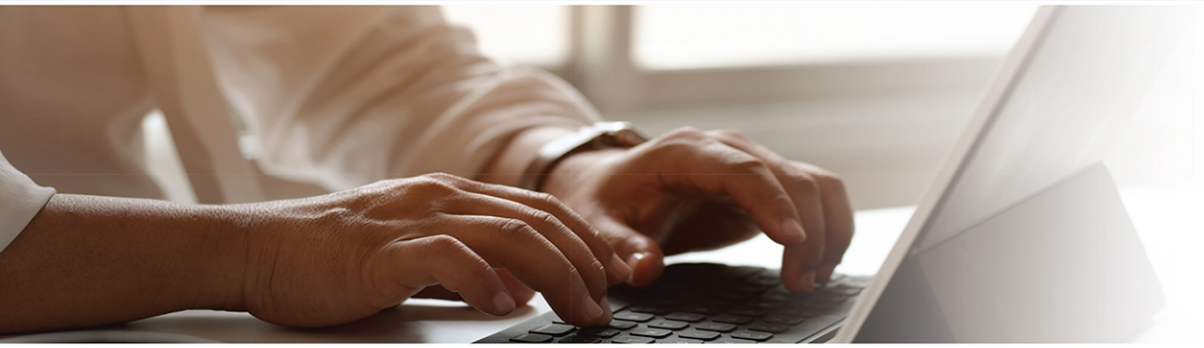

Practice Tests

Flash Cards

Code Examples

Study Planner

Cert Guide

Advance your IT career with hands-on learning

AWS Certified Developer – Associate

(DVA-C01)

PEARSON IT CERTIFICATION

MARKO SLUGA

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

AWS Certified Developer–Associate (DVA-C01)

Cert Guide

MARKO SLUGA

Pearson IT Certification

AWS Certified Developer–Associate (DVA-C01) Cert Guide

ScoutAutomatedPrintCode

Library of Congress Control Number: 2019956930

ISBN-13: 978-0-13-585329-0
ISBN-10: 0-13-585329-X

Trademark Acknowledgments

Editor-in-Chief: Mark Taub

Product Line Manager: Brett Bartow

Acquisitions Editor: Malobika Chakraborty

Managing Editor: Sandra Schroeder

Development Editor: Christopher Cleveland

Senior Project Editor: Lori Lyons

Technical Editor: Anthony Sequeira

Copy Editor: Kitty Wilson

Editorial Assistant: Cindy Teeters

Cover Designer: Chuti Prasertsith

Production Manager: Vaishnavi Venkatesan/ codeMantra

Composition: codeMantra

Indexer: Cheryl Ann Lenser

Proofreader: Charlotte Kughen

Figure Credits

Cover: Mirtmirt/Shutterstock

Figure 1-1, 1-2: National Institute of Standards and Technology special publication © NIST

Figure 1-9: AWS Global infrastructure © 2020 Amazon Web Services, Inc

Figure 1-10: Amazon S3 + Amazon CloudFront: A Match Made in the Cloud, 27 JUN 2018, © 2020 Amazon Web Services, Inc

Figure 1-11 1-12, 1-13: Screenshot of AWS © 2020 Amazon Web Services, Inc

Figure 1-14 through 1-19: Screenshot of AWS Management © 2020 Amazon Web Services, Inc

Figure 1-20, 1-21: Screenshot of AWS CLI © 2020 Amazon Web Services, Inc

Figure 2-7: How to Use SAML to Automatically Direct Federated Users to a Specific AWS Management Console Page by Alessandro Martini © Amazon Web Services, Inc

Figure 2-8: AWS IAM Now Supports Amazon, Facebook, and Google Identity Federation by Jeff Barr © Amazon Web Services, Inc

Figure 3-1: Difference between Public IP and Private IP address © 2020, Difference Between

Figure 3-3: Latency numbers © 2020 GitHub, Inc, https://gist.github.com/2841832

Figure 3-5: Multiple Data Center HA Network Connectivity © 2019, Amazon Web Services, Inc

Figure 3-6: Unsupported VPC Peering Configurations © 2020, Amazon Web Services, Inc

Figure 3-7: Amazon EBS Snapshots © 2020, Amazon Web Services, Inc

Figure 3-9: AWS Elastic Load Balancer © Aman Sardana

Figure 3-10: Scaling Cooldowns for Amazon EC2 Auto Scaling © 2020, Amazon Web Services, Inc

Figure 3-11: High Availability with Route53 DNS Failover © Randika Rathugamage

Figure 3-12 through 3-15: Screenshot of Amazon EC2 © Amazon Web Services, Inc

Figure 3-16: Screenshot of CLI command © Amazon Web Services, Inc

Figure 4-1: Screenshot of Amazon S3 © Amazon Web Services, Inc

Figure 4-2: Indexing Metadata in Amazon Elasticsearch Service Using AWS Lambda and Python © Amazon Web Services, Inc

Figure 4-5: Amazon RDS Multi-AZ Deployments and Read Replicas © 2006-2020 Percona LLC

Figure 4-6: Amazon Aurora DB Clusters © 2020, Amazon Web Services, Inc

Figure 4-9: Choosing the Right DynamoDB Partition Key © 2019, Amazon Web Services, Inc

Figure 4-10: How to use Amazon DynamoDB global tables to power multiregion architectures © 2019, Amazon Web Services, Inc

Figure 4-11: Jeff Barr, 200 Amazon CloudFront Points of Presence © 2020, Amazon Web Services, Inc.

Figure 4-12: CloudFront Events That Can Trigger a Lambda Function ©2020, Amazon Web Services, Inc.

Figure 4-13: Using Field-Level Encryption to Help Protect Sensitive Data © 2020, Amazon Web Services, Inc.

Figure 5-1 through 5-8: Screenshot of AWS Lambda © Amazon Web Services, Inc

Figure 5-10, 5-11: Basic Amazon SQS Architecture © 2020, Amazon Web Services, Inc

Figure 6-2: DevOps in 3 Sentences © DEV Community 2016 - 2020

Figure 6-3: Difference between agile, CI/CD, and DevOps © 2020 Synopsys, Inc

Figure 6-4 through 6-22: Screenshot of AWS Cloud9 © 2019, Amazon Web Services, Inc

Figure 6-23: Screenshot of CodePipeline © 2019, Amazon Web Services, Inc

Figure 7-1 through 7-7: Screenshot of Amazon RDS © 2019, Amazon Web Services, Inc

Figure 7-8 through 7-23: Screenshot of AWS DMS © 2019, Amazon Web Services, Inc

Figure 8-1 through 8-21: Screenshot of Amazon CloudWatch © 2019, Amazon Web Services, Inc

About the Author

Marko Sluga has more than 20 years of experience in IT and has had the benefit of witnessing the rise of cloud computing. Marko has worked on a variety of cloud-related projects, from the early stages of SOC, corporate virtualization, and open-source API offerings to modern, fully automated, intelligent, serverless, and cloud-native solutions. Marko has been working with Amazon Web Services (AWS) since the start of the 2010s and holds three associate, two professional, and three specialty AWS certifications. Marko performs training and advising on cloud technologies and strategies, DevOps, and IT system and process optimization to clients from a wide range of companies, including startups, SMBs, enterprise businesses, and Fortune 500 companies. Marko runs his own cloud training, coaching, and consulting practice under the markocloud.com brand.

About the Technical Reviewer

Anthony Sequeira, CCIE No. 15626, is a seasoned trainer and author regarding various levels and tracks of Cisco, Microsoft, and AWS certifications. Anthony formally began his career in the information technology industry in 1994 with IBM in Tampa, Florida. He quickly formed his own computer consultancy, Computer Solutions, and then discovered his true passion—teaching and writing about information technologies.

