Inc

Figure 10-36, screenshot of IAM policy simulator © 2021, Amazon Web Services, Inc

Figure 10-37, screenshot of OUs in AWS organizations © 2021, Amazon Web Services, Inc

Figure 10-38, screenshot of sharing subnets with AWS RAM © 2021, Amazon Web Services, Inc

Figure 11-1, screenshot of detailed CloudTrail event © 2021, Amazon Web Services, Inc

Figure 11-2, screenshot of creating a CloudTrail trail © 2021, Amazon Web Services, Inc

Figure 11-3, screenshot of CloudTrail authentication event © 2021, Amazon Web Services, Inc

Figure 11-4, screenshot of storing RDS credentials as a secret © 2021, Amazon Web Services, Inc

Figure 11-5, screenshot of GuardDuty settings © 2021, Amazon Web Services, Inc

Figure 11-6, screenshot of Amazon Inspector options © 2021, Amazon Web Services, Inc

Figure 11-7, screenshot of Trusted Advisor security checks © 2021, Amazon Web Services, Inc

Figure 11-8, screenshot of Trusted Advisor results © 2021, Amazon Web Services, Inc

Figure 12-2, screenshot of creating a new master key when enabling encryption © 2021, Amazon Web Services, Inc

Figure 12-3, screenshot of creating a new master key © 2021, Amazon Web Services, Inc

Figure 12-5, screenshot of S3 ACL settings © 2021, Amazon Web Services, Inc

Figure 12-6, screenshot of blocking public access on an S3 bucket by default © 2021, Amazon Web Services, Inc

Figure 12-9, screenshot of WORM policy settings © 2021, Amazon Web Services, Inc

Figure 12-10, screenshot of KMS console © 2021, Amazon Web Services, Inc

Figure 12-11, screenshot of generating CMKs with KMS for an RDS instance © 2021, Amazon Web Services, Inc

Figure 12-13, screenshot of certificate choices in Amazon Certificate Manager © 2021, Amazon Web Services, Inc

Figure 13-1, screenshot of comparing regional prices © 2021, Amazon Web Services, Inc

Figure 13-2, screenshot of requesting a quota change © 2021, Amazon Web Services, Inc

Figure 13-3, screenshot of the limits calculator © 2021, Amazon Web Services, Inc

Figure 13-4, screenshot of reserved instance pricing © 2021, Amazon Web Services, Inc

Figure 13-5, screenshot of reserved instance options © 2021, Amazon Web Services, Inc

Figure 13-6, screenshot of spot instance pricing options © 2021, Amazon Web Services, Inc

Figure 13-7, screenshot of spot fleet allocation request © 2021, Amazon Web Services, Inc

Figure 13-8, screenshot of spot capacity pools © 2021, Amazon Web Services, Inc

Figure 13-9, screenshot of S3 storage pricing © 2021, Amazon Web Services, Inc

Figure 13-10, screenshot of EBS price calculations © 2021, Amazon Web Services, Inc

Figure 13-11, screenshot of cost allocation tags © 2021, Amazon Web Services, Inc

Figure 13-13, screenshot of DynamoDB scaling policy © 2021, Amazon Web Services, Inc

Figure 13-15, screenshot of traffic charges at AWS © 2021, Amazon Web Services, Inc

Figure 13-16, screenshot of accessing the billing and cost management console © 2021, Amazon Web Services, Inc

Figure 13-17, screenshot of cost explorer recommendations © 2021, Amazon Web Services, Inc

Figure 13-18, screenshot of budget choices © 2021, Amazon Web Services, Inc

Figure 13-19, screenshot of cost explorer details © 2021, Amazon Web Services, Inc

Figure 13-20, screenshot of enabling billing alerts © 2021, Amazon Web Services, Inc

Figure 14-1, screenshot of schedule your AWS exam © 2021, Amazon Web Services, Inc

Figure 14-2, screenshot of viewing your AWS training profile © 2021, Amazon Web Services, Inc

CHAPTER 3

Highly Available and Fault-Tolerant Architectures

This chapter covers content that's important to the following exam domain/objective:

- **Domain 1: Design Resilient Architectures**

  Objective 1.2 Design highly available and/or fault-tolerant architectures

The terms *high availability* and *fault tolerant* take on unique meanings in today's hosted cloud. Amazon wants to make sure that you understand how these terms differ in a cloud environment from in on-premises situation. For example, you might be accustomed to using *clustering* as a common method to protect database data. In the cloud, clustering has been mainly been replaced with horizontal scaling and placing multiple copies of databases in different data centers. You could still build a cluster in the cloud, but there are better ways to achieve high availability and fault tolerance—or at least Amazon and many customers feel there are better ways. Ultimately, no matter the terminology used, your application should be always up and available when the end user needs it. In addition, your data records should always be available. And both your application and your data should be able to handle problems in the backend; you need compute power for the many web servers that are available to take over, and your data needs to be stored in numerous physical locations.

The questions on the AWS Certified Solutions Architect - Associate (SAA-C02) exam quantify your understanding of the services at AWS that enable you to design your application stacks to be highly available and very fault tolerant.

"Do I Know This Already?" Quiz

The "Do I Know This Already?" quiz allows you to assess whether you should read this entire chapter thoroughly or jump to the "Exam Preparation Tasks" section. If you are in doubt about your answers to these questions or your own assessment of your knowledge of the topics, read the entire chapter. Table 3-1 lists the major headings in this chapter and their corresponding "Do I Know This Already?" quiz questions. You can find the answers in Appendix A, "Answers to the 'Do I Know This Already?' Quizzes and Q&A Sections."

Table 3-1 "Do I Know This Already?" Section-to-Question Mapping

Foundation Topics Section	Questions
Comparing Architecture Designs	1, 2
Disaster Recovery and Business Continuity	3, 4
Automating Architecture	5, 6
Elastic Beanstalk	7, 8

CAUTION The goal of self-assessment is to gauge your mastery of the topics in this chapter. If you do not know the answer to a question or are only partially sure of the answer, you should mark that question as wrong for purposes of the self-assessment. Giving yourself credit for an answer you correctly guess skews your self-assessment results and might provide you with a false sense of security.

1. Which of the following is a characteristic of applications that are designed to be highly available?
   a. Minimal downtime
   b. Increased performance
   c. Higher level of security
   d. Cost-effective operation

2. Which of the following is a characteristic of applications that are designed with a high level of fault tolerance?
   a. Automatic recovery from most failures
   b. Higher speed
   c. Higher performance
   d. Cost-effective operation

3. What does the term RPO stand for?
   a. Reliability point objective
   b. Recovery point objective
   c. Restore point objective
   d. Return to an earlier phase of operation

4. What does the term RTO stand for?
   a. Recovery tier objective
   b. Recovery time objective
   c. The amount of data that will be lost
   d. The time the application will be down

5. What is the purpose of using CloudFormation?
   a. To recover from failures
   b. To automate the building of AWS infrastructure components
   c. To deploy applications with automation
   d. To document manual tasks

6. What CloudFormation component is used to check deployment changes to existing infrastructure?
   a. CloudFormation template
   b. Stack
   c. Change set
   d. JSON script

7. What two components does Elastic Beanstalk deploy?
   a. Infrastructure and storage
   b. Application and infrastructure
   c. Compute and storage
   d. Containers and instances

8. What term defines application updates that are applied to a new set of EC2 instances?
   a. Rolling
   b. Immutable
   c. All at once
   d. Blue/green

Foundation Topics

Comparing Architecture Designs

Successful applications hosted at AWS have some semblance of high availability and/or fault tolerance in the underlying design. As you prepare for the AWS Certified Solutions Architect - Associate (SAA-C02) exam, you need to understand these concepts conceptually in conjunction with the services that can provide a higher level of redundancy and availability. After all, if an application is designed to be highly available, it will have a higher level of redundancy; therefore, regardless of the potential failures that may happen from time to time, your application will continue to function at a working level without massive, unexpected downtime.

The same is true with a fault-tolerant design: If your application has tolerance for any operating issues such as latency, speed, and durability, your application design will be able to overcome most of these issues most of the time. The reality is that nothing is perfect, of course, but you can ensure high availability and fault tolerance for hosted applications by properly designing the application.

Designing for High Availability

Imagine that your initial foray into the AWS cloud involves hosting an application on a single large EC2 instance. This would be the simplest approach in terms of architecture, but there will eventually be issues with availability, such as the instance failing or needing to be taken down for maintenance. There are also design issues related to application state and data records stored on the same local instance.

As a first step, you could split the application into a web tier and a database tier with separate instances to provide some hardware redundancy; this improves availability because the two systems are unlikely to fail at the same time. If the mandate for this application is that it should be down for no more than 44 hours in any given 12-month period, this equates to 99.5% uptime. Say that tests show that the web server actually has 90% availability, and the database server has 95% reliability, as shown in Figure 3-1. Considering that the hardware, operating system, database engine, and network connectivity all have the same availability level, this design results in a total availability of 85.5%.

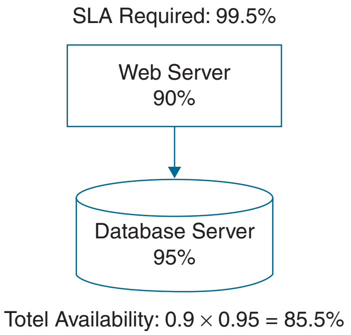


FIGURE 3-1 Availability Calculation for a Simple Hosted Application

This amount of uptime and availability obviously needs to be increased to meet the availability mandate. Adding an additional web server to the design increases the overall availability to 94.5%, as shown in Figure 3-2.

Key Topic

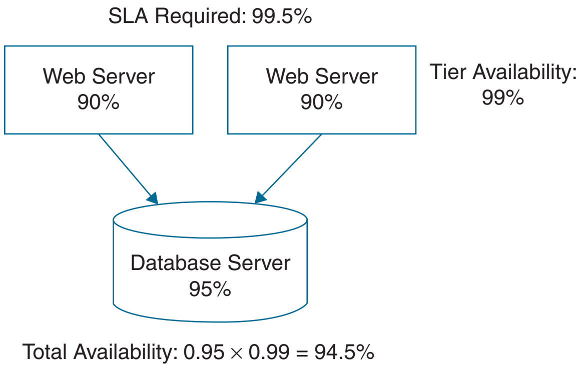


FIGURE 3-2 Increasing Application Availability by Adding Compute

The design now has some additional fault tolerance for the web tier so that one server can fail, but the application will still be available. However, you also need to increase the database tier availability because, after all, it's where the important data is stored. Adding to the existing database infrastructure by adding two replica database servers in addition to adding a third web server to the web server tier results in both tiers achieving a total availability figure of 99.8% and achieving the SLA goal, as shown in Figure 3-3.

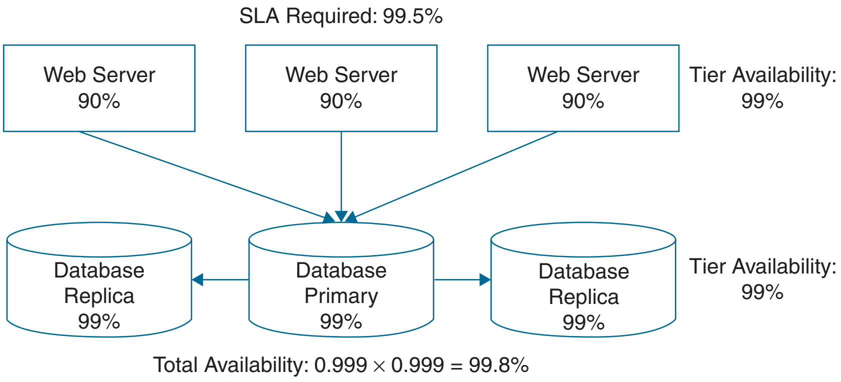


FIGURE 3-3 Adding Availability to Both Tiers

Adding Fault Tolerance

You can apply the same availability principles just discussed to the AWS data centers that host the application tiers. To do so, you increase application availability and add fault tolerance by designing with multiple availability zones, as shown in Figure 3-4.

Key Topic

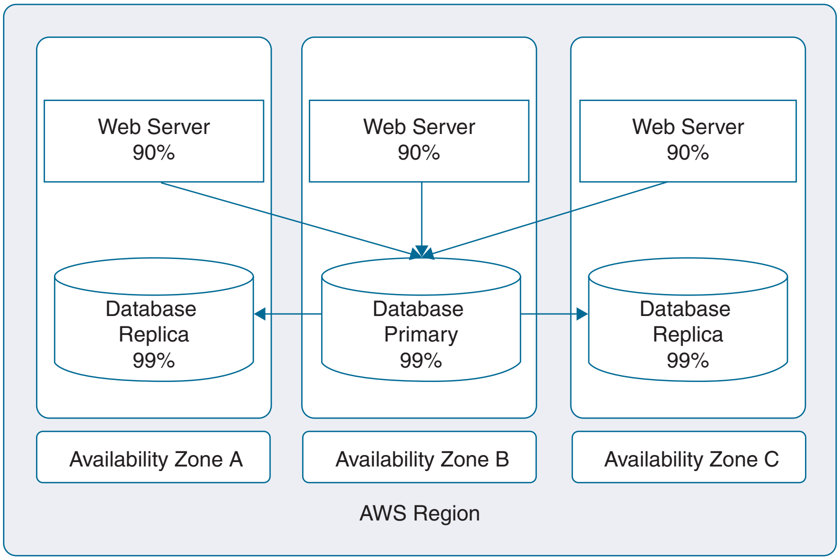


FIGURE 3-4 Hosting Across Multiple AZ's to Increase Availability and Fault Tolerance

Remember that each AWS availability zone has at least one physical data center. AWS data centers are built for high-availability designs, are connected together with high-speed/low-latency links, and are located far enough away from each other that a single natural disaster won't affect all of the data centers in each AWS region at the same time. Thanks to this design, your application has high availability, and because of the additional availability, the application has more tolerance for failures when they occur.

Removing Single Points of Failure

Eliminating as many single points of failure as possible in your application stack design will also greatly increase high availability and fault tolerance. A single point of failure can be defined as any component of your application stack that will cause the rest of your integrated application to fail if that particular individual component fails. Take some time to review Table 3-2, which discusses possible mitigation paths for single points of failure.

Key Topic

Table 3-2 Avoiding Single Points of Failure

Possible Single Point of Failures	Mitigation Plan	Reason
On-premises DNS	Route 53 DNS	Anycast DNS services hosted across all AWS regions. Health checks and DNS failover.
Third-party load balancer	Elastic Load Balancing (ELB) services	ELB instances form a massive regional server farm with EIP addresses for fast failover.
Web/app server	ELB/Auto Scaling for each tier	Scale compute resources automatically up and down to meet demand.
RDS database servers	Redundant data nodes (primary/standby)	Synchronized replication between primary and standby nodes provides two copies of updated data
EBS data storage	Snapshots and retention schedule	Copy snapshots across regions adding additional redundancy.
Authentication problem	Redundant authentication nodes	Multiple Active Directory domain controllers provide alternate authentication options.
Data center failure	Multiple availability zones	Each region has multiple AZs providing high-availability and failover design options.
Regional disaster	Multi-region deployment with Route 53	Route 53 routing policy provides geo-redundancy for applications hosted across regions.

AWS recommends that you use a load balancer to balance the load of requests between multiple servers and to increase overall availability and reliability. The servers themselves are physically located in separate availability zones targeted directly by the load balancer. However, a single load-balancing appliance could be a single point of failure; if the load balancer failed, there would be no access to the application. You could add a second load balancer, but doing so would make the design more complicated as failover to the redundant load balancer would require a DNS update for the client, and that would take some time. AWS gets around this problem through the use of elastic IP addresses that allow for rapid IP remapping, as shown in Figure 3-5. The elastic IP address is assigned to multiple load balancers, and if one load balancer is not available ,the elastic IP address attaches itself to another load balancer.

Key Topic

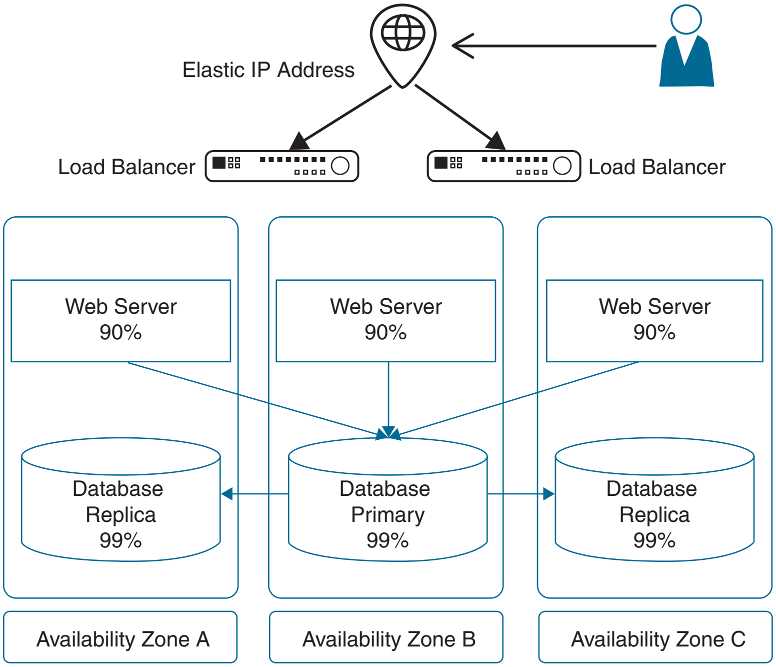


FIGURE 3-5 Using Elastic IP Addresses to Provide High Availability

You can think of the elastic IP address as being able to float between resources as required. This software component is the mainstay of providing high-availability infrastructure at AWS. You can read further details about elastic IP addresses in Chapter 8, "Networking Solutions for Workloads."

Table 3-3 lists AWS services that can be improved with high availability, fault tolerance, and redundancy.

Key Topic

Table 3-3 Planning for High Availability, Fault Tolerance, and Redundancy

AWS Service	High Availability	Fault Tolerance	Redundancy	Multi-Region
EC2 instance	Additional instance	Multiple availability zones	Auto Scaling	Route 53 health checks
EBS volume	Cluster design	Snapshots	AMI	Copy AMI/snapshot
Load balancer	Multiple AZs	Elastic IP addresses	Server farm	Route 53 geo proximity load balancing options
Containers	Elastic Container Service (ECS)	Fargate management	Application load balancer/Auto Scaling	Regional service not multi-region
RDS deployment	Multiple AZs	Synchronous replication	Snapshots/backup EBS data volumes and transaction records	Regional service not multi-region.
Custom EC2 database	Multiple AZs and replicas	Asynchronous/Synchronous replication options	Snapshots/backup EBS volumes	Custom high-availability and failover designs across regions with Route 53 Traffic Policies
Aurora (MySQL/PostgreSQL)	Replication across 3 AZs	Multiple writers	Clustered shared storage VSAN	Global database hosted and replicated across multiple AWS regions
DynamoDB (NoSQL)	Replication across 3 AZs	Multiple writers	Continuous backup to S3	Global table replicated across multiple AWS regions
Route 53	Health checks	Failover routing	Multi-value answer routing	Geolocation/geo proximity routing
S3 bucket	Same-region replication	Built-in	Built-in	Cross-region replication

NOTE If your company uses a service to schedule EBS snapshots, make sure you have the ability to remove unneeded snapshots. If you are using S3 versioning and replication, make sure lifecycle policies have been created to purge old object versions when they are no longer required.

Disaster Recovery and Business Continuity

How do you work around service failure at AWS? You must design for failure. Each customer must use the tools and services available at AWS to create an application environment with the goal of 100% availability. When failures occur at AWS, automated processes must be designed and in place to ensure proper failover with minimum to zero data loss. You can live with computer loss, although it might be painful; data loss is unacceptable—and it does not have to happen.

It is important to understand the published AWS uptime figures. For example, Amazon Relational Database Service (RDS) has been designed to fail a mere 52 minutes per year, but this does not mean you can schedule this potential downtime. As another example, just because Route 53, AWS's DNS service, is designed for 100% uptime does not mean that Route 53 will not have issues. The published uptime figures are not guarantees; instead, they are what AWS strives for—and typically achieves.

When a cloud service fails, you are out of business during that timeframe. Failures are going to happen, and designing your AWS hosted applications for maximum uptime is the goal. You also must consider all the additional external services that allow you to connect to AWS: your telco, your ISP, and all the other moving bits. Considering all services in the equation, it's difficult—and perhaps impossible—to avoid experiencing some downtime.

There are two generally agreed upon metrics that define disaster recovery:

Key Topic

- **Recovery point objective (RPO):** ***RPO*** is the amount of data that you can acceptably lose over a defined length of time. If your RPO is defined as 5 minutes, for example, your disaster recovery needs to be designed to restore the data records to within 5 minutes of when the disaster first occurred. An RDS deployment performs automatic snapshots on a schedule, allowing you to roll back up to 35 days. Transaction logs are also backed up, allowing you to roll back to within 5 minutes of operations. So you might not lose more than 5 minutes of data, but it will generally take longer than 5 minutes to have your data restored when failure occurs. Keep in mind that with an RDS deployment across multiple availability zones, you may be back in operation without losing any data within minutes; how much time it takes depends on how long it takes for the DNS updates to reach the end user and point the user to the new primary database server.

- **Recovery time objective (RTO):** ***RTP*** is the actual time it takes to restore a business process or an application back to its defined service level. If RTO is set to 1 hour, for example, the application should be up and functional within that 1-hour timeframe.

A customer operating in the AWS cloud needs to define acceptable values for RPO and RTO based on their needs and requirements and build that in to a service-level agreement.

Backup and Restoration

On-premises DR has traditionally involved regular backups to tape and storage of the tapes off-site, in a safe location. This approach works, but recovery takes time. AWS offers several services that can assist you in designing a backup and restoration process that is much more effective than the traditional DR design. Most, if not all, third-party backup vendors have built-in native connectors to directly write to S3 as the storage target. Backups can be uploaded and written to S3 storage using a public Internet connection or using a faster private Direct Connect or VPN connection.

NOTE The AWS Backup service allows you to create a centralized backup policy for EBS volumes, RDS deployments, Amazon Elastic File System, DynamoDB tables, and AWS Storage Gateway.

Pilot Light Solution

When you design a pilot light disaster recovery configuration, your web, application, and primary database server are on premises and fully operational. Copies of the web and application servers are built on EC2 instances in the AWS cloud and are ready to go but are not turned. Your on-premises primary database server replicates updates and changes to the standby database server hosted in the AWS cloud, as shown in Figure 3-6. When planning what AWS region to use for your disaster recovery site, the compliance rules and regulations that your company follows dictate which regions can be used. In addition, you want the region/availability zones to be as close as possible to your physical corporate location.

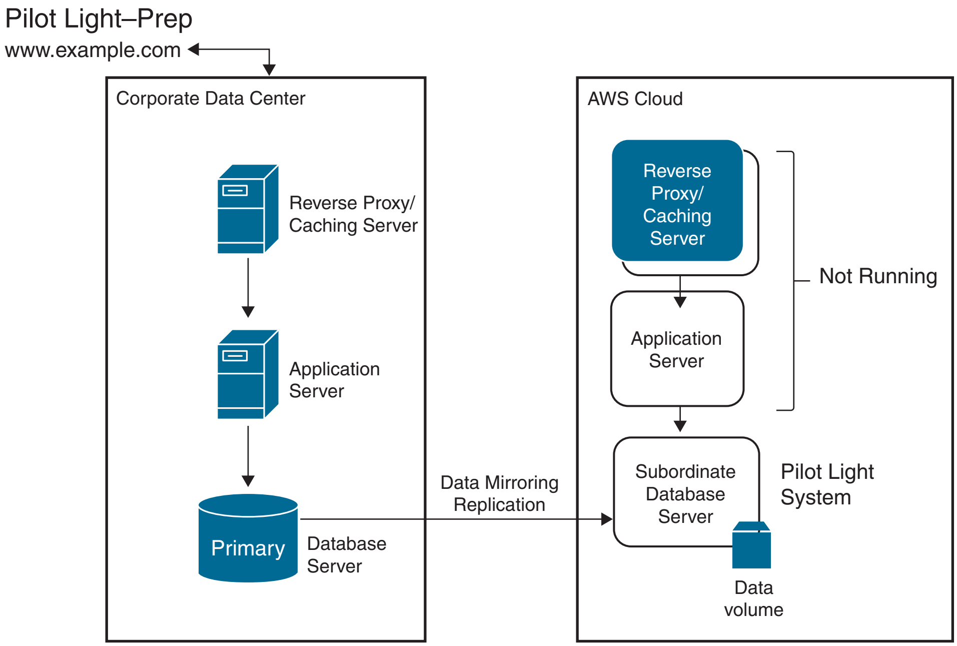


FIGURE 3-6 Pilot Light Setup

When a disaster occurs, the web and application instances and any other required infrastructure, such as a load balancer at AWS are initiated. The standby database server at AWS needs to be set as the primary database server, and the on-premise DNS services have to be configured to redirect traffic to the AWS cloud as the disaster recovery site, as shown in Figure 3-7. The RTO to execute a ***pilot light*** deployment is certainly faster than the backup and restoration scenario; there is no data loss, but there is no access to the hosted application at AWS until configuration is complete. The key to having a successful pilot light solution is to have all of your initial preconfiguration work automated with CloudFormation templates, ensuring that your infrastructure is built and ready to go as fast as possible. We will discuss CloudFormation automation later in this chapter.

Warm Standby Solution

A ***warm standby*** solution speeds up recovery time because all the components in the warm standby stack are already in active operation—hence the term *warm*—but at a smaller scale of operation. Your web, application, and database servers are all in operation, including the load balancer, as shown in Figure 3-8.

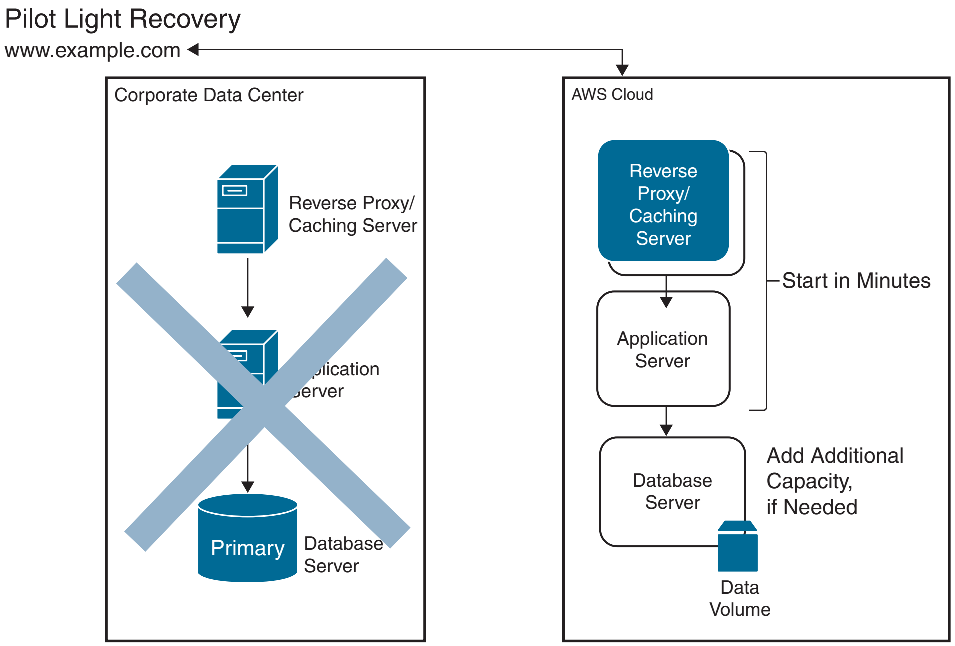


FIGURE 3-7 Pilot Light Response

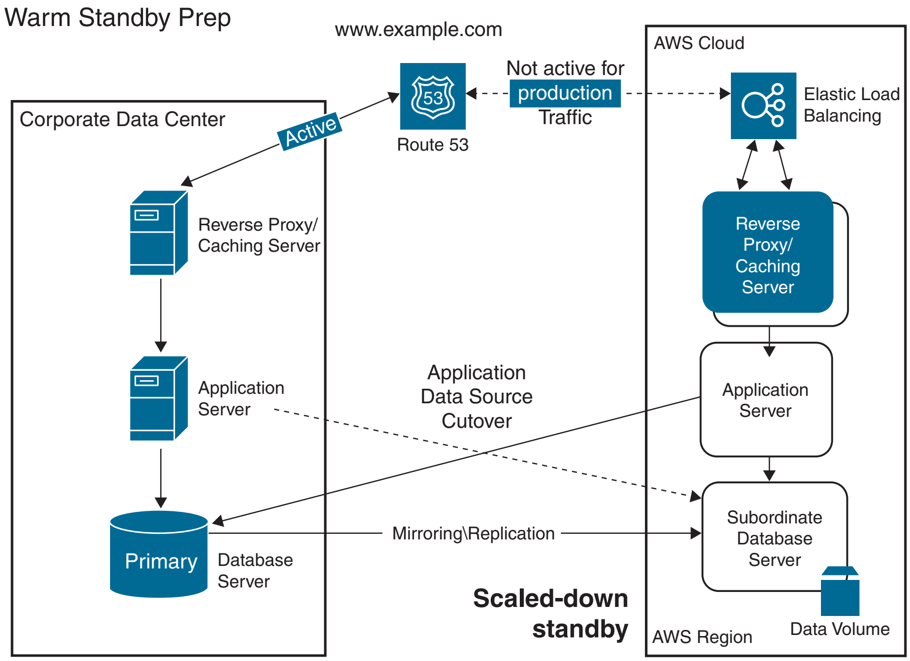


FIGURE 3-8 Warm Standby Setup

The key variable is that the warm standby application stack is not active for production traffic until disaster strikes. When a disaster occurs, you recover by increasing the capacity of your web and application servers by changing the size or number of EC2 instances and reconfiguring DNS records to reroute the traffic to the AWS site, as shown in Figure 3-9. Because all resources were already active, the recovery time is shorter than with a pilot light solution; however, a warm standby solution is more expensive than a pilot light option as more resources are running 24/7.

Key Topic

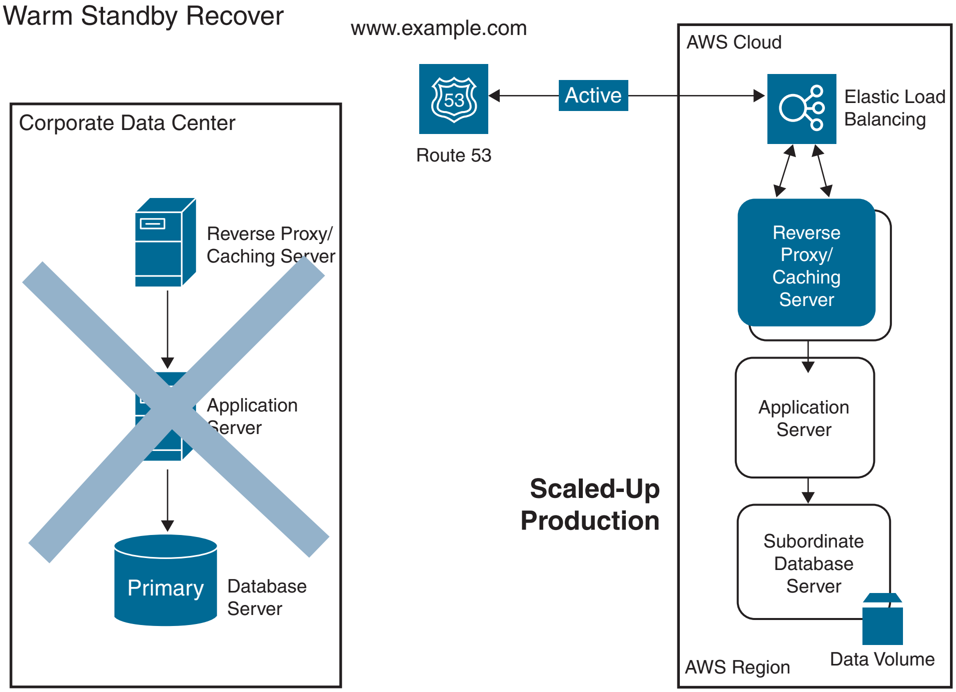


FIGURE 3-9 Warm Standby Response

An application that requires less downtime with minimal data loss could also be deployed by using a warm standby design across two AWS regions. The entire workload is deployed to both AWS regions, using a separate application stack for each region.

NOTE Keep in mind that this design must consider possible data sovereignty requirements if you are using a region hosted in the EU that is bound by the General Data Protection Regulation (GDPR).

Because data replication occurs across multiple AWS regions, the data will eventually be consistent, but the time required to replicate to both locations could be substantial. By using a read-local/write-global strategy, you could define one region as the primary for all database writes. Data would be replicated for reads to the other AWS region. If the primary database region then fails, failover to the passive site occurs. Obviously, this design has plenty of moving parts to consider and manage. This design could also take advantage of multiple availability zones within a single AWS region instead of using two separate regions.

Hot Site Solution

If you need RPO and RTO to be very low, you might want to consider deploying a ***hot site*** solution with active-active components running both on premises and in the AWS cloud. The secret sauce of the hot site is Route 53, the AWS DNS service. The database is mirrored and synchronously replicated, and the web and application servers are load balanced and in operation, as shown in Figure 3-10. The application servers are synchronized to the live data located on premises.

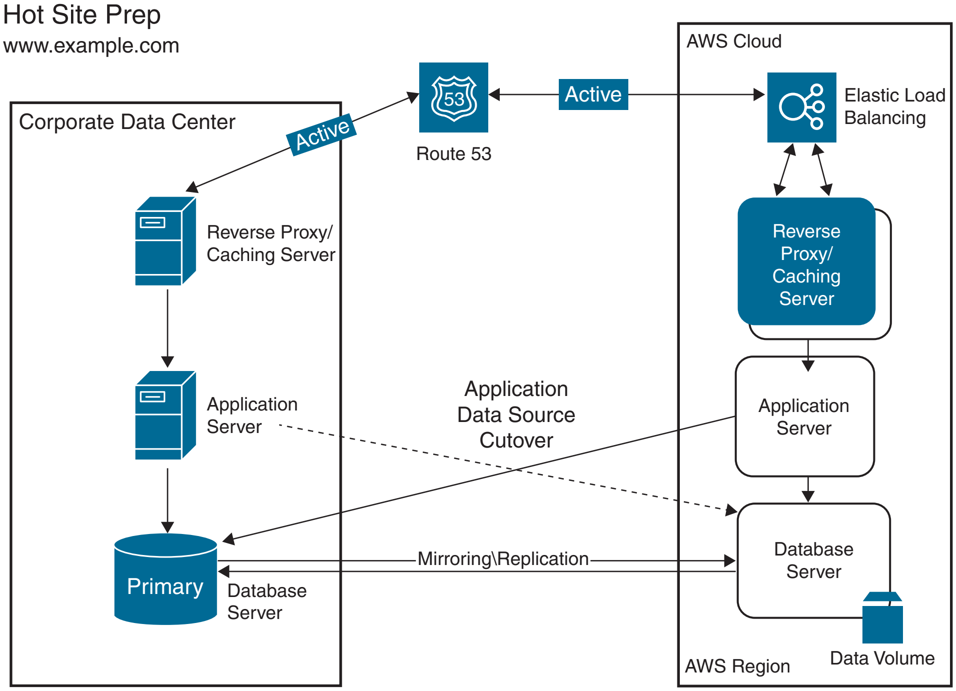


FIGURE 3-10 Hot Site Setup

Both application tiers are already in full operation; if there's an issue with one of the application stacks, traffic gets redirected automatically, as shown in Figure 3-11. With AWS, you can use Auto Scaling to scale resources to meet the capacity requirements if the on-premises resources fail. A hot site solution is architected for disaster recovery events.