Anthony joined Mastering Computers in 1996 and lectured to massive audiences around the world about the latest in computer technologies. Mastering Computers became the revolutionary online training company KnowledgeNet, and Anthony trained there for many years.

Anthony is currently pursuing his second CCIE in the area of Cisco Data Center. Anthony is a full-time instructor at CBT Nuggets.

Dedication

I would like to dedicate this book to my mother, Marta Sluga, who has always put an emphasis on learning being the most important aspect of success.

Acknowledgments

This manuscript was made truly great by the incredible technical review of Anthony Sequeira.

I would also like to express my gratitude to Chris Cleveland, development editor of this book. His dedication made this book several cuts above the rest.

Finally, thanks you so much to Paul Carlstroem for giving me the benefit of the doubt and being patient and understanding with all my ups and downs during the writing process.

Contents at a Glance

Contents

Introduction

I would like to welcome you and extend my gratitude for choosing this publication as your guide on the journey to becoming an AWS Certified Developer. The main purpose of the book is to guide you through the process of learning about Amazon Web Services from the point of view of a developer. The book covers the topics that are listed as required knowledge when preparing for the AWS Certified Developer–Associate exam.

This book also provides examples and code snippets to help you learn how to perform the tasks being described in the book and also gives you the knowledge and tools required to develop applications on the AWS cloud computing environment.

Taking any exam should not be taken lightly. Many experts who rate the IT industry exams have put the AWS exams on the top of the scale as far as difficulty is concerned. Some have gone so far as to claim that AWS sets the bar for everyone in the industry much higher. But don't worry, by reading through this book and following the examples, you should gain valuable knowledge that you can put to use when you decide to take the AWS Certified Developer–Associate exam.

But a book can only go so far, and throughout the book I stress that having hands-on experience with AWS services, tools, and platforms is crucial to being prepared to pass the exam. Think of the learning process as having two parts:

- Gaining theoretical knowledge and practicing (which is what this book is designed to do)
- Getting real-world hands-on experience (which will be helpful as you use AWS on a daily basis)

The Goals of the AWS Certified Developer–Associate Certification

It is important to understand the knowledge requirements for passing the AWS Certified Developer–Associate exam. This will help you lay down a solid foundation of understanding of the concepts that the exam is designed to test and allow you to carve a focused path through the complicated world of AWS. I recommend creating a skills map that you can update as you learn. This part of the book will help you create an initial skills map and help you focus on the parts that matter.

Each AWS certification exam conforms to an exam blueprint. You can use the blueprint as a reference tool to get an overview of which areas of knowledge the exam is designed to test. The AWS Certified Developer–Associate exam blueprint also states that taking and passing the exam will prove:

- Your understanding of core AWS services
- Your understanding of AWS architecture best practices
- Your proficiency in developing, deploying, and debugging cloud-based applications using AWS

Recommended Prerequisite Skills

In the blueprint, AWS also outlines the experience that a candidate attempting the AWS Certified Developer–Associate exam should have. The ability to understand and be able to use AWS services should be complemented by a strong background in development. AWS outlines that each candidate attempting the exam should have experience in the following areas:

- One or more years of hands-on experience developing and maintaining an AWS-based application
- In-depth knowledge of at least one high-level programming language
- Understanding of core AWS services, uses, and basic AWS architecture best practices
- Proficiency in developing, deploying, and debugging cloud-based applications using AWS
- Ability to use the AWS service APIs, AWS CLI, and SDKs to write applications
- Ability to identify key features of AWS services
- Understanding of the AWS shared responsibility model
- Understanding of application life cycle management
- Ability to use a CI/CD pipeline to deploy applications on AWS
- Ability to use or interact with AWS services
- Ability to apply a basic understanding of cloud-native applications to write code
- Ability to write code using AWS security best practices (such as not using secret and access keys in code but instead using IAM roles)
- Ability to author, maintain, and debug code modules on AWS
- Proficiency writing code for serverless applications
- Understanding of the use of containers in the development process

As you see, the list of recommended AWS knowledge is quite extensive and mainly covers real-world experience, which is invaluable in being able to develop on AWS. Of course, this list of requirements is intended for your own assessment. AWS does not require you to prove your experience and does allow you to take the exam even if you do not possess all the knowledge recommendations. The basic rule is that the more recommendations you meet, the more likely you are to pass the exam. This book is designed to provide the theoretical part of the recommendations and to allow you to read and study the concepts at your own pace. But I highly encourage you to gain the required hands-on experience with all of the above before you attempt the exam.

The Domains of Knowledge

The AWS Certified Developer–Associate exam is designed to test the areas of knowledge that are aggregated into five domains of knowledge. The following table lists the breakdown of each of the domains represented on the exam.

Domain	Percentage of Representation on Exam
1: Deployment	22%
2: Security	26%
3: Development with AWS Services	30%
4: Refactoring	10%
5: Monitoring and Troubleshooting	12%
	Total 100%

The sections that follow describe each domain in greater detail.

Domain 1: Deployment

The Deployment domain, as the name indicates, focuses on testing the understanding and knowledge of how to deploy applications on AWS. As a developer, you should have a good understanding of how to deploy applications on AWS using the CLI, the SDK, CI/CD pipelines, and AWS deployment processes and patterns.

The exam will test your ability to implement deployment and provisioning best practices and gauge whether you are able to determine the right solution to use for deploying an application. The exam also focuses on testing your understanding of tools and approaches that allow developers to integrate the deployment of applications into their code.

The Deployment domain involves understanding serverless application design because an increasing number of applications are being deployed in this manner due to the benefits of running serverless.

The exam will test you on the following AWS topics:

- **The CLI and the SDKs:** The exam will evaluate your understanding of what can be achieved with the CLI and SDKs and how these tools are applicable when deploying infrastructure services in AWS.
- **CloudFormation:** The exam will test your understanding of when CloudFormation is applicable, what its general purpose is, and also the ability to read the CloudFormation syntax in JSON or YAML.
- **Elastic Beanstalk:** The exam will test your ability to understand the benefits and advantages of Elastic Beanstalk as well as the limitations of the solution. You should understand the capabilities of Elastic Beanstalk and the most common use cases and should have a good understanding of the Elastic Beanstalk update process and the processes of customizing a deployment.
- **CodeDeploy and CodePipeline:** The exam will ask questions focused on determining the understanding of the deployment part of a typical CI/CD pipeline as per the best practices outlined by AWS.
- **AWS Lambda:** The exam will evaluate whether you understand the AWS Lambda deployment procedure and how it can integrate with other AWS services to provide a supporting role during a deployment.

- **Static websites on S3:** In some cases, the exam will evaluate whether you understand when a static website on S3 is the right type of deployment option for the outlined case.

Domain 2: Security

Possibly the most important aspect of any application is security. The Security domain is designed to make sure you understand how to design, develop, and deploy applications on AWS with security in mind. Among the most important aspects tested on the exam is the understanding of authentication and authorization, with a focus on calls to the AWS infrastructure as well as the security of the application running on top of AWS.