Key Topic

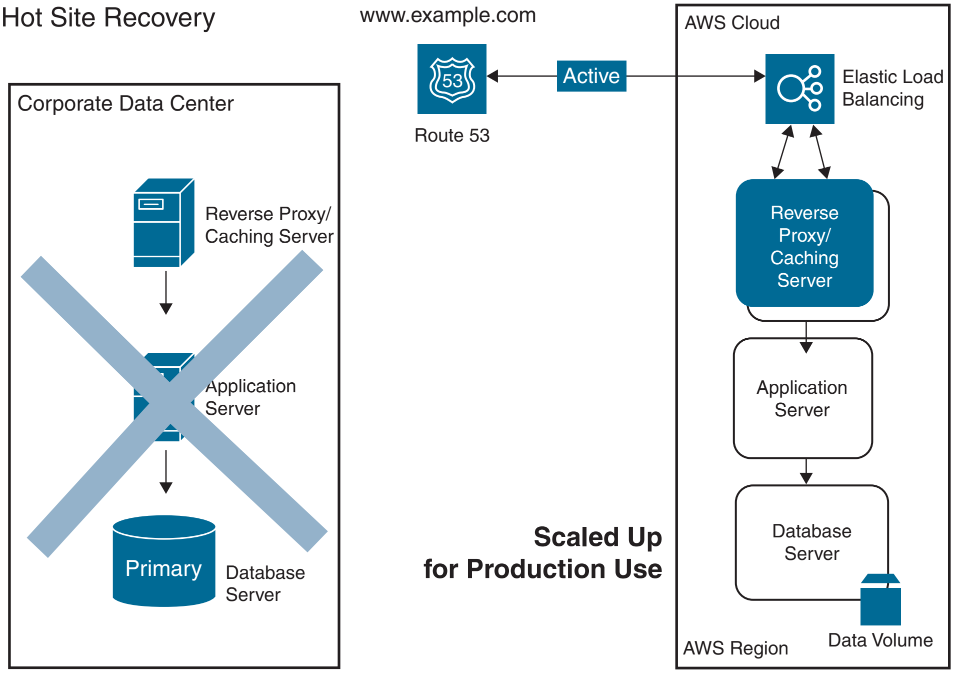


FIGURE 3-11 Hot Site Response

Multi-Region Active-Active Application Deployment

An active-active deployment of an application across multiple AWS regions adds a level of redundancy and availability, but it has an increased cost as each region hosts a complete application stack.

For the AWS Certified Solutions Architect - Associate (SAA-C02) exam, it is important to know that for an automated ***multi-region*** deployment, AWS offers Amazon Aurora, a relational database solution for maintaining the consistency of database records across the two AWS regions. Aurora, which is a relational database that is PostgreSQL and MySQL compatible, can function as a single global database operating across multiple AWS regions. An Aurora global database has one primary region and up to five read-only secondary regions. Cross-region replication latency with Aurora is typically around 1 second. Aurora allows you to create up to 16 additional database instances in each AWS region; these instances all remain up to date because Aurora storage is a shared virtual SAN clustered shared storage solution.

With Aurora, if your primary region faces a disaster, one of the secondary regions can be promoted to take over the reading and writing responsibilities, as shown in Figure 3-12. Aurora cluster recovery can be accomplished in less than 1 minute. Applications that use this type of database design would have an effective RPO of 1 second and an RTO of less than 1 minute. Web and application servers in both AWS regions are placed behind elastic load-balancing services (ELB) at each tier level and also use Auto Scaling to automatically scale each application stack, when required, based on changes in application demand. Keep in mind that Auto Scaling can function across multiple availability zones; Auto Scaling can mitigate an availability zone failure by adding the required compute resources in a single availability zone.

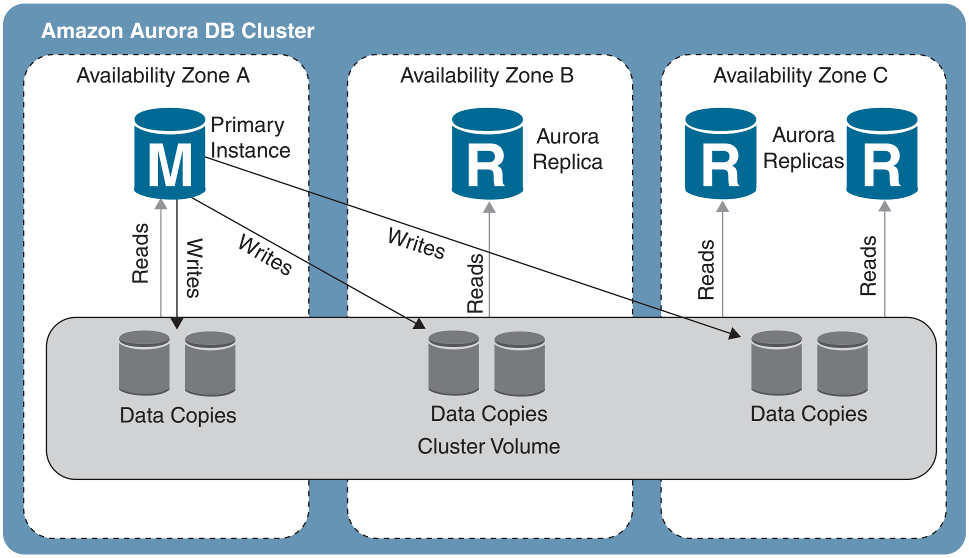


FIGURE 3-12 Aurora DB Cluster with Multiple Writes

The AWS Certified Solutions Architect - Associate (SAA-C02) exam is likely to ask you to consider best practices based on various scenarios. There are many potential solutions to consider in the real world, and Amazon wants to ensure that you know about a variety of DR solutions.

NOTE Each disaster has several attributes that you need to be aware of:

- **Duration:** The failure might be so small that no customer impact is felt.
- **Latency:** The application might be so slow to respond that the customer may believe that the application is broken.
- **Impact:** The failure may affect the entire company or certain areas of the company.

The AWS Service-Level Agreement (SLA)

Many technical people over the years have described cloud ***service-level agreements (SLAs)*** as being inadequate—especially when they compare cloud SLAs with on-premises SLAs. Imagine that you were a cloud provider with multiple data centers and thousands of customers. How would you design an SLA? You would probably tell your customers something like "We do the best job we can, but computers do fail, as we know." If you think about the difficulty of what to offer a customer when hardware and software failures occur, you will probably come up with the same solution that all cloud providers have arrived at. For example, AWS uses the following language in an SLA:

> AWS will use commercially reasonable efforts to make the AWS service in question available with a monthly uptime percentage of at least this defined percentage for each AWS region. In the event that the listed AWS service does not meet the listed service commitment, you will be eligible to receive a service credit.

With AWS, many core services have separate SLAs. Certainly, the building blocks of any application—compute, storage, CDN, and DNS services—have defined service levels (see Table 3-4). However, an SLA does not really matter as much as how you design your application to get around failures when they occur.

Table 3-4 Service-Levels at AWS

AWS Service	General Service Commitment
EC2 instances	99.99%
EBS volumes	99.99%
RDS	99.95%
S3 buckets	99.9%
Route 53	100 %
CloudFront	99.9%
Aurora	99.99%
DynamoDB	99.99%
Elastic Load Balancing	99.99%

NOTE Visit www.cloudharmony.com/status-for-aws to review the uptime status of AWS services and locations.

As you think about AWS cloud service SLAs, keep in mind that each service is going to fail, and you're not going to have any warning of these failures. This is about the only guarantee you have when hosting applications in the public cloud: The underlying cloud services are going to fail unexpectedly. AWS services are typically stable for months, but failures do happen unexpectedly.

Most of the failures that occur in the cloud are compute failures. An instance that is powering an application server, a web server, a database server, or a caching server fails. What happens to your data? Your data in the cloud is replicated, at the very least, within the AZ where your instances are running. (Ideally, your data records reside on multiple EBS volumes.) This does not mean you can't lose data in the cloud; if you never back up your data, you will probably lose it. And because customers are solely in charge of their own data, 100% data retention is certainly job one.

Automating AWS Architecture

Many systems have been put in place over the years to help successfully manage and deploy complicated software applications on complicated hardware stacks.

If you look at the AWS cloud as an operating system hosted on the Internet, the one characteristic of AWS that stands above all others is the level of integrated automation used to deploy, manage, and recover AWS services. There is not a single AWS service offered that is not heavily automated for deployment and in its overall operation. When you order a virtual private network (VPN), it's created and available in seconds. If you order an Elastic Compute Cloud (EC2) instance, either through the AWS Management Console or by using the AWS command-line interface (CLI) tools, it's created and available in minutes. Automated processes provide the just-in-time response you want when you order cloud services.

AWS services are being changed, enhanced, and updated 24/7, with features and changes appearing every day. AWS as a whole is deployed and maintained using a combination of developer agility and automated processes; it is able to move quickly and easily with a partnership of developers, system operations, project managers, network engineers, and security professionals working together from the initial design stages, through the development process, to production and continual updates.

AWS wasn't always so automated and regimented. In the early days, Amazon was a burgeoning online e-commerce bookseller. Not so very long ago, you would order a virtual machine from a cloud provider and wait several days for an email to arrive, telling you that your service was ready to go. As the Amazon e-commerce site became more popular, new problems appeared in the realm of scaling online resources to match customers' needs. Over time, Amazon developed rules for all developers, mandating that each underlying service that supported the Amazon store must be accessible through a core set of shared application programming interfaces (APIs) that were shared with all developers; each service needed to be built and maintained on a common core of compute and storage resources.

Amazon built and continues to build its hosting environment using mandated internal processes, which can be described as a mixture of the following:

- **ITIL:** ***Information Technology Infrastructure Library (ITIL)*** is a framework of best practices for delivering IT services.
- **Scrum:** ***Scrum*** is a framework in which a development team works together to manage product development.
- **Agile:** ***Agile*** is a software development cycle in which developers plan, design, develop, test, and evaluate as a team with open communications.
- **DevOps:** DevOps is a continuation of the Agile framework that features full collaboration among the development and operations teams.

Many people at AWS are working together in an effective manner to make hundreds of changes to the AWS hardware and software environment every month. In addition, all AWS services are being monitored, scaled, rebuilt, and logged through completely automated processes. Amazon avoids using manual processes, and your long-term goal should be for your company to do the same.

As your experience with AWS grows, you're going to want to start using automation to help run and manage your day-to-day AWS operations and to solve problems when they occur. There are numerous services available that don't cost anything additional to use aside from the time it takes to become competent in using them. This might sound too good to be true, but most of Amazon's automation services are indeed free to use; you are charged only for the AWS compute and storage resources that each service uses.

Automation services will always manage your resources more effectively than you can manually. At AWS, the automation of infrastructure is typically called *infrastructure as code (IAC)*. When you create resources using the AWS Management Console, in the background, AWS uses automated processes running scripts to finish the creation and management of resources.

Regardless of how you define your own deployment or development process, there are a number of powerful tools in the AWS toolbox that can help you automate procedures:

- **CloudFormation:** For stack-based architecture deployments
- **Service Catalog:** To help secure CloudFormation templates
- **Elastic Beanstalk:** For deploying applications and infrastructure together

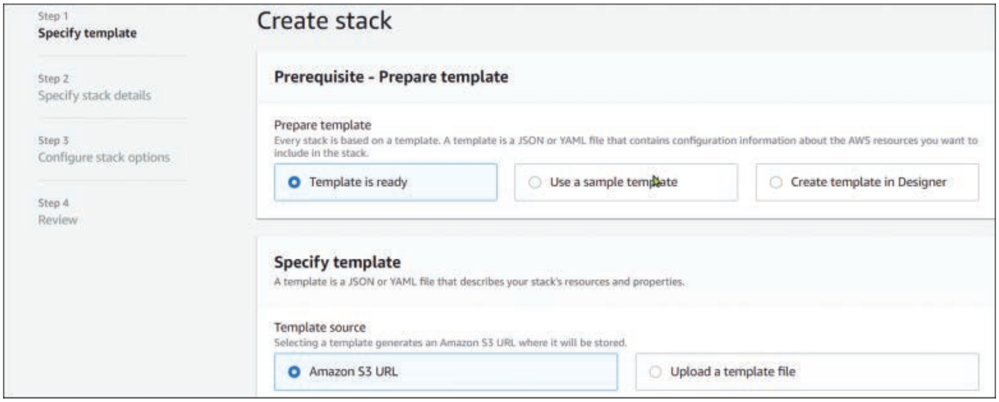
Automating Infrastructure with CloudFormation

The second and third times you deploy EC2 instances using the AWS Management Console, you will probably not perform the steps the same way you did the first time. Even if you do manage to complete a manual task with the same steps, by the tenth installation your needs will have changed or better options will have become available. A manual process rarely stays the same over time. To make changes easier, you can automate even the simplest manual processes at AWS.

If you peek under the hood at any management service running at AWS, you'll find the process command set driven by ***JavaScript Object Notation (JSON)*** scripts. At the GUI level, you use the Management Console to fill in the blanks; when you click Create, JSON scripts are executed in the background to carry out your requests. AWS's extensive use of JSON is similar to Microsoft Azure's extensive use of PowerShell; at AWS, JSON scripts are used internally for many tasks and processes; creating security policy with Identity and Access Management (IAM) and working with CloudFormation are two examples that you will commonly come across.

If you use Windows EC2 instances at AWS, you can also use PowerShell scripting. Both Microsoft Azure and AWS heavily rely on automation tools for everyday deployments. In the background, Azure relies heavily on PowerShell scripting and automation.

CloudFormation is an AWS-hosted orchestration engine that works with JSON templates to deploy AWS resources on demand or on predefined triggers (see Figure 3-13). AWS uses CloudFormation extensively, and so can you. More than 300,000 AWS customers use CloudFormation to manage deployment of just about everything, including all infrastructure stack deployments. A CloudFormation template declares the infrastructure stack to be created, and the CloudFormation engine automatically deploys and links the needed resources. You can add additional control variables to each CloudFormation template to manage and control the precise order of the installation of resources.



Step 1
Specify template

Step 2
Specify stack details

Step 3
Configure stack options

Step 4
Review

Create stack

Prerequisite - Prepare template

Prepare template
Every stack is based on a template. A template is a JSON or YAML file that contains configuration information about the AWS resources you want to include in the stack.

● Template is ready ○ Use a sample template ○ Create template in Designer

Specify template
A template is a JSON or YAML file that describes your stack's resources and properties.

Template source
Selecting a template generates an Amazon S3 URL where it will be stored.

● Amazon S3 URL ○ Upload a template file

FIGURE 3-13 The CloudFormation Console

Consider this comparison of the manual AWS deployment process and the automated process that starts with CloudFormation:

- **Time spent:** Over time, running manual processes at AWS becomes a big waste of time for the human carrying out the process. In the past, maintaining manual processes such as building computer systems and stacks provided job security; these days, it's just not a prudent way to deploy production resources. For one thing, there are many steps in a manual process—which means there are many places the process can go wrong. Every CloudFormation deployment does take time, but it takes much less time than a manual process because each step in a CloudFormation script is essential. There are no wasted steps, and all steps are in the proper order. Over time, executing an automated process to build EC2 instances will save you hours or even weeks of time; the CloudFormation process runs in the background, allowing you to do something else. CloudFormation can also perform updates and deletions for existing AWS resources.

- **Security issues:** Humans make mistakes, and manual changes can end up being huge security mistakes due to a lack of oversight. CloudFormation templates can be secured and controlled for usage by specific IAM users and groups. Templates also carry out the same steps every time they are executed, which prevents the fat-finger mistakes that humans make. The AWS Service Catalog service integrates with CloudFormation to mandate which users or accounts can use CloudFormation templates to build infrastructure stacks.

- **Documentation:** It is difficult to document manual processes if they constantly change, and it can be difficult to find time to create documentation. CloudFormation templates are readable, and when you get used to the format, they are actually self-documenting. Again, there are no wasted steps in a CloudFormation script; what is described is exactly what is deployed. If mistakes are found in a CloudFormation script during deployment, all changes that have been carried out are reversed.

- **Repeatability:** If you're lucky, you can repeat your manual steps the same way every time. However, trying to do so is a waste of valuable time in the long run. With a CloudFormation script, you can deploy and redeploy the listed AWS resources in multiple environments, such as separate development, staging, and production environments. Every time a CloudFormation template is executed, it repeats the same steps.

- **Cost savings:** CloudFormation automation carries out stack deployments and updates much faster than manual processes ever could. In addition, CloudFormation automation can be locked down using a companion service called AWS Systems Manager, discussed later in this chapter, to ensure that only specific IAM users and groups can access and execute specific CloudFormation deployment tasks.

CloudFormation Components

CloudFormation works with templates, stacks, and change sets. A *CloudFormation template* is an AWS resource blueprint that can create a complete application stack or a single stack component, such as a VPC network complete with multiple subnets, Internet gateways, and NAT services all automatically deployed and configured. You can create a template type called a *change set* to help visualize how proposed changes will affect AWS resources deployed by a CloudFormation template.

CloudFormation Templates

Key Topic

Each CloudFormation template is a text file that follows either JSON or YAML formatting standards. CloudFormation responds to files saved with JSON, YAML, and .txt extensions. Each template can deploy or update multiple AWS resources or a single resource, such as a VPC or an EC2 instance. Example 3-1 shows a CloudFormation template in JSON format, and Example 3-2 displays the same information in YAML format. It's really a matter of personal preference which format you use. When creating CloudFormation templates, you might find YAML easier to read, which could be helpful in the long term. YAML seems more self-documenting because it's easier to read.

Example 3-1 CloudFormation Template in JSON Format

```
{
    "AWSTemplateFormatVersion" : "2010-09-09",
    "Description" : "EC2 instance",
    "Resources" : {
    "EC2Instance" : {
    "Type" : "AWS::EC2::Instance",
    "Properties" : {
    "ImageId" : "ami-0ff8a91497e77f667",
    "InstanceType" : "t1.micro"
    }
    }
    }
}
```

Example 3-2 CloudFormation Template in YAML Format

```
AWSTemplateFormatVersion: '2010-09-09'
Description: EC2 instance
Resources:
EC2Instance:
Type: AWS::EC2::Instance
Properties:
ImageId: ami-0ff8a91497e77f667
```

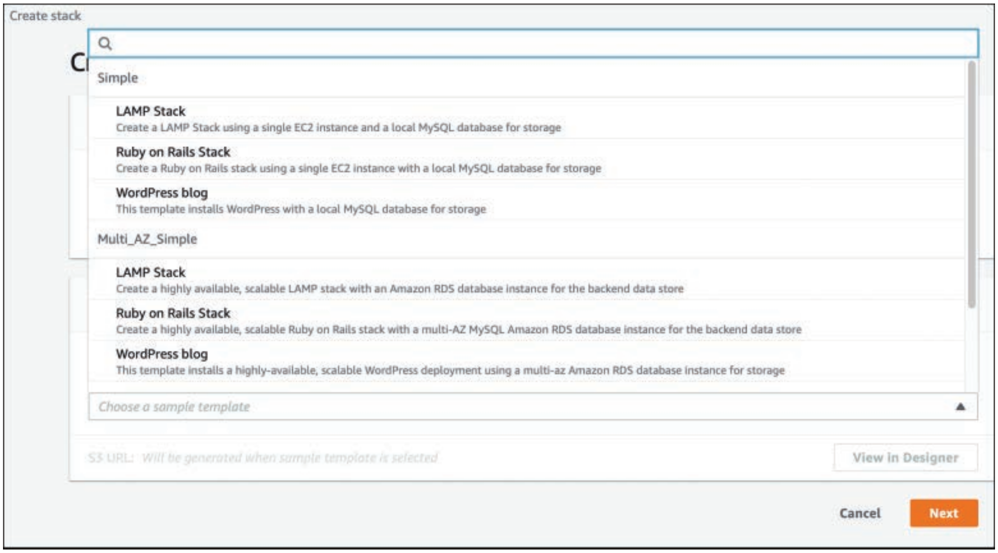
CloudFormation templates can have multiple sections, as shown in Example 3-3. However, the only mandatory section is Resources. As with any other template or script, the better the internal documentation, the more usable a CloudFormation template is—for the author as well as other individuals. It is highly recommended to use the Metadata section for comments to ensure that users understand the script.

Example 3-3 Valid Sections in a CloudFormation Template

```
"AWSTemplateFormatVersion": "version date",
"AWSTemplateFormatVersion": "2010-09-09"
<TemplateFormatVersion: Defines the current CF template version>

"Description": "Here are the additional details about this template
and what it does",
<Description: Describes the template: must always follow the version
section>

"Metadata": {
   "Metadata" : {
   "Instances" :  {"Description : "Details about the instances"},
   "Databases" : {"Description : "Details about the databases"}
  }
},
<Metadata: Additional information about the resources being deployed
by the template>

   "Parameters" : {
   "InstanceTypeParameter" : {
        "Type" : "String" ,
        "Default" : "t2.medium",
        "AllowedValues" : ["t2.medium", "m5.large", "m5.xlarge"],
        "Description" : "Enter t2.medium, m.5large, or m5.xlarge.
        Default is t2.medium."
  }
}
<Parameters: Defines the AWS resource values allowed to be selected
and used by your template>

"Mappings" : {
     "RegionMap" : [
         "us-east-1            : { "HVM64 : "ami-0bb8a91508f77f868"},
         "us-west-1           : { "HVM64 : "ami-0cdb828fd58c52239"},
         "eu-west-1           : { "HVM64 : "ami-078bb4163c506cd88"},
         "us-southeast-1     : { "HVM64 : "ami-09999b978cc4dfc10"},
         "us-northeast-1     : { "HVM64 : "ami-06fd42961cd9f0d75"}
  }
}
<Mappings: Defines conditional parameters defined by a "key"; in this
example, the AWS region and a set of AMI values to be used>

  "Conditions": {
  "CreateTestResources": {"Fn::Equals" : [{"Ref" : "EnvType"}, "test"]}
},
<Conditions: Defines dependencies between resources, such as the
order when resources are created, or where resources are created.
For example, "test" deploys the stack in the test environment>
```

CloudFormation Stacks

AWS has many sample CloudFormation templates that you can download from the online CloudFormation documentation, as shown in Figure 3-14, and deploy in many AWS regions. A CloudFormation stack can be as simple as a single VPC or as complex as a complete three-tier application stack, complete with the required network infrastructure and associated services.



Simple

LAMP Stack
Create a LAMP Stack using a single EC2 instance and a local MySQL database for storage

Ruby on Rails Stack
Create a Ruby on Rails stack using a single EC2 instance with a local MySQL database for storage

WordPress blog
This template installs WordPress with a local MySQL database for storage

Multi_AZ_Simple

LAMP Stack
Create a highly available, scalable LAMP stack with an Amazon RDS database instance for the backend data store

Ruby on Rails Stack
Create a highly available, scalable Ruby on Rails stack with a multi-AZ MySQL Amazon RDS database instance for the backend data store

WordPress blog
This template installs a highly-available, scalable WordPress deployment using a multi-az Amazon RDS database instance for storage

Choose a sample template

S3 URL: Will be generated when sample template is selected

FIGURE 3-14 AWS Sample Stacks and CloudFormation Templates

CloudFormation can be useful for deploying infrastructure at AWS, including in the following areas:

- **Network:** You can define a baseline template for developers to ensure that their VPC network deployment matches company policy.
- **Frontend infrastructure:** You can deploy Internet gateways, associated route table entries, or load balancers into existing or new AWS network infrastructure.
- **Backend infrastructure:** You can create database infrastructure, including primary and alternate database instances, subnet groups, and associated security groups.
- **Two-tier application:** Using a two-tier CloudFormation script allows you to handle failures or disasters by enabling you to rebuild a complete application stack with required network and infrastructure components or to launch the application stack in another AWS region.

- **Windows Server Active Directory:** You can deploy Active Directory on Windows Server 2008 R2 instances in a VPC.
- **Demo applications:** You can define an application stack for demonstrations, thus allowing the sales team or end users to quickly create the entire environment.
- **AWS managed services:** You can use CloudFormation templates to automate the setup of any AWS managed services. For example, you can enable and set up AWS Config or Inspector by using a CloudFormation template.

NOTE There are many standardized CloudFormation templates, called AWS Quick Starts, available from AWS (see https://aws.amazon.com/quickstart/). AWS solution architects and trusted partners have built these templates to help you deploy complete solutions on AWS. These templates are designed following the current AWS best practices for security and high availability.

Creating an EC2 Instance

Example 3-4 provides a simple example that shows how to create an EC2 instance using a CloudFormation template. The template parameters are easily readable from top to bottom. Under Properties, the EC2 instance ID, subnet ID, and EC2 instance type must all be already present in the AWS region where the template is executed; if they are not, the deployment will fail. If there are issues in the CloudFormation script during deployment, CloudFormation rolls back and removes any infrastructure that the template created. The Ref statement is used in this template to attach the elastic IP (EIP) address to the defined EC2 instance that was deployed and referenced under the resources listed as EC2 Machine.

NOTE I highly recommend looking at the AWS Quick Starts website https://aws.amazon.com/quickstart/ to see how powerful CloudFormation can be.

Example 3-4 CloudFormation Template for Creating an EC2 Instance

```
AWSTemplateFormatVersion: 2010-09-09
Description: EC2 Instance Template
"Resources": {
 "EC2Machine": {
 "Type": "AWS::EC2::Instance",
 "Properties": {
   "ImageId": "i-0ff407a7042afb0f0",
   "NetworkInterfaces": [{
   "DeviceIndex": "0",
   "DeleteOnTermination": "true",
   "SubnetId": "subnet-7c6dd651"
   }]
   "InstanceType": "t2.small"
   }
  }
},
"EIP": {
  "Type": "AWS::EC2::EIP",
  "Properties": {
  "Domain": "VPC"
  }
},
"VpcIPAssoc": {
"Type": "AWS::EC2::EIPAssociation",
  "Properties": {
  "InsanceId":  {
  "Ref": "EC2Machine"
  },
  "AllocationId": {
  "Fn::GetAtt": ["EIP",
  "AllocationId"]
  }
  }
}
```

Updating with Change Sets

Change sets allow you to preview how your existing AWS resources will be modified when a deployed CloudFormation resource stack needs to be updated (see Figure 3-15). You select an original CloudFormation template to edit and input the desired set of changes. CloudFormation then analyzes your requested changes against the existing CloudFormation stack and produces a change set that you can review and approve or cancel.

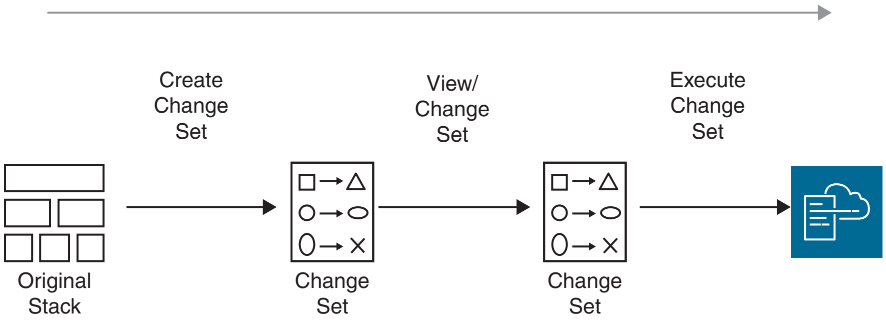


FIGURE 3-15 Using Change Sets with CloudFormation

You can create multiple change sets for various comparison purposes. Once a change set is created, reviewed, and approved, CloudFormation updates your current resource stack.

CloudFormation Stack Sets

A stack set allows you to create a single CloudFormation template to deploy, update, or delete AWS infrastructure across multiple AWS regions and AWS accounts. When a CloudFormation template will be deploying infrastructure across multiple accounts, as shown in Figure 3-16, and AWS regions, you must ensure that the AWS resources that the template references are available in each AWS account or region. For example, EC2 instances, EBS volumes, and key pairs are always created in a specific region. These region-specific resources must be copied to each AWS region where the CloudFormation template is executed. It is important to review global resources such as IAM roles and S3 buckets that are being created by the CloudFormation template to make sure there are no naming conflicts during creation, as global resources must be unique across all AWS regions.

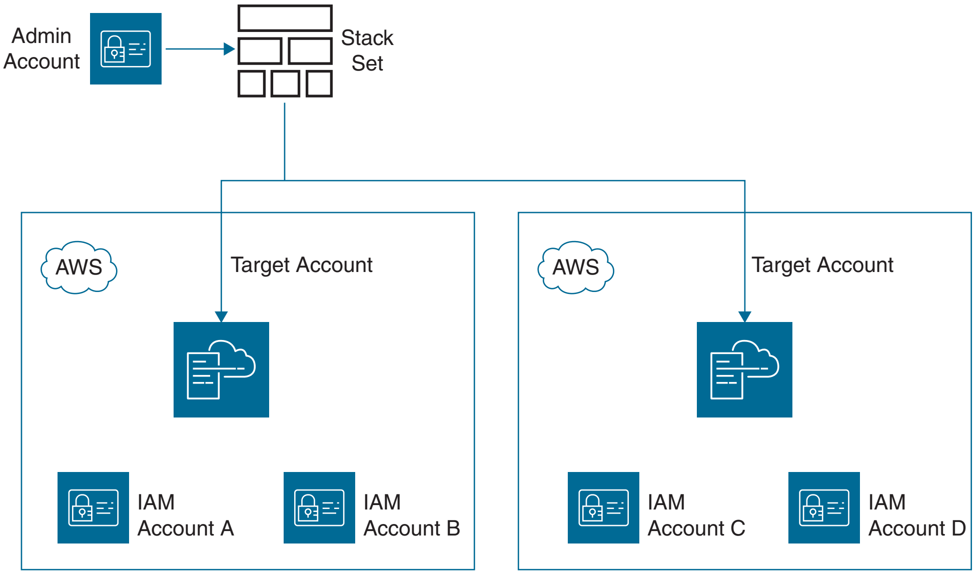


FIGURE 3-16 A Stack Set with Two AWS Target Accounts

Once a stack set is updated, all instances of the stack that were created are updated as well. For example, if you have 10 AWS accounts across 3 AWS regions, 30 stack instances are updated when the primary stack set is executed. If a stack set is deleted, all corresponding stack sets are also deleted.

A stack set is first created in a single AWS account. Before additional stack instances can be created from the primary stack set, trust relationships using IAM roles must be created between the initial AWS administrator account and the desired target accounts.

For testing purposes, one example available in the AWS CloudFormation console is a sample stack set that allows you to enable AWS Config across selected AWS regions or accounts. Keep in mind that AWS Config allows you to control AWS account compliance by defining rules that monitor specific AWS resources to ensure that the desired level of compliance has been followed.

Key Topic

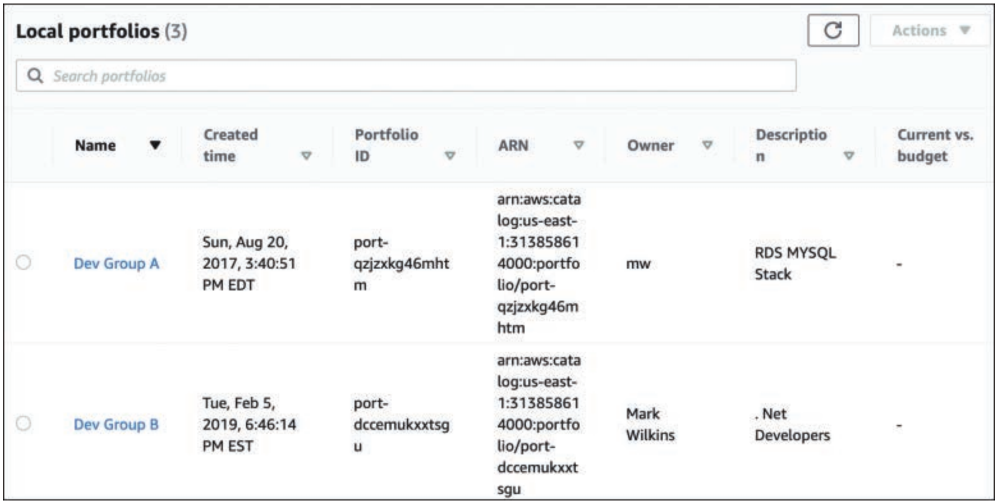
Third-Party Solutions

There are a number of third-party solutions, such as Chef, Puppet, Ansible, and TerraForm, for performing automated deployments of compute infrastructure. CloudFormation does not replace these third-party products but can be a useful tool for building automated solutions for your AWS infrastructure if you don't use one of the third-party orchestration tools. AWS has a managed service called OpsWorks that comes in three flavors and might be useful to your deployments at AWS if your company currently uses Chef or Puppet:

- **AWS OpsWorks Stacks:** This flavor allows you to manage applications and services that are hosted at AWS and on premises by running Chef recipes, bash, or PowerShell scripts.
- **AWS OpsWorks for Chef Automate:** This flavor allows you to build a fully managed Chef Automate server that supports the latest versions of Chef server and Chef Automate, any community-based tools or cookbooks, and native Chef tools.
- **OpsWorks for Puppet Enterprise:** This is a fully managed Puppet Enterprise environment that patches, updates, and backs up your existing Puppet environment and allows you to manage and administrate both Linux and Windows server nodes hosted on EC2 instances and on premises.

AWS Service Catalog

Using a CloudFormation template provides great power for creating, modifying, and updating AWS infrastructure. Creating AWS infrastructure always costs money. Say that you would like to control who gets to deploy specific CloudFormation templates. You can use Service Catalog to manage the distribution of CloudFormation templates as a product list to an AWS account ID, an AWS Organizations account, or an organizational unit contained within an AWS organization. Service Catalog is composed of portfolios, as shown in Figure 3-17, each of which is a collection of one or more products.



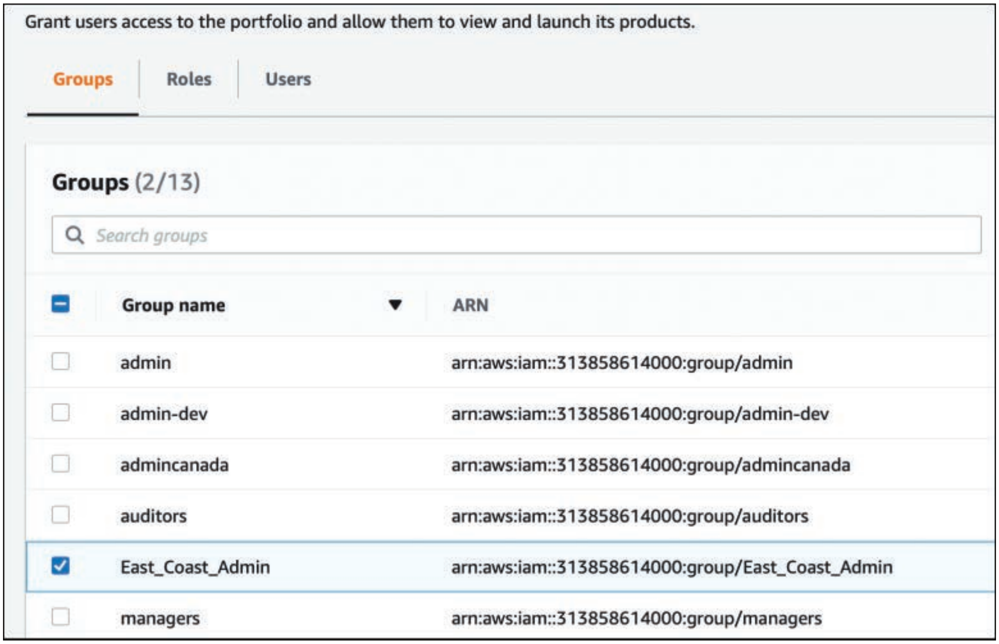
	Name ▼	Created time ▽	Portfolio ID ▽	ARN ▽	Owner ▽	Description ▽	Current vs. budget
○	Dev Group A	Sun, Aug 20, 2017, 3:40:51 PM EDT	port-qzjzxkg46mhtm	arn:aws:catalog:us-east-1:313858614000:portfolio/port-qzjzxkg46mhtm	mw	RDS MYSQL Stack	-
○	Dev Group B	Tue, Feb 5, 2019, 6:46:14 PM EST	port-dccemukxxtsgu	arn:aws:catalog:us-east-1:313858614000:portfolio/port-dccemukxxtsgu	Mark Wilkins	. Net Developers	-

FIGURE 3-17 Portfolios in Service Catalog

When an approved product is selected, Service Catalog delivers a confirmation template to CloudFormation, which then executes the template and creates the product. Third-party products hosted in the AWS Marketplace are also supported by Service Catalog, and software appliances are bundled with a CloudFormation template.