The exam will test you on the following AWS topics:

- **Types of credentials:** You must understand the different types of credentials that can be used to access AWS and the services running on AWS, including the username and password, access key and secret key, key pairs, and multifactor authentication (MFA).
- **The CLI and the SDKs:** The exam will evaluate your understanding of authentication practices that should be observed when using the AWS CLI and the AWS SDKs.
- **IAM:** The exam will test your understanding of the practices associated with managing users, groups, and roles; assigning policies; and granting access via the least privilege approach.
- **IAM federation:** The exam will give special focus to evaluating your understanding of how to federate authentication and authorization with external directories and identity providers.
- **Security groups and NACLs:** The exam will focus on understanding how to secure an application over the network; thus basic understanding of the way security groups and NACLs operate in VPC is required.

Domain 3: Development with AWS Services

As this exam tests your development skills, a big portion of it (about a third) is dedicated to the development aspects. The exam will test your knowledge of how to implement application designs into code for applications running on servers and serverless platforms, and it will also test your knowledge of how to interact with the AWS infrastructure through the AWS CLI, the SDKs, and the APIs.

The exam will test you on the following AWS topics:

- **AWS CLI and the SDKs:** The exam will test your ability to use the SDKs and the CLI to interact with the AWS services and deliver application components straight out of the code. Some focus is given to the ability to understand the command structure and identify the correct command. The exam will also include questions that test your general understanding of the capabilities of the CLI and SDKs.
- **DevOps and Code* tools:** The exam will focus on your ability to understand the DevOps approach to development and identify the functionality of the CodeCommit, CodeDeploy, CodeBuild, and CodePipeline tools.

Domain 4: Refactoring

Many enterprises are in the midst of a cloud adoption process, and therefore, the exam will test your ability to understand which AWS services and features will best suit your application and how to migrate existing applications and application code to AWS. The exam will test you on the following AWS topics:

- **AWS migration tools:** The exam will test your basic understanding of what AWS migration tools can be used to transfer (VPN, DirectConnect), transport (Snowball/Snowmobile), or transform (AWS DMS) the data from on-premises systems to AWS.
- **Managed AWS services:** The exam will test your understanding of which managed services can be used to refactor an application that is being implemented on or migrated to AWS.

Domain 5: Monitoring and Troubleshooting

The Monitoring and Troubleshooting domain is designed to test your ability to write code that can be integrated with the AWS monitoring and logging tools. In addition, the exam will test your ability to analyze the environment by looking at the performance information and logs captured by these tools.

The exam will test you on the following AWS topics:

- **CloudWatch:** The exam will evaluate your ability to capture performance data and logs to CloudWatch. Further, it will test your ability to use and analyze the captured data to perform troubleshooting, scaling, and optimization on the application being monitored. You should also have a clear understanding of the features and limitations of CloudWatch, CloudWatch Logs, and CloudWatch Alarms.
- **CloudTrail:** The exam will test your ability to trace the actions in the environment and provide an audit-compliant log of events and actions in the AWS account.
- **General troubleshooting:** The exam will include questions on general troubleshooting of applications, such as steps to be taken when an application is experiencing unusual behavior. Although some might say that this is beyond the scope of AWS, being able to understand application behavior and general troubleshooting steps is required.

Where the Domains/Objectives Are Covered in the Book

The following table presents the domain objectives listed in the exam blueprint and where they are covered in this book:

Domain/Objective	Chapter(s) Where This Is Covered
Domain 1: Deployment	
1.1 Deploy written code in AWS using existing CI/CD pipelines, processes, and patterns.	Chapters 3 and 6
1.2 Deploy applications using Elastic Beanstalk.	Chapter 3
1.3 Prepare the application deployment package to be deployed to AWS.	Chapters 3 and 5
1.4 Deploy serverless applications.	Chapters 3 and 5

Domain/Objective	Chapter(s) Where This Is Covered
Domain 2: Security	
2.1 Make authenticated calls to AWS services.	Chapters 1 and 2
2.2 Implement encryption using AWS services.	Chapter 2
2.3 Implement application authentication and authorization.	Chapter 2
Domain 3: Development with AWS Services	
3.1 Write code for serverless applications.	Chapters 3, 4, and 5
3.2 Translate functional requirements into application design.	Chapters 1, 3, 4, and 5
3.3 Implement application design into application code.	Chapters 3, 4, and 5
3.4 Write code that interacts with AWS services by using APIs, SDKs, and AWS CLI.	Chapters 1, 3, 4, and 5
Domain 4: Refactoring	
4.1 Optimize application to best use AWS services and features.	Chapters 1, 4, 5, and 7
4.2 Migrate existing application code to run on AWS.	Chapters 5 and 7
Domain 5: Monitoring and Troubleshooting	
5.1 Write code that can be monitored.	Chapter 8
5.2 Perform root cause analysis on faults found in testing or production.	Chapter 8

Facts About the Exam

The exam, which needs to be taken at an authorized exam proctor location, consists of 70 to 80 multiple-choice and multiple-answer questions and is available in English, Japanese, and Simplified Chinese. The allotted time for the exam is 130 minutes. The registration

fee for the exam is US$150. If you would like to try your knowledge before you take the actual exam, you can take an online practice exam consisting of 20 questions at any time. The registration fee for the practice exam is US $20.

Taking the practice exam is a good idea if you would like to get a feel for the exam with sample questions that come out of the same pool of questions as the actual exam. The practice exam can be a great tool to help you gauge your knowledge and determine whether you are ready to pass the exam. However, passing the practice exam does not guarantee that you will pass the real exam.

Exam Questions

The questions that will be testing your knowledge on the exam carry different weights. Each question has a certain score assigned to it, and the scores of all the questions together will add up 1000. AWS will be scoring you according to a percentage out of 1000 points rather than based on the number of questions you get correct. All the exam questions are always scored in full. This means an incorrect or missing answer for a multiple-response question will cause the score to be determined as 0. Make sure to take your time and carefully read each question as some of the questions might be lengthy and will be hiding crucial information that would easily help you determine the right answer.

Passing Score

The passing score of each exam is never fully determined, and the score to shoot for is not released publicly; however, the typical passing score for this exam is 720 points. AWS uses statistical analysis of multiple metrics to determine the passing score. This means that you should be as prepared as possible to pass the exam. But that can be difficult to do, so I recommend setting a certain "confidence level" for yourself. This confidence level can be determined by looking at the requirements and the content of this book and taking practice exams like the ones provided in the Pearson Test Prep software for this book. I like to set the confidence level of the content at 90%, meaning you should be able to answer most of the questions you encounter on a certain topic.

Gauging Your Confidence Level

Use the exams in the Pearson Test Prep software to gauge your confidence level. With a 90% content confidence level, you should be able to answer at least 80% of the questions correctly. I urge you not to attempt the exam before you reach this confidence level.

Keep in mind that AWS sources a lot of the exam question content from the FAQs for each service, so another way to prepare for the exam is to read the FAQs and try to answer them yourself. If you can get to this level, then you should be able to pass the exam. The idea behind this is that an AWS certified developer should be able to do any task outlined in the FAQs by heart. A certified developer would certainly still need to consult the documentation and contact AWS support if needed.

Taking the Exam

Once you have determined that your confidence level is high enough to take the exam, make sure to follow a few simple rules that will help you relax and pass the exam with

ease. Make sure you have enough time to get to the exam location. I usually plan to be at the proctor about 30 minutes early, which helps me deal with any kind of delays on the way. Try to clear your calendar before you take the exam; you've been studying for the exam for quite a while so don't try to cram your exam into an already packed day. Any additional stress might prohibit you from relaxing when taking the exam, and being relaxed during the exam is very important.

What to Bring Along?

You will be required to carry two forms of ID because the proctor will need to verify your identity and sign you in to the exam. You will be given a secure locker to store your essentials; you are not allowed to bring anything into the exam room. The lockers at some locations are quite small and will not fit a laptop bag. I therefore recommend that you make sure you really do take only your essentials to the exam. You will then be taken to the exam station, where you will verify your information and have to agree to the terms and conditions onscreen before you start the exam. This is a good time to take a deep breath and relax, release the stress of traffic on the way, and be ready to start the exam.

Ready, Set, Go!

Once you start the exam, the timer will start ticking down from the allotted time. Make sure to read each question and all the answers carefully. If you are unsure of an answer, try not to spend too much time thinking about it but move on and try to answer other questions. You can mark each question you are unsure of for review, and I highly recommend doing that and continuing on. You will probably be able to answer all the easier questions and get to the end of the exam before your time expires. Use the time left over to focus on the questions that you marked for review. This way, you are certain to get the highest number of questions answered. Remember that any unanswered questions will not be scored, meaning you can lose a lot of points due to that fact.

Submitting the Exam

Once you are happy with your answers, you can submit your exam. A quick survey is then presented, asking a handful of questions. After answering the survey, you are presented with the result of the exam—either pass or a fail with a percentages score. It might take some time for the exam to be recorded; after AWS receives your results, you will receive an email with a breakdown of scores across the domains. If you fail the exam, this breakdown will help you better prepare for the next time.

Keep calm and good luck!

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:** These are the core sections of each chapter. They explain the concepts for the topics in that chapter.
- **Exam Preparation Tasks:** This section provides 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 their page numbers. Although the contents of the entire chapter could be 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 Developer–Associate exam may be unlikely to ask a question such as "Define this term," the exam does require that you learn and know a lot of AWS-related 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 just covered by answering these questions and reading the answer explanations.
- **Web-based practice exam:** The companion website includes the Pearson Test Prep application, which allows 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 eight core chapters—Chapters 1 through 8. Chapter 9 provides preparation tips and suggestions for how to approach the exam. Each core chapter covers a subset of the topics and technologies that you will encounter on the AWS Certified Developer–Associate (DVA-C01) exam.

How to Access the Pearson Test Prep (PTP) App

You have two options for installing and using the Pearson Test Prep application: a web app and a desktop app. To use the Pearson Test Prep application, start by finding the registration code that comes with the book. You can find the code in these ways:

- **Print book:** Look in the cardboard sleeve in the back of the book for a piece of paper with your book's unique PTP code.
- **Premium Edition:** If you purchase the Premium Edition eBook and Practice Test directly from the Pearson IT Certification website, the code will be populated on your account page after purchase. Just log in at www.pearsonITcertification.com, click **account** to see details of your account, and click the **digital purchases** tab.
- **Amazon Kindle:** For those who purchase a Kindle edition from Amazon, the access code will be supplied directly from Amazon.
- **Other bookseller e-books:** Note that if you purchase an e-book version from any other source, the practice test is not included because other vendors to date have not chosen to vend the required unique access code.

NOTE Do not lose the activation code because it is the only means by which you can access the online content for the book.

NOTE Amazon eBook (Kindle) customers: It is easy to miss Amazon's email that lists your PTP access code. Soon after you purchase the Kindle eBook, Amazon should send an email. However, the email uses very generic text and makes no specific mention of PTP or practice exams. To find your code, read every email from Amazon after you purchase the book. Also do the usual checks for ensuring that your email arrives, such as checking your spam folder. If you have trouble getting an access code from Amazon, contact Pearson's tech support at http://pearsonitp.echelp.org.

NOTE Other eBook customers: As of the time of publication, only the publisher and Amazon supply PTP access codes when you purchase their eBook editions of this book.

Customizing Your Exams

In the exam settings screen of the Pearson Test Prep Software, you can choose to take exams in one of three modes:

- **Study mode:** Allows you to fully customize your exams and review answers as you are taking the exam. This is typically the mode you use first to assess your knowledge and identify information gaps.
- **Practice Exam mode:** Locks certain customization options to present a realistic exam experience. Use this mode when you are preparing to test your exam readiness.
- **Flash Card mode:** Strips out the answers and presents you with only the question stem. This mode is great for late-stage preparation when you really want to challenge yourself to provide answers without the benefit of seeing multiple-choice options. This mode does not provide the detailed score reports that the other two modes provide, so it is not the best mode for helping you identify knowledge gaps.

In addition to these three modes, you will be able to select the source of your questions. You can choose to take exams that cover all of the chapters, or you can narrow your selection to just a single chapter or the chapters that make up specific parts in the book. All chapters are selected by default. If you want to narrow your focus to individual chapters, simply deselect all the chapters and then select only those on which you wish to focus in the Objectives area.

You can also select the exam banks on which to focus. Each exam bank comes complete with a full exam of questions that cover topics in every chapter. The two exams included online with the purchase of this book are available to you, as are two additional exams of unique questions available with the Premium Edition. You can have the test engine serve up exams from all four banks or just from one individual bank by selecting the desired banks in the exam bank area.

You can make several other customizations to your exam from the exam settings screen, such as the time allotted to take the exam, the number of questions served up, whether

to randomize questions and answers, whether to show the number of correct answers for multiple-answer questions, and whether to serve up only specific types of questions. You can also create custom test banks by selecting only questions that you have marked or questions on which you have added notes.

Updating Your Exams

If you are using the online version of the Pearson Test Prep software, you should always have access to the latest version of the software as well as the exam data. If you are using the Windows desktop version, every time you launch the software while connected to the Internet, it checks whether there are any updates to your exam data and automatically downloads any changes made since the last time you used the software.

Sometimes, due to a number of factors, the exam data may not fully download when you activate your exam. If you find that figures or exhibits are missing, you may need to manually update your exams. To update a particular exam you have already activated and downloaded, simply select the **Tools** tab and click the **Update Products** button. Again, this is only an issue with the desktop Windows application.

If you want to check for updates to the Windows desktop version of the Pearson Test Prep exam engine software, simply select the **Tools** tab and click the **Update Application** button. Doing so allows you to ensure that you are running the latest version of the software engine.

CHAPTER 4

Storing Data in AWS

This chapter covers the following subjects:

Storing Static Assets in AWS: One of the key aspects of cloud computing is the ability to consume a virtually unlimited amount of resources. The first part of this chapter covers how to use the S3 and Glacier services to store and deliver unlimited amounts of static data in AWS.

Relational Versus Nonrelational Databases: To prepare you for the next two chapters, this section provides a short overview of the differences between relational and nonrelational databases and data types suitable for each database type.

Deploying Relational Databases in AWS: This section examines the deployment of relational databases using AWS Relational Database Service (RDS).

Handling Nonrelational Data in AWS: Some datasets are just not suitable for relational databases. When sustained and predictable performance for relatively simple datasets is required, you can use the DynamoDB service in AWS. This section examines the characteristics of DynamoDB and shows how to use DynamoDB in your applications.

Caching Data in AWS: The last part of this chapter covers the different options for caching data and accelerating the delivery of content from the storage systems covered in this chapter.

This chapter covers content important to the following exam domains:

Domain 3: Development with AWS Services

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

Domain 4: Refactoring

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

The challenge of maintaining and storing data in the most efficient manner has been plaguing enterprises for decades. There is never enough storage, and storage performance is quite often a factor in poor application performance. Moreover, storing data securely and preventing disastrous consequences of losing data can be a huge challenge. As the old saying goes, "If your data is not stored in three places at once, it does not exist persistently."