Each IAM user in an AWS account can be granted access to a Server Catalog portfolio of multiple approved products. Because products are built using common confirmation templates, any AWS infrastructure components, including EC2 instances and databases hosted privately in a VPC, can be deployed. In addition, VPC endpoints using AWS PrivateLink allow access to the AWS Service Catalog service.

When you're creating Service Catalog products, you can use constraints with IAM roles to limit the level of administrative access to the resources contained in the stack being deployed by the product itself. You can also assign server actions for rebooting, starting, or stopping deployed EC2 instances, as shown in Figure 3-18.



Grant users access to the portfolio and allow them to view and launch its products.

Groups (2/13)

⊟	Group name ▼	ARN
☐	admin	arn:aws:iam::313858614000:group/admin
☐	admin-dev	arn:aws:iam::313858614000:group/admin-dev
☐	admincanada	arn:aws:iam::313858614000:group/admincanada
☐	auditors	arn:aws:iam::313858614000:group/auditors
☑	East_Coast_Admin	arn:aws:iam::313858614000:group/East_Coast_Admin
☐	managers	arn:aws:iam::313858614000:group/managers

FIGURE 3-18 IAM Group Constraints Controlled by Service Catalog

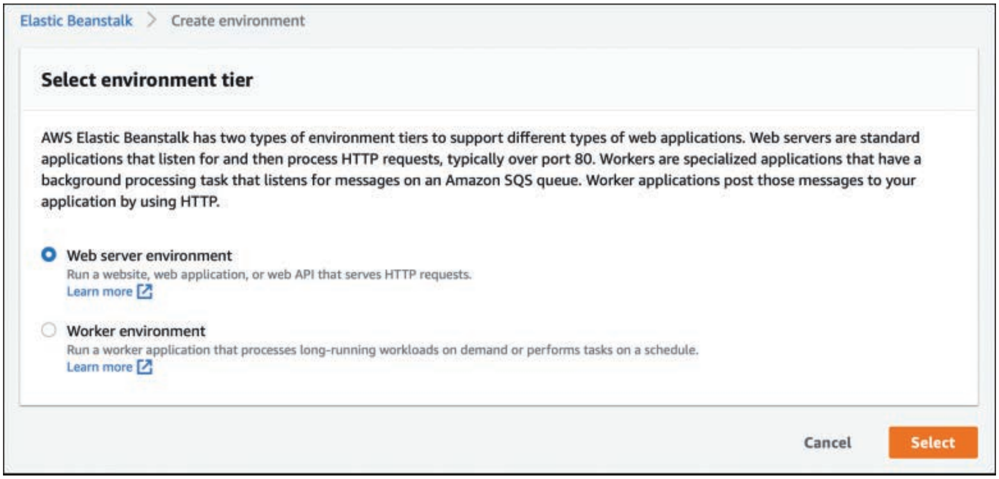
In addition, you can add rules that control any parameter values that the end user enters. For example, you could mandate that specific subnets must be used for a stack deployment. You can also define rules that allow you to control which AWS account and region a product can launch.

If you list deployed products by version number, you can allow end users to select the latest versions of products so they can update older versions of currently deployed products. In this way, you can use CloudFormation and Service Catalog together to create a self-serve portal for developers consisting of portfolios and products.

Elastic Beanstalk

When moving to the AWS cloud, developers typically have little time and budget but must develop a web application or migrate an existing web app into the AWS cloud while adhering to the company's compliance standards. The web application needs to be reliable, able to scale, and easy to update. In such situations, Elastic Beanstalk can be of some help.

Elastic Beanstalk, which has been around since 2011, was launched as a platform as a service (PaaS) offering from AWS to help enable developers to easily deploy web applications hosted on AWS Linux and Windows EC2 instances in the AWS cloud. As briefly mentioned earlier in this chapter, Elastic Beanstalk automates both application deployment, as shown in Figure 3-19, and the required infrastructure components, including single and multiple EC2 instances behind an elastic load balancer hosted in an Auto Scaling group. Monitoring of your Elastic Beanstalk environment is carried out with CloudWatch metrics for monitoring the health of your application. Elastic Beanstalk also integrates with AWS X-Ray, which can help you monitor and debug the internals of your hosted application.



Elastic Beanstalk > Create environment

Select environment tier

AWS Elastic Beanstalk has two types of environment tiers to support different types of web applications. Web servers are standard applications that listen for and then process HTTP requests, typically over port 80. Workers are specialized applications that have a background processing task that listens for messages on an Amazon SQS queue. Worker applications post those messages to your application by using HTTP.

● Web server environment
Run a website, web application, or web API that serves HTTP requests.
Learn more

○ Worker environment
Run a worker application that processes long-running workloads on demand or performs tasks on a schedule.
Learn more

FIGURE 3-19 Elastic Beanstalk Creating Infrastructure and Installing an Application

Elastic Beanstalk supports a number of development platforms, including Java (Apache HTTP or Tomcat) for PHP, Node.js (Nginx or Apache HTTP), Python (Apache HTTP), Ruby (Passenger), .NET (IIS), and the Go language. Elastic Beanstalk allows you to deploy different runtime environments across multiple technology stacks that can all be running AWS at the same time; the technology stacks can be EC2 instances or Docker containers.

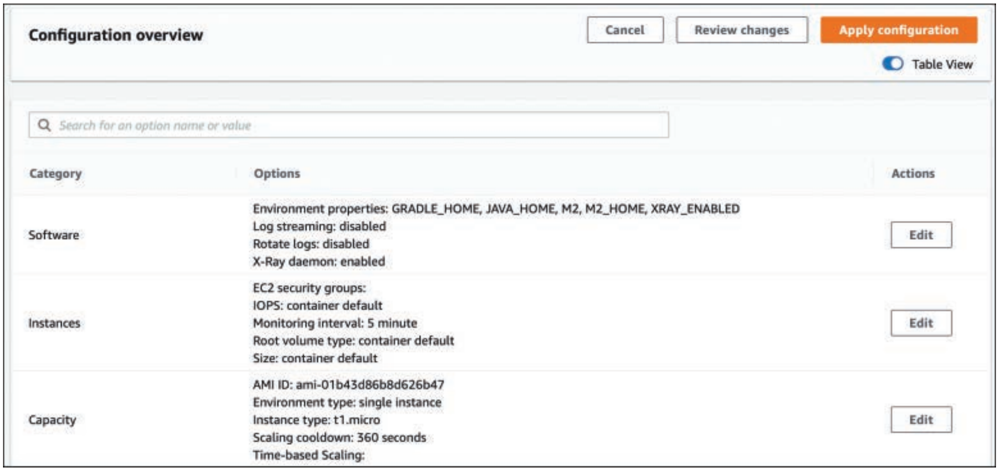
Developers can use Elastic Beanstalk to quickly deploy and test applications on a predefined infrastructure stack. If an application does not pass the test, the infrastructure can be quickly discarded at little cost. Keep in mind that Elastic Beanstalk is not a development environment (like Visual Studio). An application must be written and ready to go before Elastic Beanstalk is useful. After an application has been written, debugged, and approved from your Visual Studio (or Eclipse) development environment combined with the associated AWS Toolkit, you need to upload your code. Then you can create and upload a configuration file that details the infrastructure that needs to be built. Elastic Beanstalk completes the deployment process for the infrastructure and the application. The original goal for Elastic Beanstalk was to greatly reduce the timeframe for hardware procurement for applications (which in some cases took weeks or months).

Elastic Beanstalk is useful in organizations that are working with a DevOps mentality, where the developer is charged with assuming some operational duties. Elastic Beanstalk can help developers automate tasks and procedures that were previously carried out by administrators and operations folks when an application was hosted in an on-premises data center. Elastic Beanstalk carries out the following tasks automatically:

Key Topic

- Provisions and configures EC2 instances, containers, and security groups using a CloudFormation template
- Configures your RDS database server environment
- Configures your load balancer and Auto Scaling
- Stores the application server's source code, associated logs, and artifacts in an S3 bucket
- Enables CloudWatch alarms that monitor the load of your application, triggering Auto Scaling for your infrastructure as necessary
- Routes access from the hosted application to a custom domain
- Performs blue/green deployments and immutable updates

Elastic Beanstalk is free of charge to use; you are charged only for the resources used for the deployment and hosting of your applications. The AWS resources that you use are provisioned within your AWS account, and you have full control of these resources. In contrast, with other PaaS solutions, the provider blocks access to the infrastructure resources. At any time, you can go into the Elastic Beanstalk configuration of your application and make changes, as shown in Figure 3-20. Although Beanstalk functions like a PaaS service, you can tune and change the infrastructure resources as desired.



Category	Options	Actions
Software	Environment properties: GRADLE_HOME, JAVA_HOME, M2, M2_HOME, XRAY_ENABLED Log streaming: disabled Rotate logs: disabled X-Ray daemon: enabled	Edit
Instances	EC2 security groups: IOPS: container default Monitoring interval: 5 minute Root volume type: container default Size: container default	Edit
Capacity	AMI ID: ami-01b43d86b8d626b47 Environment type: single instance Instance type: t1.micro Scaling cooldown: 360 seconds Time-based Scaling:	Edit

FIGURE 3-20 Modifying the Capacity of the Elastic Beanstalk Application Infrastructure

Applications supported by Elastic Beanstalk include simple HTTPS web applications and applications with worker nodes that can be subscribed to Amazon Simple Queue Service (SQS) queues to carry out more complex, longer-running processes.

After an application has been deployed by Elastic Beanstalk, you can have AWS automatically update the selected application platform environment by enabling managed platform updates, which can be deployed during a defined maintenance window. These updates include minor platform version updates and security patching but not major platform updates to the web services being used; major updates must be initiated manually.

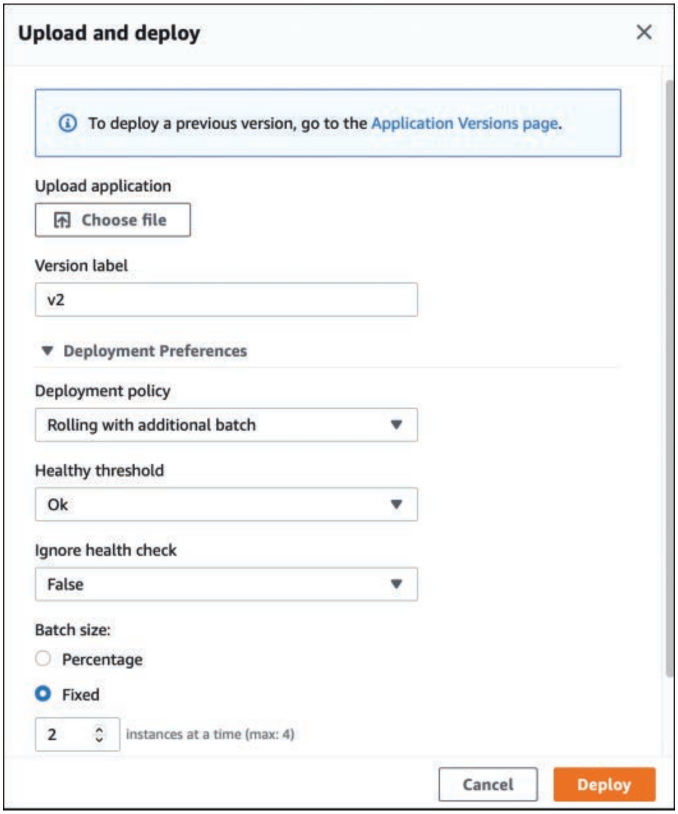
Database support for Elastic Beanstalk includes any application that can be installed on an EC2 instance, RDS database options, and DynamoDB. A database can be provisioned by Elastic Beanstalk during launch or can be exposed to the application through the use of environmental variables. You can also choose to deploy instances that are hosting your applications in multiple AZs and control your application HTTPS security and authentication by deploying Application Load Balancer.

Updating Elastic Beanstalk Applications

Key Topic

You can deploy new versions of an application to your Elastic Beanstalk environment in several ways, depending on the complexity of the application. During updates, Elastic Beanstalk archives the old application version in an S3 bucket. The methods available for updating Elastic Beanstalk applications include the following:

- **All at once:** The new application version is deployed to all EC2 instances simultaneously. Your application is unavailable while the deployment process is under way. If you want to keep an older version of your application functioning until the new version is deployed, you should choose the immutable method or the blue/green update method.
- **Rolling:** The application is deployed in batches to a select number of EC2 instances defined in each batch configuration, as shown in Figure 3-21. As the batches of EC2 instances are being updated, they are detached from the load balancer queue. When the update is finished and passes load-balancing health checks, the batch is added back to the load-balancing queue once again. The first updated batch of EC2 instances must be healthy before the next batch of EC2 instances is updated.



Upload and deploy

ⓘ To deploy a previous version, go to the Application Versions page.

Upload application

Version label: v2

▼ Deployment Preferences

Deployment policy: Rolling with additional batch

Healthy threshold: Ok

Ignore health check: False

Batch size:

○ Percentage

● Fixed

2 instances at a time (max: 4)

FIGURE 3-21 Apply Rolling Updates to an Elastic Beanstalk Application

- **Immutable:** The application update is only installed on new EC2 instances contained in a second Auto Scaling group launched in your environment. Only after the new environment passes health checks is the old application version removed. The new application servers are made available all at once. Because new EC2 instances and auto scaling groups are being deployed, the immutable update process takes longer.
- **Blue/green:** The new version of the application is deployed to a separate environment. When the new environment is healthy, the CNAMEs of the two environments are swapped so that traffic is redirected to the new application version. In this scenario, if a production database is to be used in the application design, to maintain connectivity the database must be installed separately from the Elastic Beanstalk deployment. Externally installed databases remain operational and are not removed when the new Elastic Beanstalk application version is installed and swapped on the application's EC2 instances.
- **Traffic-splitting deployments:** Canary testing can also be included as part of your application deployment. Elastic Beanstalk can launch a set of new instances with a new version of the application; however, only a specific percentage of incoming traffic will initially be forwarded to the new application instances for a defined length of time. If the new application instances remain healthy during the evaluation period, Elastic Beanstalk then forwards traffic to the new instances and terminates the old instances. If the new instances don't pass their health checks, all traffic is moved back to the current instances, and the new instances that were under evaluation are terminated, leaving the existing instances online with no service interruption.

Deployment Methodologies

Developers getting ready to create their first application in the cloud can look to a number of rules that are generally accepted for successfully creating applications that run exclusively in the public cloud.

Several years ago, Heroku cofounder Adam Wiggins released a suggested blueprint for creating native software as a service (SaaS) application hosted in the public cloud called the Twelve-Factor App Methodology. These guidelines can be viewed as a set of best practices to consider using when deploying applications in the cloud. Of course, depending on your deployment methods, you may quibble with some of the rules—and that's okay. There are many methodologies available to deploy applications. There are also many complementary management services hosted at AWS that greatly speed up the development process, regardless of the model used.

The development and operational model that you choose to embrace will follow one of these development and deployment paths.

NOTE Heroku is a managed platform as a service (PaaS) provider that Salesforce owns. Incidentally, Heroku is hosted at AWS. The software engineers at Heroku were attempting to provide guidance for applications that were going to be created in the public cloud based on their real-world experience.

- **Waterfall:** In this model, deployment is broken down into phases, including proper analysis, system design, implementation and testing, deployment, and ongoing maintenance. In the Waterfall model, each of these phases has to be completed before the next phase can begin. If your time line is short, and all of the technologies to be used are fully understood, then perhaps this model can still work in the cloud. However, cloud providers are introducing services that free you from having to know how the service works in detail so that you can just use the service as part of your design. For example, an S3 bucket can be used for storage, without any knowledge of the technical details of the S3 storage array. In the cloud, all the infrastructure components are already there and functional, and you don't have to build storage arrays or even databases from scratch; you can merely order or quickly configure the service, and you are off and running. When developing in the cloud, if your time line for development is longer than six months, most hosted cloud services will have changed and improved in that time frame, forcing you to take another look at your design.

- **Agile:** In this model, the focus is on process adaptability, and teams can be working simultaneously on the planning, design, coding, and testing processes. The entire process cycle is divided into a relatively shorter time frame, such as 1 month. At the end of the month, the first build of the product is demoed to the potential customers, feedback is provided, and that feedback is accepted into the next process cycle and the second version of the product. This process continues until a final production version of the product is delivered. This process might continue indefinitely if an application continually changes. Think of any application that you have installed on your personal devices and consider how many times that application gets updated. It's probably updated every few months at a minimum; for example, the Google Chrome browser updates itself at least every couple weeks. AWS has a number of services that can help with Agile deployment, including Elastic Beanstalk and CloudFormation.

- **Big Bang:** In this model, there is no specific process flow; when the money is available, development starts, and eventually software is developed. This model can work in the cloud as there are no capital costs to worry about, and work can proceed when there is a budget. But without planning and understanding of requirements, long-term projects may have to be constantly revised over time due to changes in the cloud and changes from the customer.