A typical enterprise might make tremendous investments in data storage hardware, storage area networks, storage management software, replication, snapshots, backup software, virtual tape libraries, and all kinds of different solutions for storing different data types on different tiers, only to find itself needing to make more hefty investments a year later. I have personally witnessed millions of dollars being spent on data storage solutions with little effect on the final outcome over the long term. It seems the storage industry has no need to plan obsolescence of their products as storage is the only resource in computing that will keep growing and growing.

So what is the solution? Well, it's mostly about selecting the right storage back end for the right type of data. It is not possible to solve a data crisis with a one-size-fits-all service; rather, you need to take a multipronged approach including classifying your data, deciding which data is suitable for the cloud, and selecting the right type of cloud solution for storing that data. Some data might be bound by compliance, confidentiality, or governance that therefore might need to stay on premises, but for most other data, a much more cost-effective way is to store it in the cloud. AWS offers several different services for storing your data, and this chapter takes a look at each of them.

"Do I Know This Already?" Quiz

The "Do I Know This Already?" quiz allows you to assess whether you should read the entire chapter. Table 4-1 lists the major headings in this chapter and the "Do I Know This Already?" quiz questions covering the material in those headings so you can assess your knowledge of these specific areas. The answers to the "Do I Know This Already?" quiz appear in Appendix A, "Answers to the 'Do I Know This Already?' Quizzes and Q&A Sections."

Table 4-1 "Do I Know This Already?" Foundation Topics Section-to-Question Mapping

Foundations Topics Section	Questions
Storing Static Data in AWS	1, 2, 5, 10, 11
Deploying Relational Databases in AWS	3, 6, 7
Handling Nonrelational Data in AWS	4, 8, 12
Caching Data in AWS	9, 13

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. You are asked to provide an HTTP-addressable data store that will have the ability to serve a static website. Which data back end would be the most suitable to complete this task?
   - **a.** DynamoDB
   - **b.** EBS
   - **c.** Glacier
   - **d.** S3

2. **Complete this sentence:** The S3 service allows for storing an unlimited amount of data as long as individual files are not larger than _____ and any individual PUT commands do not exceed _____.
   a. 5 GB; 5 MB
   b. 5 GB; 5 GB
   c. 5 TB; 5 GB
   d. 5 TB; 5 MB
3. Which of these databases is not supported by RDS?
   a. Cassandra
   b. Microsoft SQL
   c. Oracle
   d. MariaDB
4. To determine the number of read capacity units required for your data, what do you need to consider ?
   a. Whether reads are performed in the correct sequence
   b. Whether reads are strongly or eventually consistent
   c. Whether reads are coming from one or multiple sources
   d. All of these answers are correct.
5. Which of the following is not an S3 service tier?
   a. S3 Standard
   b. S3 Accelerated Access
   c. S3 Infrequent Access
   d. S3 Reduced Redundancy Store
6. RDS has the ability to deliver a synchronous replica in another availability zone in which mode?
   a. Multi-AZ mode
   b. High-availability mode
   c. Cross-AZ mode
   d. Master-slave mode
7. Your company is implementing a business intelligence (BI) platform that needs to retain end-of-month datasets for analytical purposes. You have been asked to create a script that will be able to create a monthly record of your complete database that can be used for analytics purposes only if required. What would be the easiest way of doing this?
   a. In RDS, choose to create an automated backup procedure that will create a database snapshot every month. The snapshot can be restored to a working database if required by the BI software.
   b. Write a script that will run on a predetermined day and hour of the month and snapshot the RDS database. The snapshot can be restored to a working database if required by the BI software.

c. Write a script that will offload all the monthly data from the database into S3. The data in S3 can be imported into a working database if required by the BI software.

d. In RDS, choose to create an automated export procedure that will offload all the monthly data from the database into S3. The data in S3 can be imported into a working database if required by the BI software.

8. If your application has unknown and very spiky read and write performance characteristics, which of the following should you consider choosing?
   a. Using a NoSQL solution such as Memcached
   b. Auto-scaling the DynamoDB capacity
   c. Distributing data across multiple DynamoDB tables
   d. Using the on-demand model for DynamoDB
9. Which service would you select to accelerate the delivery of video files?
   a. S3 Accelerated Access
   b. ElastiCache
   c. CloudCache
   d. CloudFront
10. When uploading files to S3, it is recommended to do which of the following? (Choose all that apply.)
    a. Split files 100 MB in size to multipart upload them to increase performance
    b. Use a WAN accelerator to increase performance
    c. Add metadata when initiating the upload
    d. Use a VPN connection to increase security
    e. Use the S3 HTTPS front end to increase security
    c. Add metadata after the upload has completed
11. Which of these data stores would offer be the least expensive way to store millions of log files that are kept for retention purposes?
    a. DynamoDB
    b. EBS
    c. Glacier
    d. S3
12. DynamoDB reads are performed via:
    a. HTTP NoSQL requests to the DynamoDB API.
    b. HTTP HEAD requests to the DynamoDB API.
    c. HTTP PUT requests to the DynamoDB API.
    d. HTTP GET requests to the DynamoDB API.
13. Which ElastiCache engine can support Multi-AZ deployments?
    a. Redis
    b. Memcached
    c. DAX
    d. All of these answers are correct.

Foundation Topics

Depending on the way you deliver content, you can classify your data in to three major categories:

- **Static assets:** Any type of content that cannot be opened from the storage environment directly (via block access) but that must rather be transferred locally (downloaded) and only then opened can be considered a static asset. These assets can be any types of files, such as text, videos, images, archives, packages, and other data blobs that reside on a web server and are accessed only via the web service. Because static assets are delivered across the network, the access times for data ranges from tens of milliseconds to seconds to minutes and even to hours (for very large files over slow links).
- **Dynamic assets:** These assets are any type of content that is opened and used from the storage environment directly via block-level access. These can be any types of files that are consumed by services to maintain data records or state, such as databases, log files, and executable files. Usually dynamic assets are opened by a certain process and start writing the records. A dynamic asset is not accessible to other processes on the file system and is accessible only through a service such as a database service or an API. Because static assets can be accessed directly through block access, you usually see the latencies being measured in a few milliseconds to a few seconds.
- **In-memory assets:** An in-memory asset is any type of content that is loaded into memory and used by one or multiple processes within a server directly via access to the memory. Most commonly these assets are any kind of block-level data that is cached for performance, in-memory databases, and other data that needs to be accessed with the lowest possible latency. As the cost of memory per gigabyte is much higher than the cost of disks, it is common to store only "hot data" or caches in memory to increase the performance of traditional disk-based systems. Because in-memory assets are ready to serve, the latencies for delivering data can be decreased down to microseconds.

Storing Static Assets in AWS

To identify static assets, you can simply scan the file system of your application and look at all the files that have not been changed since they were created. If the creation time matches the last time the file was modified, you can be certain the asset is a static piece of data. In addition, any files being delivered via an HTTP, FTP, SCP, or other type of service can fall into the category of static assets because these files are most likely not being consumed on the server but are rather being consumed by a client connecting to the server through one of these protocols. Once you have identified your static assets, you need to choose the right type of data store. It needs to be able to scale with your data and needs to do that in an efficient, cost-effective way. In AWS, Simple Storage Service (S3) is used to store any kind of data blobs on an object storage back end with unlimited capacity.

Amazon S3

Amazon S3 is essentially a serverless storage back end that is accessible via HTTP/HTTPS. The service is fully managed and has a 99.99% high availability SLA per region and a 99.999999999% SLA for durability of data. The 99.99% high availability means you can expect to have less than 45 minutes of service outage per region during a monthly billing

cycle, and the 99.999999999% durability means the probability of losing a file is equal to 1 in 10,000,000 per every 10,000 years.