Before deploying applications in the cloud, you should carefully review your current development process and perhaps consider taking some of the steps in the Twelve-Factor App Methodology, which are described in the following sections. Your applications that are hosted in the cloud also need infrastructure; as a result, these rules for proper application deployment in the cloud don't stand alone; cloud infrastructure is also a necessary part of the rules. The following sections look at the 12 rules of the Twelve-Factor App Methodology from an infrastructure point of view and identify the AWS services that can help with each rule. This information can help you understand both the rules and the AWS services that can be useful in application development.

Rule 1: Use One Codebase That Is Tracked with Version Control to Allow Many Deployments

In development circles, this rule is non-negotiable; it must be followed. Creating an application usually involves three separate environments: development, testing, and production (see Figure 3-22). The same codebase should be used in each environment, whether it's the developer's laptop, a set of testing server EC2 instances, or the production EC2 instances. Operating systems, off-the-shelf software, dynamic-link libraries (DLLs), development environments, and application code are always defined and controlled by versions. Each version of application code needs to be stored separately and securely in a safe location. Multiple AWS environments can take advantage of multiple availability zones and multiple VPCs.

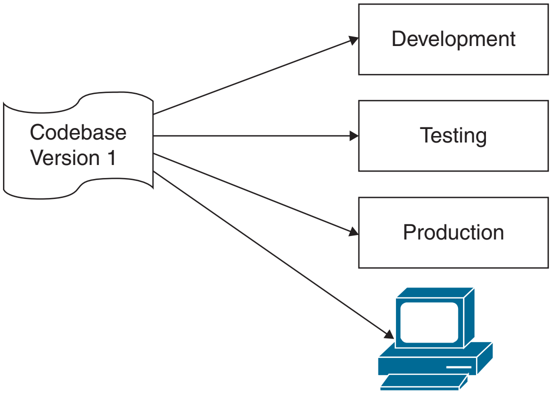


FIGURE 3-22 One Codebase, Regardless of Location

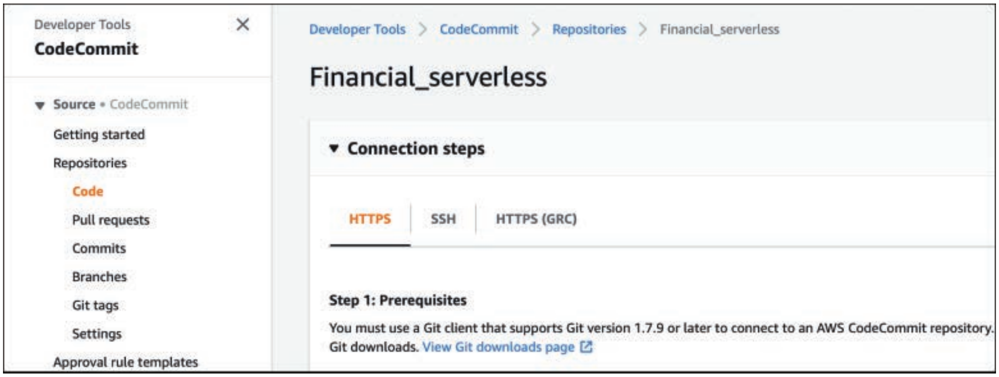
Developers typically use code repositories such as GitHub to store their code. As your codebase undergoes revisions, each revision needs to be tracked; after all, a single codebase might be responsible for thousands of deployments, and documenting and controlling the separate versions of the codebase just makes sense. Amazon has a code repository, called CodeCommit, that may be more useful than Git for applications hosted at AWS.

At the infrastructure level at Amazon, it is important to consider dependencies. The AWS infrastructure components to keep track of include the following:

- **AMIs:** Images for web, application, database, and appliance instances. AMIs should be version controlled.
- **EBS volumes:** Boot volumes and data volumes should be tagged by version number for proper identification and control.
- **EBS snapshots:** Snapshots used to create boot volumes are part of the (AMI).
- **Containers:** Each container image should be referenced by its version number.

AWS CodeCommit

CodeCommit is a hosted AWS version control service with no storage size limits (see Figure 3-23). It allows AWS customers to privately store their source and binary code, which are automatically encrypted at rest and at transit, at AWS. CodeCommit allows customers to store code versions at AWS rather than at Git without worrying about running out of storage space. CodeCommit is also HIPAA eligible and supports Payment Card Industry Data Security Standard (PCI DSS) and ISO 27001 standards.



Financial_serverless

▾ Connection steps

Step 1: Prerequisites

You must use a Git client that supports Git version 1.7.9 or later to connect to an AWS CodeCommit repository. Git downloads. View Git downloads page

FIGURE 3-23 A CodeCommit Repository

CodeCommit supports common Git commands and, as mentioned earlier, there are no limits on file size, type, and repository size. CodeCommit is designed for collaborative software development environments. When developers make multiple file changes, CodeCommit manages the changes across multiple files. S3 buckets also support file versioning, but S3 versioning is really meant for recovery of older versions of files; it is not designed for collaborative software development environments; as a result, S3 buckets are better suited for files that are not source code.

Rule 2: Explicitly Declare and Isolate Dependencies

Any application that you have written or will write depends on some specific components, such as a database, a specific operating system version, a required utility, or a software agent that needs to be present. You should document these dependencies so you know the components and the version of each component required by the application. Applications that are being deployed should never rely on the assumed existence of required system components; instead, each dependency needs to be declared and managed by a dependency manager to ensure that only the defined dependencies will be installed with the codebase. A dependency manager uses a configuration file to determine what dependency to get, what version of the dependency to get, and what repository to get it from. If there is a specific version of system tools that the codebase always requires, perhaps the system tools could be added to the operating system that the codebase will be installed on. However, over time, software versions for every component will change. An example of a dependency manager could be Composer, which is used with PHP projects, or Maven, which can be used with Java projects. Another benefit of using a dependency manager is that the versions of your dependencies will be the same versions used in the development, testing, and production environments.

If there is duplication with operating system versions, the operating system and its feature set can also be controlled by AMI versions, and CodeCommit can be used to host the different versions of the application code. CloudFormation also includes a number of helper scripts that can allow you to automatically install and configure applications, packages, and operating system services that execute on EC2 Linux and Windows instances. The following are a few examples of these helper scripts:

- **cfn-init:** This script can install packages, create files, and start operating system services.
- **cfn-signal:** This script can be used with a wait condition to synchronize installation timings only when the required resources are installed and available.
- **cdn-get-metadata:** This script can be used to retrieve metadata from the EC2 instance's memory.

Rule 3: Store Configuration in the Environment

Your codebase should be the same in the development, testing, and production environments. However, your database instances or your S3 buckets will have different paths, or URLs, used in testing or development. Obviously, a local database shouldn't be stored on a compute instance operating as a web server or as an application server. Other configuration components, such as API keys, plus database credentials for access and authentication, should never be hard-coded. You can use AWS Secrets for storing database credentials and secrets, and you can use IAM roles for accessing data resources at AWS, including S3 buckets, DynamoDB tables, and RDS databases. You can use API Gateway to store your APIs.

Development frameworks define environment variables through the use of configuration files. Separating your application components from the application code allows you to reuse your backing services in different environments, using environment variables to point to the desired resource from the development, testing, or production environment. Amazon has a few services that can help centrally store application configurations:

- **AWS Secrets:** This service allows you to store application secrets such as database credentials, API keys, and OAuth tokens.
- **AWS Certificate Manager (ACM):** This service allows you to create and manage public Secure Sockets Layer/Transport Layer Security (SSL/TLS) certificates used for any hosted AWS websites or applications. ACM also enables you to create a private certificate authority and issue X.509 certificates for identification of IAM users, EC2 instances, and AWS services.
- **AWS Key Management Services:** This service can be used to create and manage encryption keys.
- **AWS Systems Manager Parameter Store:** This service stores configuration data and secrets for EC2 instances, including passwords, database strings, and license codes.

Rule 4: Treat Backing Services as Attached Resources

All infrastructure services at AWS can be defined as backing services, and AWS services can be accessed by HTTPS private endpoints. Backing services hosted at AWS are connected over the AWS private network and include databases (for example, Relational Database Service [RDS], DynamoDB), shared storage (for example, S3 buckets, Elastic File System [EFS]), Simple Mail Transfer Protocol (SMTP) services, queues (for example, Simple Queue Service [SQS]), caching systems (such as ElastiCache, which manages Memcached or Redis in-memory queues or databases), and monitoring services (for example, CloudWatch, Config, CloudTrail).

Under certain conditions, backing services should be completely swappable; for example, a MySQL database hosted on premises should be able to be swapped with a hosted copy of the database at AWS without requiring a change to application code; the only variable that needs to change is the resource handle in the configuration file that points to the database location.

NOTE All backing services provided by AWS services have associated metrics that can be monitored using CloudWatch alarms and, optionally, CloudWatch events and rules.

Rule 5: Separate the Build and Run Stages

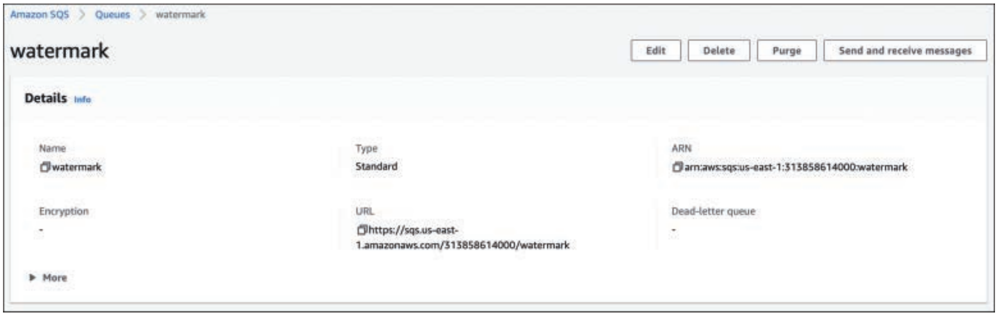
If you are creating applications that will be updated, whether on a defined schedule or at unpredictable times, you will want to have defined stages during which testing can be carried out on the application state before it is approved and moved to production. Amazon has several such PaaS services that work with multiple stages. As discussed earlier in this chapter, Elastic Beanstalk allows you to upload and deploy your application code combined with a configuration file that builds the AWS environment and deploys your application.

The Elastic Beanstalk build stage could retrieve your application code from the defined repo storage location, which could be an S3 bucket. Developers could also use the Elastic Beanstalk CLI to push your application code commits to AWS CodeCommit. When you run the CLI command **EB create** or **EB deploy** to create or update an EBS environment, the selected application version is pulled from the defined CodeCommit repository, and the application and required environment are uploaded to Elastic Beanstalk. Other AWS services that work with deployment stages include the following:

- **AWS CodePipeline:** This service provides a continuous delivery service for automating deployment of your applications using multiple staging environments.
- **AWS CodeDeploy:** This service helps automate application deployments to EC2 instances hosted at AWS or on premises.
- **AWS CodeBuild:** This service compiles your source code, runs tests on prebuilt environments, and produces code ready to deploy without your having to manually build the test server environment.

Rule 6: Execute an App as One or More Stateless Processes

Stateless processes provide fault tolerance for the instances running your applications by separating the important data records being worked on by the application and storing them in a centralized storage location such as an SQS message queue. An example of a stateless design using an SQS queue could be a design in which an SQS message queue is deployed as part of the workflow to add a corporate watermark to all training videos uploaded to an associated S3 bucket (see Figure 3-24). A number of EC2 instances could be subscribed to the watermark SQS queue; every time a video is uploaded to the S3 bucket, a message is sent to the watermark SQS queue. The EC2 servers subscribed to the SQS queue poll for any updates to the queue; when an update message is received by a subscribed server, the server carries out the work of adding a watermark to the video and then stores the video in another S3 bucket.



Name
watermark

Type
Standard

ARN
arn:aws:sqs:us-east-1:313858614000:watermark

Encryption
-

URL
https://sqs.us-east-1.amazonaws.com/313858614000/watermark

Dead-letter queue
-

FIGURE 3-24 Using SQS Queues to Provide Stateless Memory-Resident Storage for Applications

Others stateless options available at AWS include the following:

- **AWS Simple Notification Services:** This hosted messaging service allows applications to deliver push-based notifications to subscribers such as SQS queues or Lambda.
- **Amazon MQ:** This hosted managed message broker service specifically designed for Apache Active MQ is an open-source message broker service that provides functionality similar to that of SQS queues.
- **AWS Simple Email Service:** This hosted email-sending service includes an SMTP interface that allows you to integrate the email service into your application for communicating with an end user.

Let's look at an example of how these services can solve availability and reliability problems. Say that a new employee at your company needs to create a profile on the first day of work. The profile application runs on a local server, and each new hire needs to enter pertinent information. Each screen of information is stored within the application running on the local server until the profile creation is complete. This local application is known to fail without warning, causing problems and wasting time. You decide to move the profile application to the AWS cloud, which requires a proper redesign with hosted components to provide availability and reliability by hosting the application on multiple EC2 instances behind a load balancer in separate availability zones. Components such as an SQS queue can retain the user information in a redundant data store. Then, if one of the application servers crashes during the profile creation process, another server takes over, and the process completes successfully.

Data that needs to persist for an undefined period of time should always be stored in a redundant storage service such as a DynamoDB database table, an S3 bucket, an SQS queue, or a shared file store such as EFS. When the user profile creation is complete, the application can store the relevant records and can communicate with the end user by using Amazon Simple Email Service (SES).

Rule 7: Export Services via Port Binding

Instead of using a local web server installed on a local server host and accessible only from a local port, you should make services accessible by binding to external ports where the services are located and accessible using an external URL. For example, all web requests can be carried out by binding to an external port, where the web service is hosted and from which it is accessed. The service port that the application needs to connect to is defined by the development environment's configuration file (see the section "Rule 3: Store Configuration in the Environment," earlier in this chapter). The associated web service can be used multiple times by different applications and the different development, testing, and production environments.

Rule 8: Scale Out via the Process Model

If your application can't scale horizontally, it's not designed for dynamic cloud operation. Many AWS services, including these, are designed to automatically scale horizontally:

- **EC2 instances:** Instances can be scaled with EC2 Auto Scaling and CloudWatch metric alarms.
- **Load balancers:** The ELB load balancer infrastructure horizontally scales to handle demand.

- **S3 storage:** The S3 storage array infrastructure horizontally scales in the background to handle reads.
- **DynamoDB:** DynamoDB horizontally scales tables within an AWS region. Tables can also be designed as global tables, which can scale across multiple AWS regions.

Rule 9: Maximize Robustness with Fast Startup and Graceful Shutdown

User session information can be stored in Amazon ElastiCache or in in-memory queues, and application state can be stored in SQS message queues. Application configuration and bindings, source code, and backing services are hosted by AWS managed services, each with its own levels of redundancy and durability. Data is stored in a persistent packing storage location such as S3 buckets, RDS, or DynamoDB databases (and possibly EFS or FSx shared storage). Applications with no local dependencies and integrated hosted redundant services can be managed and controlled by a number of AWS management services.

- A web application hosted on an EC2 instance can be ordered to stop listening through Auto Scaling or ELB health checks.
- Load balancer failures are redirected using Route 53 alias records to another load balancer, which is assigned the appropriate elastic IP (EIP) address.
- The RDS relational database primary instance automatically fails over to the standby database instance. The primary database instance is automatically rebuilt.
- DynamoDB tables are replicated a minimum of six times across three availability zones throughout each AWS region.
- Spot EC2 instances can automatically hibernate when resources are taken back.
- Compute failures in stateless environments return the current job to the SQS work queue.
- Tagged resources can be monitored by CloudWatch alerts using Lambda functions to shut down resources.

Rule 10: Keep Development, Staging, and Production as Similar as Possible

With this rule, *similar* does not refer to the number of instances or the size of database instances and supporting infrastructure. Your development environment must be exact in the codebase being used but can be dissimilar in the number of instances or database servers being used. Aside from the infrastructure components, everything else in the codebase must remain the same. CloudFormation can be used to automatically build each environment using a single template file with conditions that define what infrastructure resources to build for each development, testing, and production environment.

Rule 11: Treat Logs as Event Streams

In development, testing, and production environments, each running process log stream must be stored externally. At AWS, logging is designed as event streams. CloudWatch logs or S3 buckets can be created to store EC2 instances' operating system and application logs. CloudTrail logs, which track all API calls to the AWS account, can also be streamed to CloudWatch logs for further analysis. Third-party monitoring solutions support AWS and can interface with S3 bucket storage. All logs and reports generated at AWS by EC2 instances or AWS managed services eventually end up in an S3 bucket.

Rule 12: Run Admin/Management Tasks as One-Off Processes

Administrative processes should be executed using the same method, regardless of the environment in which the administrative task is executed. For example, an application might require a manual process to be carried out; the steps to carry out the manual process must remain the same, whether they are executed in the development, testing, or production environment.

The goal in presenting the rules of the Twelve-Factor App Methodology is to help you think about your applications and infrastructure and, over time, implement as many of the rules as possible. This might be an incredibly hard task to do for applications that are simply lifted and shifted to the cloud. Newer applications that are completely developed in the cloud should attempt to follow these rules as closely as possible.

Exam Preparation Tasks

As mentioned in the section "How to Use This Book" in the Introduction, you have a couple of choices for exam preparation: the exercises here, Chapter 14, "Final Preparation," and the exam simulation questions in the Pearson Test Prep Software Online.

Review All Key Topics

Review the most important topics in the chapter, noted with the key topics icon in the outer margin of the page. Table 3-5 lists these key topics and the page number on which each is found.