S3 delivers all content through the use of content containers called *buckets*. Each bucket serves as a unique endpoint where files and objects can be aggregated (see Figure 4-1). Each file you upload to S3 is called a *key*; this is the unique identifier of the file within the S3 bucket. A key can be composed of the filename and prefixes. Prefixes can be used to structure the files even further and to provide a directory-like view of the files, as S3 has no concept of directories.

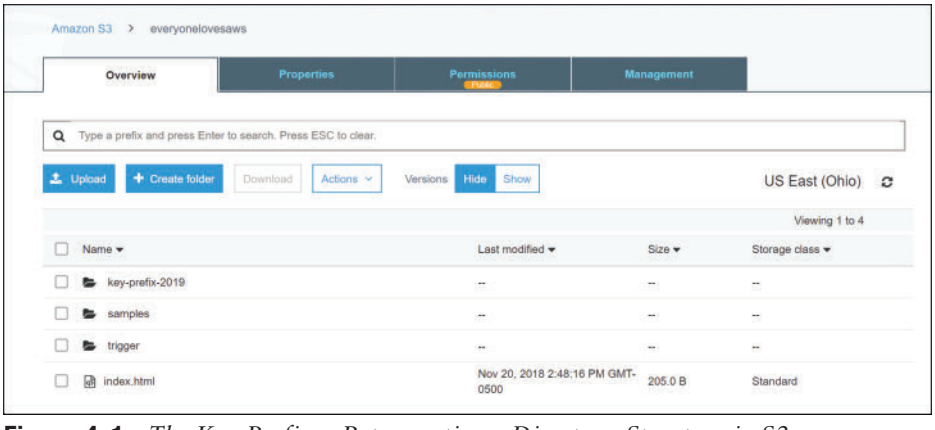

Figure 4-1 *The Key Prefixes Representing a Directory Structure in S3*

With S3 you can control access permissions at the bucket level, and thus you can define the level of access to the bucket itself. You can essentially make a bucket completely public by allowing anonymous access, or you can strictly control the access to each key in the bucket. There are two different ways of allowing access to an S3 bucket:

- **Use an access control list (ACL) or a bucket ACL:** With the ACL approach, you can control the permissions on a broader spectrum than by using the bucket policy. This approach is designed to quickly allow access to a large group of users, such as another account or everyone with a specific type of access to all the keys in the bucket; for example, an ACL can easily be used to define that everyone can list the contents of the bucket.
- **Use a bucket policy:** A bucket policy is a JSON-formatted document that is structured exactly like an IAM policy. A bucket policy can be used to granularly control access to a bucket and its contents. For example, by using a bucket policy, you can allow a specific user to access only a specific key within a bucket.

Delivering Content from S3

The S3 service is very easy to use when developing applications because it is addressable via standard HTTP method calls. And because the service delivers files through a standard HTTP web interface, it is well suited for storing any kind of static website content, sharing files, hosting package repositories, and even hosting a static website that can have extended app-like functionality with client-side scripting. Developers are also able to use the built-in change notification system in S3 to send messages about file changes and allow for processing with other AWS services, such as AWS Lambda, which can pick up any file

coming onto S3 and perform transformations, record metadata, and so on so that the static website functionality can be greatly enhanced. Figure 4-2 illustrates how a file being stored on S3 can trigger a dynamic action on AWS Lambda.

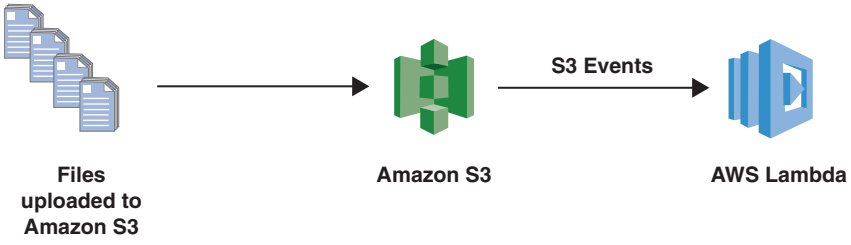

Figure 4-2 *S3 Events Triggering the Lambda Service*

Because S3 is basically an API that you can communicate with, you can simply consider it programmatically accessible storage. By integrating your application with S3 API calls, you can greatly enhance the capability of ingesting data and enhancing raw storage services with different application-level capabilities. S3's developer-friendly features have made it the gold standard for object storage and content delivery.

Key Topic

Working with S3 in the AWS CLI

To create a bucket, you can simply use the **aws s3api create-bucket** command:

```
aws s3api create-bucket --bucket bucket-name --region region-id
```

Say that you want to create a bucket called **everyonelovesaws**. If you are following along with this book, you will have to select a different name because AWS bucket names are global, and the **everyonelovesaws** bucket already exists for the purpose of demonstrating FQDNs. To create the bucket, simply replace *bucket-name* and set your desired region:

```
aws s3api create-bucket --bucket everyonelovesaws --region us-east-2
```

After the bucket is created, you can upload an object to it. You can upload any arbitrary file, but in this example, you can upload the index file that will later be used for the static website:

```
aws s3 cp index.html s3://everyonelovesaws/
```

This simply uploads this one file to the root of the bucket. To do a bit more magic, you can choose to upload a complete directory, such as your website directory:

```
aws s3 cp /my-website/ s3://everyonelovesaws/ --recursive
```

You might also decide to include only certain files by using the **--exclude** and **--include** switches. For example, when you update your website HTML, you might want to update all the HTML files but omit any other files, such as images, videos, CSS, and so on. You might need to use multiple commands and search for all the HTML files. To do all this with one command, you can simply run the following:

```
aws s3 cp /my-website/ s3://everyonelovesaws/ --recursive
--exclude "*" --include "*.html"
```

By excluding everything (*) and including only *.html, you ensure that all HTML files get uploaded while all the content that hasn't changed is not touched.

When accessing content within a bucket on S3, there are three different URLs that you can use. The first (default) URL is structured as follows:

`http{s}://s3.{region-id}.amazonaws.com/{bucket-name}/{optional key prefix}/{key-name}`

As you can see, the default naming schema makes it easy to understand: First you see the region the bucket resides in (from the region ID in the URL). Then you see the structure defined in the bucket/key-prefix/key combination.

Here are some examples of files in S3 buckets:

- A file in the root of the everyonelovesaws bucket: https://s3.us-east-2.amazonaws.com/everyonelovesaws/index.html
- A file with the key prefix key-prefix-2019/ in the everyonelovesaws bucket: https://s3.us-east-2.amazonaws.com/everyonelovesaws/key-prefix-2019/index.html
- A file with the key prefix key-prefix-2019/08/20 in the everyonelovesaws bucket: https://s3.us-east-2.amazonaws.com/everyonelovesaws/key-prefix-2019/08/20/index.html

However, the default format might not be the most desirable, especially if you want to represent the S3 data as being part of your website. For example, suppose you want to host all your images on your S3 website, and you would like to redirect the subdomain images.mywebsite.com to an S3 bucket. The first thing to do would be to create a bucket with that exact name images.mywebsite.com in it so you can create a CNAME in your domain and not break the S3 request.

To create a CNAME, you can use the second type of FQDN in your URL that is provided for each bucket, with the following format:

`{bucket-name}.s3.{optional region-id}.amazonaws.com`

As you can see, the regional ID is optional, and the bucket name is a subdomain of s3.amazonaws.com, so it is easy to create a CNAME in your DNS service to redirect a subdomain to the S3 bucket. For the image redirection, based on the preceding syntax, you would simply create a record like this:

```
images.mywebsite.com    CNAME    images.mywebsite.com.
s3.amazonaws.com.
```

If you want to disclose the region ID, you can optionally create an entry with the region ID in the target name.