Key Topic

Table 3-5 Chapter 3 Key Topics

Key Topic Element	Description	Page Number
Figure 3-2	Increasing application availability by adding compute resources	89
Figure 3-4	Increasing fault tolerance	90
Table 3-2	Avoiding single points of failure	91
Figure 3-5	Using elastic IP addresses for high availability	92
Table 3-3	Planning for high availability, fault tolerance, and redundancy	93
List	Recovery point objective	94
List	Recovery time objective	95
Figure 3-9	Warm standby response	98
Figure 3-11	Hot site response	100
Paragraph	CloudFormation templates	107
Section	Third-party solutions	114
Figure 3-19	Using Elastic Beanstalk to create infrastructure and install an application	117
List	Elastic Beanstalk tasks	118
Paragraph	Updating Elastic Beanstalk applications	119

Define Key Terms

Define the following key terms from this chapter and check your answers in the glossary:

service-level agreement (SLA), recovery point objective (RTO), recovery time objective (RPO), pilot light, warm standby, hot site, multi-region, Information Technology Infrastructure Library (ITIL), Scrum, Agile, JavaScript Object Notation (JSON)

Q&A

The answers to these questions appear in Appendix A. For more practice with exam format questions, use the Pearson Test Prep Software Online.

1. How do you achieve AWS's definition of high availability when designing applications?
2. What AWS service is not initially designed with fault tolerance or high availability?

3. What is the easiest way to remove single points of failure when operating in the AWS cloud?
4. What is the difference between a pilot light solution and a warm standby solution?
5. What is the difference between RPO and RTO?
6. What two AWS database services allow you to deploy globally across multiple AWS regions?
7. How can you build a self-serve portal for developers at AWS and control what architecture is deployed?
8. What service allows you to deploy an architectural solution for an application that you have already written?

Index

A

accelerated computing instances, 224
ACID (atomicity, consistency, isolation and durability), 442–443
ACM (Amazon Certificate Manager), 546
active-active application deployment, 100–102
AD DS (Active Directory Domain Service), identity federation, 503–505
Agile, 104
alarms, CloudWatch, 259
  action settings, 263–264
  creating, 262–263
ALB (Application Load Balancing), 266, 270, 271–272
  access logs, 284
  connection draining, 282
  creating a load balancer, 272–274
  health checks, 279–280
  HTTPS listener security settings, 276–277
  maintaining user sessions, 278
  monitoring with CloudWatch, 283
  rule choices, 274–276
  security, 280–283
  SNI (Server Name Identification), 281
  sticky session support, 278–279
  target groups, 271
    attributes, 277
    routing, 277
Amazon Aurora, 100–102, 427, 433
  communicating with, 432
  deployment options, 428–429
  failover, 431
  storage, 429–431
Amazon Inspector, 525–526
Amazon Lightsail, 232–233
Amazon MQ, 152
Amazon Redshift, 447–448
  clusters, 448
  columnar data storage, 448
  Concurrency Scaling, 448
  replication, 448–449
Amazon SNS (Simple Notification Services), 144–147
AMIs (Amazon Machine Images), 233–235
  AWS Linux, 235–236
  AWS Marketplace, 237
  build considerations, 240–242
  choosing, 235
  custom, 237–239
  custom instance store, 239–240
  Windows, 236
API Gateway, 156–160
APIs (application programming interfaces), 11, 13–14, 156–157
application integration services, 144
  Amazon SNS (Simple Notification Services), 144–147
  Amazon SQS (Simple Queue Service), 147–149
  AWS Step Functions, 149–152

Application Load Balancer, 20
application security, 19–20
applications
  migrating to AWS, 22–23, 24
    lift and shift/re-hosting, 23
    replacing with SaaS application, 24
  security, 517
  stateless, 139
architecture(s)
  active-active application deployment, 100–102
  designing for high availability, 88–90
    adding fault tolerance, 90–91
    removing single points of failure, 91–93
  and disaster recovery, 94
    RPO (recovery point objective), 94
  ECS (Elastic Container Service), 243
  fault tolerance, 89
  stateful design, 138–140
  stateless design, 139–140
  storing user state information, 140–142
authentication, IAM (Identity and Access Management), 461–464
auto scaling, 290
  ASGs (auto scaling groups), 291–296
  cooldown period, 296
  DynamoDB, 439–440
  launch templates, 290
  lifecycle hooks, 297
  termination policy, 296–297
automation, 103–104, 113
  CloudFormation, 11, 105–106
    change sets, 113
    creating an EC2 instance, 111–112
    stacks, 109–111
    templates, 105, 107–109
  Elastic Beanstalk, 117–119
    application updates, 119–121
  third-party solutions, 114–115
availability, 33–34, 56
AWS (Amazon Web Services), 3, 4, 10, 29. *See also* architecture(s); EC2 (Elastic Compute Cloud) instances; storage; VPC (virtual private cloud)
  Application Load Balancer, 20
  automation, 103–104
  availability, 34
  AZs (availability zones), 6, 39–40, 353
    distribution, 40–41
    multiple, 42–43
  BAA (Business Associate Addendum), 50
  budgets, 595–596
  BYOIP (Bring-Your-Own IP), 368–370
  CloudEndure Migration, 57–58
  CloudFormation, 11, 105–106
    change sets, 113
    creating an EC2 instance, 111–112
    stack sets, 113–114
    stacks, 109–111
    templates, 105, 107–109
  CloudFront, 73–74
    HTTPS access, 76
    OAI (origin access identity), 77
    origin failover, 78–79
    regional edge caches, 75
    restricting distribution of content, 78
    securing access to content, 75–76
    serving private content, 76–77
    use cases, 74–75
    and WAF, 77
  CloudTrail, 11, 520–521, 522
    creating trails, 521–522
  CloudWatch, 33, 247–249, 264
    agent, 253
    alarms, 259, 262–264
    basic monitoring, 249–250
    and EC2 Auto Scaling, 261
    events, 260
    included services, 255–256
    logs, 250–253

metrics, 248, 249, 250, 258–259
namespace, 257
pricing, 261
timestamps, 260
CodeCommit, 124–125
compliance rules, 45–47
FedRAMP (Federal Risk and Authorization Management Program), 51
global frameworks, 49–50
HIPAA (Health Insurance Portability and Accountability Act), 50
NIST (National Institute of Standards and Technology), 51–52
North American frameworks, 48–49
shared responsibility model, 48
Cost Explorer, 593–595, 597–599
cost(s)
billing, 592–593
budgets, 595–596
calculating, 55–56, 555
compute, 558–559
of management services, 556
managing, 599–600
data storage, 170
data transfer, 319–320
DataSync, 320
Direct Connect, 320
SFTP (SSH File Transfer Protocol), 321–322
Snow Family, 320–321
EBS (Elastic Block Storage), 33
EC2-Classic networking, 336
ECS (Elastic Container Service), 33
edge locations, 63–64
services, 64
Elastic Beanstalk, 14, 117–119
ELB (Elastic Load Balancing), 58
essential characteristics
broad network access, 6
on-demand self-service, 5
measured service, 7–8
rapid elasticity, 7
resource pooling, 6
Fargate, 245–246
free trial period, 20–21
GuardDuty, 524–525
IaaS (infrastructure as a service), 10–12
IAM (Identity and Access Management), 19
identity federation, 503–505
KMS (Key Management Service), 543–544, 545
latency, 53–54
managed services, 10–11, 16, 33
API Gateway, 156–160
Lamdba@Edge, 78–79
migrating to, 22–23
applications with local dependencies, 23
CloudEndure Migration, 57–58
determining what problem needs to be solved, 21–22
lift and shift/re-hosting, 23
multi-tier architecture solutions
design problems to overcome, 58–61
protecting against application failure, 62–63
redundancy, 61–62
three-tier design, 60
VPC (virtual private cloud), 56–58
operational benefits, 15
servers, 15
storage, 15
Organizations, 509–511
PaaS (platform as a service), 12–14
Cloud Foundry, 13
Heroku, 13
partnering with, 335–336
Promotional Credit, 9
RAM (Resource Access Manager), 511–512
RDS (Relational Database Service), 11, 14, 38

regions, 36–39
  choosing, 43–44
  GovCloud, 52–53
  services offered in, 54–55
reliability, 35
Route 53, 64–65
  alias records, 68–69
  health checks, 49–66
  resolver, 69–71
  routing policies, 67
  services, 65–66
  traffic flow policies, 67–68
Secret Manager, 523–524
security, 18
  application, 19–20
  data, 18–19
  ELB (Elastic Load Balancer), 20
  network, 19
  WAF (Web Application Firewall), 20
serverless computing, 154
Service Catalog, 115
  IAM group constraints, 116–117
  portfolios, 116
service quotas, 80–81
shared file storage, 306
SLAs (service-level agreements), 17, 43, 102–103
Storage Gateway, 322, 324
  File Gateway, 322
  Tape Gateway, 322–324
  Volume Gateway, 322
STS (Security Token Service), 501–502
tiered pricing, 557
Trusted Advisor, 526–528
uptime, 94
user sessions
  distributed, 142
  management, 142–144
  sticky, 142
  storing user state information, 140–142
VPC (virtual private cloud), 334–335, 337–338
  connectivity options, 387, 407–409
  default, 352–353
  determining number of needed, 347–349
  endpoints, 390–395
  external connections, 339, 395–396, 398, 399–406
  flow logs, 385–386
  hosted services, 338
  hypervisor, 340–341
  Layer 3 networks, 339
  peering, 387–390
  physical infrastructure, 339–340
  route tables, 356, 357–361
Well-Architected Framework, 24
Well-Architected Framework tool, 26–27
AWS Certified Solutions Architect - Associate (SAA-C02) exam, 603–605, 606
  objectives, 605
  preparation tips, 606–607
  scheduling, 608
  suggested plan for review/study, 613
  tools for final preparation, 609–613
AWS Config, 590–592
AWS Linux AMIs, 235–236
AWS Marketplace, AMIs (Amazon Machine Images), 237
AWS Shield, 71
AZs (availability zones), 6, 39–40, 353
  distribution, 40–41
  multiple, 42–43

B

BAA (Business Associate Addendum), 50
backup and restoration, 95
  DynamoDB, 445
bare-metal instances, 224–225
billing costs. *See also* pricing, AWS (Amazon Web Services), 592–593
Blackfoot devices, 342
block storage, 175

EBS (Elastic Block Storage), 177–178, 183–184
attaching a volume, 182–183
boot and data volumes, 178
burst credits, 180–181
snapshots, 184–186
volume types, 178–180
building
AMIs (Amazon Machine Images), 240–242
serverless apps, 160–161
create a static website, 161–162
create the serverless backend components, 162–163
handle user authentication, 162
register for conference, 164
set up the API Gateway, 163–164
burst credits, 180–181
BYOIP (Bring-Your-Own IP), 368–370

C

choosing
AMIs (Amazon Machine Images), 235, 237
AWS regions, 43–44
CIDR block
creating, 349
primary, 349–351
secondary, 351–352
CLB (Classic Load Balancing), 265, 266, 269–270
cloud computing, 4. *See also* AWS (Amazon Web Services)
adopter mindsets, 8–9
availability, 34
AWS (Amazon Web Services)
AZs (availability zones), 6
broad network access, 6
on-demand self-service, 5
elasticity, 7
measured service, 7–8
resource pooling, 6
billing, 8
compliance rules, 44–45
deployment methodologies, 121–122, 123
elasticity, 286–287
GCP (Google Cloud Platform), 3
IaaS (infrastructure as a service), 3, 10–12
Microsoft Azure, 3, 4
Oracle Cloud, 3
PaaS (platform as a service), 3, 12–14
reliability, 35
SLAs (service-level agreements), 17
Twelve-Factor App Methodology, 121, 123
execute an app as one or more stateless processes, 128–129
explicitly declare and isolate dependencies, 125
export services via port binding, 129
keep development, staging, and production similar, 130–131
maximize robustness with fast startup and graceful shutdown, 130
run admin/management tasks as one-off processes, 131
scale out via the process model, 129–130
separate the build and run stages, 127
store configuration in the environment, 126
treat logs as event streams, 131
treat tracking services as attached resources, 126–127
version control, 123–124
Cloud Foundry, 13
Cloud9, 14
CloudEndure Migration, 57–58
CloudFormation, 11, 105–106
change sets, 113
creating an EC2 instance, 111–112

stack sets, 113–114
stacks, 109–111
templates, 105, 107–109
CloudFront, 73–74
HTTPS access, 76
OAI (origin access identity), 77
origin failover, 78–79
regional edge caches, 75
restricting distribution of content, 78
securing access to content, 75–76
serving private content, 76–77
use cases, 74–75
and WAF, 77
CloudHSM, 545
CloudTrail, 11, 520–521, 522
creating trails, 521–522
CloudWatch, 33, 144–145, 247–249, 264
agent, 253
alarms, 259
action settings, 263–264
creating, 262–263
ALB monitoring, 283
basic monitoring, 249–250
and EC2 Auto Scaling, 261
events, 260
included services, 255–256
logs, 250–253
metrics, 248, 249, 250, 258–259, 425
namespace, 257
pricing, 261
timestamps, 260
CodeCommit, 124
compliance, 44–45
AWS (Amazon Web Services), 45–47
FedRAMP (Federal Risk and Authorization Management Program), 51
FISMA (Federal Information Security Modernization Act), 51
global frameworks, 49–50
HIPAA (Health Insurance Portability and Accountability Act), 50
NIST (National Institute of Standards and Technology), 51–52
North American frameworks, 48–49
shared responsibility model, 48
storage, 176
compute-optimized instances, 223
containers
management services
EKS (Elastic Kubernetes Service), 246–247
Fargate, 245–246
types of, 243–244
versus VMs, 243
controlled storage, 184
creating
EC2 (Elastic Compute Cloud) instances, 111–112
IAM policies, 484
load balancer with ALB, 272–274
VPC (virtual private cloud)
using AWS CLI, 347
using the Launch VPC wizard, 345–347
using the VPC Wizard, 344–345
custom instance store AMIs, 239–240

D

data security, 18–19
database(s), 38–39. *See also* SQL
Amazon Aurora, 100–102, 427
communicating with, 432
deployment options, 428–429
failover, 431
storage, 429–431
clustering, 85
design solutions, 582–584
DynamoDB, 433–434, 435, 437
Auto Scaling, 439–440
backup and restoration, 445
burst capacity, 442
capacity, 438–439

comparison with SQL, 434
data consistency, 441
DAX, 444
global tables, 443
queries, 435–436
storage node design, 442
tables, 434–435, 437
ElastiCache, 445–447
installing, 423–425
NoSQL, 437
OLTP (online transaction processing), 435
performance monitoring, 425
reducing costs, 581–582
dedicated instances, 226
on-demand self-service, AWS (Amazon Web Services), 5
DevOps, 104
disaster recovery, 94
backup and restoration, 95
hot site solution, 99–100
pilot light deployment, 95–97
RPO (recovery point objective), 94
RTO (recovery time objective), 95
warm standby solution, 96–99
Docker, 243–244, 246
DynamoDB, 433–434, 435, 437
and ACID, 442–443
adaptive capacity, 439
Auto Scaling, 439–440
backup and restoration, 445
burst capacity, 442
comparison with SQL, 434
data consistency, 441
DAX, 444
Paxos, 441
queries, 435–436
storage node design, 442
tables, 434–435, 437
capacity, 438–439
global, 443

E

EBS (Elastic Block Storage), 33, 177–178, 182, 183–184, 297, 304
attaching a volume, 182–183
boot and data volumes, 178
burst credits, 180–181
encryption, 535–536, 537–538
creating a master key, 535–536
encrypted data types, 536–537
general purpose SSD, 180–181
io1 and io2 drives, 304–305
pricing, 576
snapshots, 184, 186
administration, 185–186
taking from a Linux instance, 184
taking from a Windows instance, 185
tags, 578–579
volume types, 178–180
EC2 (Elastic Compute Cloud) instances, 4, 7, 39, 58, 152, 209, 217
accelerated computing, 224
AMIs (Amazon Machine Images), 233–235
AWS Linux, 235–236
AWS Marketplace, 237
build considerations, 240–242
choosing, 235
custom, 237–239
custom instance store, 239–240
Windows, 236
auto scaling, 285–289
ASGs (auto scaling groups), 291–296
cooldown period, 296
launch configuration, 290
launch templates, 290
lifecycle hooks, 297
termination policy, 296–297
bare-metal, 224–225
burstable performance, 420
changing the current instance type, 229–232

choosing, 220, 221
CloudWatch integration, 261
compute-optimized, 223
creating, 111–112
dedicated, 226
dedicated hosts, 225–226
enhanced networking, 227–228
f1, 224
Fleet instances, 573–575
g3, 224
general-purpose, 221
high-memory, 223
HVM (hardware virtual machine) images, 220
launch templates, 228–229
memory-optimized, 223, 420
micro, 221
naming conventions, 218–219
network performance, 226–227
pinging, 377
pricing, 559–560, 575
convertible reserved instance, 564
on-demand instance limits, 560–561
limits calculator, 562
payment options, 564
regional and zonal RIs, 565–567
requesting a quota change, 561–562
RI (reserved instance), 562–563, 565
savings plans, 567–568
scheduled RIs, 565
standard reserved instance, 564
term commitment, 563
PV (paravirtual) images, 220
spot capacity pools, 572–573
spot fleets, optimization strategies, 571–572
spot instances, 568–571
standard, 420
storage-optimized, 224, 305–306
t, 221–222
virtual cores, 219
x1, 223
z1d, 224
ECDSA (Elliptic Curve Digital Signature Algorithm), 281
ECS (Elastic Container Service), 33, 242, 244
architecture, 243
containers
types of, 243–244
versus VMs, 243
launch types, 244
registry, 245
task placement strategy, 245
edge locations, 63–64
AWS Shield, 71
services, 64
EFS (Elastic File System), 174, 187–188, 191, 306, 307–309, 312–313
DataSync, 191
lifecycle management, 190–191
performance modes, 188, 309
general purpose, 309
Max I/O, 309
pricing, 577
security, 189–190, 312
storage classes, 310–311
throughput modes, 188–189, 311
Bursting, 311–312
provisioned, 312
EIPs (elastic IP addresses), 364–366
EKS (Elastic Kubernetes Service), 246–247
Elastic Beanstalk, 14, 117–119
application updates, 119–121
ElastiCache, 174, 445–447
elasticity, 42, 286–287
AWS (Amazon Web Services), 7
ELB (Elastic Load Balancing), 20, 58, 101, 264–265, 269
choices and features, 266–267
health checks, 268–269

redundancy in design, 267–268
and security groups, 377–378
target group, 265–266
encryption
AES (Advanced Encryption Standard), 535
CloudHSM, 545
EBS (Elastic Block Storage), 535–536, 537–538
creating a master key, 535–536
encrypted data types, 536–537
envelope, 544–545
KMS (Key Management Service), 543–544, 545
S3 (Simple Storage Service), 540–542
endpoints, 390–391
Aurora, 432
gateway, 391–392
interface, 392–393
PrivateLink, 393–395
ephemeral ports, 383–385
ephemeral storage, 175, 186
architecture, 186–187
event(s)
CloudWatch, 260
notifications, 145–146
exam. *See* AWS Certified Solutions Architect - Associate (SAA-C02) exam