NOTE Bucket names are globally unique. Because every bucket name is essentially a subdomain of **.s3.amazonaws.com**, there is no way to make two buckets with the same name in all of AWS.

Here are some working examples of a bucket called images.markocloud.com that is a subdomain of the markocloud.com domain:

- FQDN with the region ID is serving the index.html key this way: http://images.markocloud.com.s3.us-east-1.amazonaws.com/index.html

- FQDN without the region ID is serving the index.html key this way: http://images.markocloud.com.s3.amazonaws.com/index.html
- FQDN with the CNAME on the markocloud.com domain is serving the index.html key this way: http://images.markocloud.com/index.html

Key Topic

Hosting a Static Website

S3 is a file delivery service that works on HTTP/HTTPS. To host a static website, you simply need to make the bucket public by providing a bucket policy and enabling the static website hosting option. Of course, you also need to upload all your static website files, including an index file.

To make a bucket serve a static website from the AWS CLI, you need to run the **aws s3 website** command using the following syntax:

```
aws s3 website s3://{bucket-name}/ --index-document {index-
document-key} --error-document {optional-error-document-key}
```

To make the everyonelovesaws bucket into a static website, for example, you would simply enter the following:

```
aws s3 website s3://everyonelovesaws/ --index-document index.html
```

You now also need to apply a bucket policy to make the website accessible from the outside world. If you are creating your own static website, you need to replace *everyonelovesaws* in the resource ARN (**"Resource": "arn:aws:s3:::***everyonelovesaws***/***-) with your bucket name, as demonstrated in Example 4-1.

Key Topic

Example 4-1 *An IAM Statement That Allows Read Access to All Items in a Specific S3 Bucket*

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PublicReadGetObject",
            "Effect": "Allow",
            "Principal": "*",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::everyonelovesaws/*"
        }
    ]
}
```

As you can see, the policy is allowing access from anywhere and performing the s3:GetObject function, which means it is allowing everyone to read the content but not allowing the listing or reading of the file metadata.

You can save this bucket policy as **eveyonelovesaws.json** and apply it to the bucket with the following command:

```
aws s3api put-bucket-policy --bucket everyonelovesaws --policy
file:// eveyonelovesaws.json
```

NOTE You have seen two different S3 CLI commands: **aws s3** and **aws s3api**. The **s3** command is designed to make it easier to work with files and objects in S3, whereas **s3api** follows the API model precisely and is purely JSON driven. You can more granularly control making the bucket public by using the **s3api put-bucket-website** command, but this command requires a JSON configuration file as input, so instead you can use the **s3** command. The **s3** command provides a bit more abstraction and achieves the same result with a simpler, shorter command.

When the static website is enabled, you are provided with a URL that looks like this:

http://everyonelovesaws.s3-website.us-east-2.amazonaws.com/

Note in this example as well as in the example of the CNAMED images bucket, that the HTTP URL is not secure. This is due to the fact that there is a limitation to bucket names containing dots when using HTTPS. The default S3 certificate used for signing is *.s3.amazonaws.com. This certificate can only sign the first subdomain of .s3.amazonaws.com. Any dot in the name will be represented as a further subdomain, which would break the certificate chain. Therefore, going to the following site will show an insecure warning:

https://images.markocloud.com.s3.amazonaws.com/index.html

This is due to the fact that the *.s3.amazonaws.com certificate only signs the "com.s3.amazonaws.com" portion of the domain name and going to the following site will now show an insecure warning since the * certificate does not sign the DNS name for the "images.markocloud." part of the domain:

https://everyonelovesaws.s3.amazonaws.com/index.html

For hosted websites, you can, of course, have dots in the name of the bucket. However, if you tried to add an HTTPS CloudFront distribution and point it to such a bucket, you would break the certificate functionality by introducing a domain-like structure to the name. Nonetheless, all static websites on S3 would still be available on HTTP directly even if there were dots in the name. The final part of this chapter discusses securing a static website through HTTPS with a free certificate attached to a CloudFront distribution.

Versioning

S3 provides the ability to create a new version of an object if it is uploaded more than once. For each key, a separate entry is created, and a separate copy of the file exists on S3. This means you can always access each version of the file and also prevent the file from being deleted because a deletion will only mark the file as deleted and will retain the specific previous versions.

To enable versioning on your bucket, you can use the following command:

```
aws s3api put-bucket-versioning --bucket everyonelovesaws
--versioning-configuration Status=Enabled
```

There are three status options: Disabled, Enabled, and Suspended. By default, a bucket has versioning disabled, but once it is enabled, it cannot be removed but only suspended. When versioning is suspended, new versions of the document are not created; rather, the newest version is overwritten, and the older versions are retained.

S3 Storage Tiers

When creating an object in a bucket, you can also select the storage class to which the object will belong. This can also be done automatically through data life cycling. S3 has six storage classes:

- **S3 Standard:** General-purpose online storage with 99.99% availability and 99.999999999% durability (that is, "11 9s").
- **S3 Infrequent Access:** Same performance as S3 Standard but up to 40% cheaper with 99.9% availability SLA and the same "11 9s" durability.
- **S3 One Zone-Infrequent Access:** A cheaper data tier in only one availability zone that can deliver an additional 25% savings over S3 Infrequent Access. It has the same durability, with 99.5% availability.
- **S3 Reduced Redundancy Storage (RRS):** Previously this was a cheaper version of S3 providing 99.99% durability and 99.99% availability of objects. RRS cannot be used in a life cycling policy and is now more expensive than S3 Standard.
- **S3 Glacier:** Less than one-fifth the price of S3 Standard, designed for archiving and long-term storage.
- **S3 Glacier Deep Archive:** Costs four times less than Glacier and is the cheapest storage solution, at about $1 per terabyte per month. This solution is intended for very long-term storage.

NOTE Due to the reduction in price of S3 Standard over time and low interest in using RRS, RRS is now more expensive. However, at press time, there is no official plan to sunset the RRS tier, which is still being used to temporarily restore data from Glacier and Glacier Deep Archive.

Data Life Cycling

S3 supports automatic life cycling and expiration of objects in an S3 bucket. You can create rules to life cycle objects older than a certain time into cheaper storage. For example, you can set up a policy that will store any object older than 30 days on S3 Infrequent Access (S3 IA). You can add additional stages to move the object from S3 IA to S3 One Zone IA after 90 days and then push it out to Glacier after a year, when the object is no longer required to be online. Figure 4-3 illustrates S3 life cycling.

Figure 4-3 *Illustration of an S3 Life Cycling Policy*

S3 Security

When storing data in the S3 service, you need to consider the security of the data. First, you need to ensure proper access control to the buckets themselves. There are three ways to grant access to an S3 bucket:

- **IAM policy:** You can attach IAM policies to users, groups, or roles to allow granular control over different levels of access (such as types of S3 API actions, like GET, PUT, or LIST) for one or more S3 buckets.
- **Bucket policy:** Attached to the bucket itself as an inline policy, a bucket policy can allow granular control over different levels of access (such as types of S3 API actions, like GET, PUT, or LIST) for the bucket itself.
- **Bucket ACL:** Attached to the bucket, an access control list (ACL) allows coarse-grained control over bucket access. ACLs are designed to easily share a bucket with a large group or anonymously when a need for read, write, or full control permissions over the bucket arises.

Both policy types allow for much better control over access to a bucket than does using an ACL.

Example 4-2 demonstrates a policy that allows all S3 actions over the bucket called everyonelovesaws from the 192.168.100.0/24 CIDR range.

Example 4-2 *S3 Policy with a Source IP Condition*

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:*",
      "Resource": "arn:aws:s3:::everyonelovesaws/*",
      "Condition": {
         "IpAddress": {"aws:SourceIp": "192.168.100.0/24"},
      }
    }
  ]
}
```

On top of access control to the data, you also need to consider the security of data during transit and at rest by applying encryption. To encrypt data being sent to the S3 bucket, you can either use client-side encryption or make sure to use the TLS S3 endpoint. (Chapter 1 covers encryption in transit in detail.) To encrypt data at rest, you have three options in S3:

- **S3 Server-Side Encryption (SSE-S3):** SSE-S3 provides built-in encryption with an AWS managed encryption key. This service is available on an S3 bucket at no additional cost.
- **S3 SSE-KMS:** SSE-KMS protects data by using a KMS-managed encryption key. This option gives you more control over the encryption keys to be used in S3, but the root encryption key of the KMS service is still AWS managed.