F

f1 instances, 224
failures
and disaster recovery, 94
protecting against, 62–63
and SLAs, 102–103
fault tolerance, 85, 88, 89
designing for, 90–91
FedRAMP (Federal Risk and Authorization Management Program), 51
firewalls. *See also* security
NACLs (network access control lists), 379, 380–381
custom setup, 382
ephemeral ports, 383–385
planning, 385
rule processing, 381–382
security groups, 19, 144
WAF (Web Application Firewall), 72–73
FISMA (Federal Information Security Modernization Act), 51
FSx, 175, 191–192, 306, 317
accessing, 193
features, 193–194
for Lustre, 306–307, 317–318
multi-AZ deployment, 192–193
performance, 316–317
single-AZ deployment, 192
system performance, 193
for Windows File Server, 315–316
functions, Lambda, 153, 154

G

g3 instances, 224
Gartners Magic Quadrant, 3
GCP (Google Cloud Platform), 3
general purpose instances, 221
GitHub, 124
GLB (Gateway Load Balancer), 266
global compliance frameworks, 49–50
GovCloud, 52–53
GuardDuty, 524–525

H

HDDs (hard disk drives), 175
health checks
ALB (Application Load Balancing), 279–280
ELB (Elastic Load Balancing), 268–269
Route 53, 49–66
Heroku, 13, 121

high availability, 29, 34, 85. *See also* availability
- designing for, 88–90
- removing single points of failure, 91–93

high-memory instances, 223
HIPAA (Health Insurance Portability and Accountability Act), 50
hyperthreading, 219

I

IaaS (infrastructure as a service), 3, 10–12
- VPC (virtual private cloud), 10

IAC (infrastructure as code), 104
IAM (Identity and Access Management), 19, 105, 116–117, 457–460, 503
- account details, 475–476
- actions, 466–467
- authentication, 461–464
- authorization process, 465–466
- best practices, 506–507
- cross-account access to AWS resources, 499–501
- groups, 474
- MFA (multifactor authentication), 479
- password policies, 476
- policies, 479
  - actions, 488–489
  - conditional elements, 494–495
  - control options, 489
  - creating, 484
  - custom, 482
  - elements, 484–486
  - identity-based, 480–482
  - job function, 480–481
  - in-line, 483
  - managed, 480–481
  - permission boundary, 489–491
  - permissions, 491–493
  - resource-based, 482–483
  - syntax and grammatical rules, 486–487
  - versions, 493–494
- policy definitions, 460–461
- policy evaluation logic, 466
- requesting access to AWS resources, 464–465
- roles, 496–497
  - service-linked, 497–498
  - using with mobile applications, 499
- rotating access keys, 477–479
- security tools, 507–509
- signing in as an IAM user, 474–475
- tags, 495–496
- users, 467–468
  - access keys, 472–474
  - IAM user, 470–472
  - root user, 468–470

identity federation, 503–505
installing, databases, 423–425
instance storage, 186
- architecture, 186–187

instances. *See* EC2 (Elastic Compute Cloud) instances
IOPS (input/output per second), 177
IP addresses
- elastic, 92, 364–366
- IPv6, 370–371
- private IPv4, 361–362
- public IPv4, 362–364, 366–367

ITIL (Information Technology Infrastructure Library), 104

J-K

JSON (JavaScript Object Notation), 105
- CloudFormation templates, 107–109

Kubernetes, 246–247. *See also* EKS (Elastic Kubernetes Service)

L

Lambda, 152–156
- functions, 153, 154

latency, 53–54

launch templates, 228–229
Layer 3 networks, 339
Linux, workload testing, 181
load balancing, 92, 139, 264–265. *See also* ELB (Elastic Load Balancing)
 ALB (Application Load Balancing), 270, 271–272
  access logs, 284
  connection draining, 282
  creating a load balancer, 272–274
  health checks, 279–280
  HTTPS listener security settings, 276–277
  maintaining user sessions, 278
  monitoring, 283
  rule choices, 274–276
  target groups, 271, 277
 CLB (Classic Load Balancing), 265, 266, 269–270
 NLB (Network Load Balancing), 284–285
 sticky sessions, 139–140, 142

M

managed services, 10–11, 33, 156–160
measured service, AWS (Amazon Web Services), 7–8
memory-optimized instances, 223
metrics, CloudWatch, 258–259, 425
micro instances, 221
Microsoft Azure, 3, 4, 11, 13–14
migrating to AWS
 applications, 22–23, 24
  lift and shift/re-hosting, 23
  replacing with SaaS application, 24
 CloudEndure Migration, 57–58
 determining what problem needs to be solved, 21–22
monitoring. *See also* CloudWatch, planning for, 253–255

N

NACLs (network access control lists), 144, 379, 380–381
 custom setup, 382
 ephemeral ports, 383–385
 planning, 385
 rule processing, 381–382
naming conventions, EC2 (Elastic Compute Cloud) instances, 218–219
network(s)
 access, AWS (Amazon Web Services), 6
 data transfer costs, 585–587, 590
 design solutions, 587–588
 public versus private traffic charges, 588–590
 reducing costs, 584–585
 security, 19
NIST (National Institute of Standards and Technology), 4–5, 51–52, 286
Nitro, 340
NLB (Network Load Balancing), 266, 284–285
North American compliance frameworks, 48–49

O

OAI (origin access identity), 77
object storage, 174–175. *See also* storage
 legal hold, 542–543
 S3 (Simple Storage Service), 194–195, 204, 206–207
  access points, 205–206
  batch operations, 200–201
  buckets, 194, 196–198
  CRR (cross-region replication), 202–203
  data consistency, 198
  encryption, 540–542
  inventory, 203–204
  object lock, 201–202
  object lock policies, 542

performance, 204
permissions, 196
replication, 202
SRR (same-region replication), 203
storage classes, 198–200
Storage Lens, 203
unlimited storage, 196
versioning, 204–205
OLTP (online transaction processing), 435

P

PaaS (platform as a service), 3, 12–14
Cloud Foundry, 13
Heroku, 13
PCI (Payment Card Industry), compliance rules, 46–47
performance
comparison of storage options, 579–581
EC2 (Elastic Compute Cloud) instances, 226–227
FSx, 316–317
latency, 53–54
reliability, 35, 56
S3 (Simple Storage Service), 204
storage, 176
and the Well-Architected Framework, 25
PIOPS (provisioned IOPS), 177
policies
bucket, 538–540
IAM (Identity and Access Management), 479
actions, 488–489
conditional elements, 494–495
control options, 489
creating, 484
custom, 482
elements, 484–486
identity-based, 480–482
job function, 480–481
in-line, 483
managed, 480–481
permission boundary, 489–491
permissions, 491–493
resource-based, 482–483
syntax and grammatical rules, 486–487
versions, 493–494
object lock, 542–543
pricing
CloudWatch, 261
compute costs, 558–559
EC2, 559–560, 575
on-demand instance limits, 560–561
savings plans, 567–568
inbound and outbound traffic charges, 367–368
management, 590–592
public versus private traffic charges, 588–590
RI (reserved instance), 562–563, 565
convertible, 564
payment options, 564
regional and zonal, 565–567
scheduled, 565
standard, 564
spot instances, 568–571
storage
AWS Backup, 576–577
EBS (Elastic Block Storage), 576
EFS (Elastic File System), 577
S3 buckets, 575–576
S3 Glacier, 576
tiered, 557
public cloud, 4–5. *See also* cloud computing

R

RDS (Relational Database Service), 11, 14, 38, 175, 417, 426–427. *See also* database(s)
Aurora, 427, 433

communicating with, 432
deployment options, 428–429
failover, 431
storage, 429–431
best practices, 425–426
CloudWatch metrics, 425
database instances, 418–419
burstable performance, 420
class types, 420
memory-optimized, 420
standard, 420
engines, 417
failures, 62
high-availability design
failover, 422
multi-AZ deployment, 420–421, 423
replication, 422
performance monitoring, 425
setup options, 423–425
reliability, 33–34, 35, 56
resiliency, 29
resource pooling, 287–288
AWS (Amazon Web Services), 6
Route 53, 64–65
alias records, 68–69
health checks, 49–66
resolver, 69–71
routing policies, 67
services, 65–66
traffic flow policies, 67–68
route tables, 360–361
custom, 357–360
main, 357
RPO (recovery point objective), 94
RTO (recovery time objective), 95

S

S3 (Simple Storage Service), 174–175, 194, 195, 204, 206–207
access points, 205–206
batch operations, 200–201
buckets, 194, 196–198
security, 538–540
versioning, 204–205
CRR (cross-region replication), 202–203
data consistency, 198
encryption, 540–542
Glacier, 18, 207, 209
archives, 208
Deep Archive, 208–209
object lock policies, 542
pricing, 207
storage at rest, 543
vaults, 208
inventory, 203–204
Object Lock, 201–202
object lock policies, 542
objects, 194–195
performance, 204
permissions, 196
pricing, 575–576
replication, 202
snapshots, 184
SRR (same-region replication), 203
storage classes, 198–200
Storage Lens, 203
unlimited storage, 196
SAML (Security Association Markup Language), 503
Scrum, 104
Secret Manager, 523–524
security, 18, 33, 48, 335. *See also* encryption
ACM (Amazon Certificate Manager), 546
ALB (Application Load Balancing), 280–283
application, 19–20
applications, 517
data, 18–19
EFS (Elastic File System), 189–190, 312

ELB (Elastic Load Balancer), 20
groups, 19, 144, 371–374
administrative access, 376–377
application server inbound ports, 375–376
custom, 374–375
database server inbound ports, 376
ELB (Elastic Load Balancing), 377–378
pinging an EC2 instance, 377
planning, 378–379
IAM (Identity and Access Management), 19, 105, 116–117, 457–460
account details, 475–476
actions, 466–467
authentication, 461–464
authorization process, 465–466
conditional elements, 494–495
creating policies, 484
cross-account access to AWS resources, 499–501
custom policies, 482
groups, 474
IAM user, 470–472
identity-based policies, 480–482
job function policies, 480–481
in-line policies, 483
managed policies, 480–481
MFA (multifactor authentication), 479
password policies, 476
permission boundaries, 489–491
permissions, 491–493
policies, 479
policy actions, 488–489
policy control options, 489
policy definitions, 460–461
policy elements, 484–486
policy evaluation logic, 466
policy syntax, 486–487
policy versions, 493–494
requesting access to AWS resources, 464–465
resource-based policies, 482–483
roles, 496–498, 499
root user, 468–470
rotating access keys, 477–479
signing in as an IAM user, 474–475
tags, 495–496
tools, 507–509
user access keys, 472–474
users and groups, 467–468
identity federation, 503–505
network, 19
S3 buckets, 538–540
WAF (Web Application Firewall), 20
Well-Architected Framework, 25
serverless apps, building, 160–161
create a static website, 161–162
create the serverless backend components, 162–163
handle user authentication, 162
register for conference, 164
set up the API Gateway, 163–164
serverless computing, 152–156
Service Catalog, 115
IAM group constraints, 116–117
portfolios, 116
session affinity, 139–140
shared file storage, 306
EFS (Elastic File System), 306
FSx, 306
FSx for Lustre, 306–307
shared responsibility model, 48
shared security model, 335
single points of failure, removing, 91–93
SLAs (service-level agreements), 17, 43, 102–103
snapshots, 184, 186
administration, 185–186
taking from a Linux instance, 184
taking from a Windows instance, 185

SQL
  Amazon Redshift, 447–448
    clusters, 448
    columnar data storage, 448
    Concurrency Scaling, 448
    replication, 448–449
  comparison with DynamoDB, 434
  queries, 435–436
  schema, 434
SSDs (solid-state drives), 175, 180–181, 305
SSO (single sign-on), 140
  identity federation, 503–505
state machines, 149
  activity tasks, 150–151
  service tasks, 151
stateful design, 138–140
  changing user state location, 140–142
stateless design, 139–140
  changing user state location, 140–142
Step Functions, 149–152. *See also* state machines
  workflows, 151
sticky sessions, 139–140, 142, 278–279
storage, 141–142, 169, 170
  Aurora, 429–431
  AWS (Amazon Web Services), 15
  block, 175
  comparison of options, 313–315
  controlled, 184
  data security, 18–19
  determining workload requirements, 176–177
  EBS (Elastic Block Storage), 177–178, 183–184
    attaching a volume, 182–183
    burst credits, 180–181
    encryption, 535–538
    general purpose SSD, 180–181
    pricing, 576
    snapshots, 184–186
    tags, 578–579
    volume types, 178–180
  EFS (Elastic File System), 174, 187–188, 191, 307–309
    DataSync, 191
    lifecycle management, 190–191
    performance modes, 188, 309
    pricing, 577
    security, 312
    storage classes, 310–311
    throughput modes, 188–189, 311–312
  ElastiCache, 174, 445–447
  ephemeral, 175
  FSx, 175, 191–192, 317
    accessing, 193
    features, 193–194
    for Lustre, 317–318
    multi-AZ deployment, 192–193
    performance, 316–317
    single-AZ deployment, 192
    system performance, 193
    for Windows File Server, 315–316
  instance, 186–187
  object, 174–175
  Paxos, 441
  performance comparison, 579–581
  personal records, 169
  RDS (Relational Database Service), 175
  resiliency, 29
  S3 (Simple Storage Service), 174–175, 194, 195, 204, 206–207
    access points, 205–206
    batch operations, 200–201
    buckets, 194, 196–198
    CRR (cross-region replication), 202–203
    data consistency, 198
    encryption, 540–542
    inventory, 203–204

objects, 194–195
performance, 204
permissions, 196
replication, 202
security, 538–540
SRR (same-region replication), 203
storage classes, 198–200
Storage Lens, 203
unlimited storage, 196
versioning, 204–205
S3 Glacier, 207, 209
archives, 208
Deep Archive, 208–209
pricing, 576
vaults, 208
shared file, 306
EFS (Elastic File System), 306
FSx, 306
FSx for Lustre, 306–307
stateful, 138–140
user state information, 140–142
storage-optimized EC2 instances, 224, 305–306

T

t instances, 221–222
tags, 578–579
cost allocation, 579
templates
CloudFormation, 105, 107–109
launch, 228–229
Trusted Advisor, 526–528
Twelve-Factor App Methodology, 121, 123
execute an app as one or more stateless processes, 128–129
explicitly declare and isolate dependencies, 125
export services via port binding, 129
keep development, staging, and production similar, 130–131
maximize robustness with fast startup and graceful shutdown, 130
run admin/management tasks as one-off processes, 131
scale out via the process model, 129–130
separate the build and run stages, 127
store configuration in the environment, 126
treat logs as event streams, 131
treat tracking services as attached resources, 126–127
version control, 123–124

U

user sessions
distributed, 142
management, 142–144
sticky, 142
storing user state information, 140–142

V

vCPU (virtual CPU), 219
virtual machines, 4. *See also* EC2 (Elastic Compute Cloud) instances
versus containers, 243
virtual private cloud, 5
VPC (virtual private cloud), 10, 19, 56–58, 334–335, 336, 337–338
AZs (availability zones), 353
Blackfoot devices, 342
CIDR block
creating, 349
primary, 349–351
secondary, 351–352
connectivity options, 387
creating
using AWS CLI, 347
using the Launch VPC wizard, 345–347
using the VPC Wizard, 344–345
default, 352–353

determining number of needed, 347–349
EIPs (elastic IP addresses), 364–366
endpoints, 390–391
gateway, 391–392
interface, 392–393
PrivateLink, 393–395
external connections, 339, 395–396
customer gateway, 404–406
Direct Connect, 407–409
egress-only Internet gateway, 398
Internet gateway, 395–396
NAT gateway, 399–401
transit gateway, 401–402
VPG (virtual private gateway), 404
VPN, 401–404
VPN CloudHub, 406
flow logs, 385–386
hosted services, 338
hypervisor, 340–341
IG (Internet Gateway), 352
inbound and outbound traffic charges, 367–368
IPv6 addresses, 370–371
Layer 3 networks, 339
packet flow, 341–342
Blackfoot devices, 342
mapping service, 343–344
peering, 387–390
physical infrastructure, 339–340
private IPv4 addresses, 361–362
public IPv4 addresses, 362–364, 366–367
route tables, 356, 360–361
custom, 357–360
main, 357
security groups, 371–374
administrative access, 376–377
application server inbound ports, 375–376
custom, 374–375
database server inbound ports, 376
ELB (Elastic Load Balancing), 377–378
pinging an EC2 instance, 377
planning, 378–379
subnets, 354–356
VPG (virtual private gateway), 404
VPNs (virtual private networks), 6, 401–404
CloudHub, 406
route propagation, 406–407

W

WAF (Web Application Firewall), 20, 72–73
and CloudFront, 77
Well-Architected Framework, 24
cost optimization, 26
operational excellence, 25
performance efficiency, 25
reliability, 25
security, 25
Well-Architected Framework tool, 26–27
Wiggins, A., 121
Windows AMIs, 236

X-Y-Z

x1 instances, 223
z1d instances, 224