- **S3 SSE-C:** With the SSE-C option, S3 is configured with server-side encryption that uses a customer-provided encryption key. The encryption key is provided by the client within the request, and so each blob of data that is delivered to S3 is seamlessly encrypted with the customer-provided key. When the encryption is complete, S3 discards the encryption key so the only way to decrypt the data is to provide the same key when retrieving the object.

Relational Versus Nonrelational Databases

Before doing a deep dive into the database services available in AWS, you need to take a look at the two major categories of databases that exist. In essence, a database is a collection of data that can be accessed in a certain ordered manner. Traditionally enterprise databases have been designed using a relational format. Enterprises traditionally needed to collect strictly formatted and well-structured data and then perform queries to get insights into how the different pieces of data in the database related to each other. When running business intelligence, analytics, ERP, accounting, management, and other tasks commonly done in an enterprise, it is always preferred to run these tasks on a well-structured dataset to get the clearest results and easily identify trends and outliers. These are typically SQL databases that are designed to run a whole SQL command on one server or even one CPU thread.

But the world has changed. Internet-connected companies today are ingesting data from sources where the structure is undefined, where data is stored for temporary purposes, where relationship information is not provided, or where the relationships are so complex that the sheer volume of data would easily overwhelm a traditional database server. The Internet-connected world needs databases where data can be stored at high velocity, where performance can be scaled linearly by adding nodes, and where data can be stored in an arbitrary format with an arbitrary structure. A new generation of databases has emerged; these databases are called NoSQL (or Not only SQL). These database models are designed to store key/value pairs, documents, data payloads of arbitrary structure, graphs, and so on. Also, nonrelational databases are typically schema-less. Figure 4-4 illustrates different data structures of SQL and NoSQL databases.

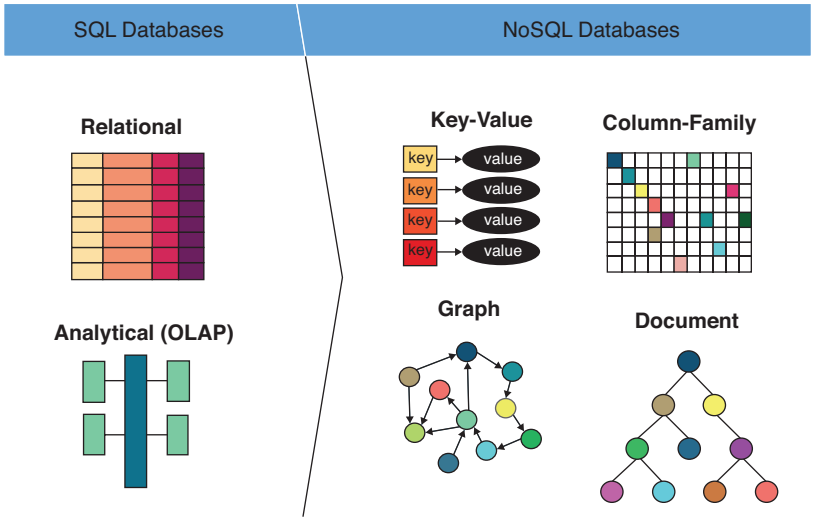

Figure 4-4 *SQL Versus NoSQL Databases*

Choosing which type of database to use is essentially governed by the data model. A typical relational database model is strictly structured by row and column, as illustrated in Table 4-2. The data must fit fully within one row and is required to be structured to fit the categories defined in the columns.

Key Topic

Table 4-2 Relational Database Table Example

Index	Name	Surname	Occupation	Active
0000	Anthony	Soprano	Waste Management Consultant	Y
0001	Christopher	Moltisanti	Disposal Operator	N

Different columns of a traditional database are indexed to expedite the retrieval of the data from the database. The index is usually loaded into memory and allows for very fast retrieval of specific pieces of data. Traditional databases are usually also ACID compliant, where ACID stands for

- **Atomicity:** Every transaction against an SQL database is atomic and (usually) cannot be broken down into smaller pieces. If a transaction fails, the whole operation needs to be restarted.
- **Consistency:** Data must be consistent at all times, even if replicated across a cluster. This means that data will be made unavailable until the replication has completed and the data is confirmed to be consistent.
- **Isolation:** Concurrent transactions can never interfere with each other. In SQL, for example, you always put a lock on a table or an index that you are modifying so no other transaction can interfere with it.
- **Durability:** Data must be stored durably and must also have the ability to be recovered in case of a failure. You should keep a transaction log that can be replayed and one or more backups of a database to maintain durability.

With a NoSQL database, you can represent the whole dataset of one row as a set of key/value pairs that are stored and retrieved as a document. This document needs to be encoded in a format from which the application can build the rows and columns represented in the document. Example 4-3 demonstrates a JSON-formatted document that represents the same data as the first row of your SQL table (refer to Table 4-2).

Key Topic

Example 4-3 *JSON-Formatted Data with Key/Value Pairs Matching the First Row of Table 4-2*

```
{
 "Index":"0000",
 "Name":" Anthony ",
 "Surname":" Soprano"
 "Occupation":" Waste Management Consultant"
 "Active":"Y"
 }
```

To speed up retrieval of the data, you need to select a key that can appear in all documents and allow for the prompt retrieval of the complete dataset. The benefit of this type of format is that only a certain part of the data—not the complete dataset—defines the structure. So you can

essentially shorten or extend the dataset with any number of additional key/value pairs on the fly. For example, if you want to add the date of last activity for a user, you can simply add an additional key/value pair to the document denoting the date, as demonstrated in Example 4-4.

Example 4-4 *Adding the Last Active Attribute to the Data*

```
{
 "Index":"0001",
 "Name":" Christopher ",
 "Surname":" Moltisanti "
 "Occupation":" Disposal Operator "
 "Active":"N"
 "Last active": 13052007
 }
```

You could even structure the day, month, and year as their own nested key/value pairs in the **Last active** key, as demonstrated in Example 4-5.

Example 4-5 *Adding an Entry as Nested Key/Value Pairs*

```
{
 "Index":"0001",
 "Name":" Christopher ",
 "Surname":" Moltisanti "
 "Occupation":" Disposal Operator "
 "Active":"N"
 "Last active": [
 { "Day":13},
 { "Month":"05"},
{ "Year":"2007" },
]
 }
```

The ability to nest keys in your database adds a lot more flexibility to the way you store and access the data in the NoSQL database. Just think of the impact of the schema modifications required to fit the new type of data into an existing SQL table: Not only would the process be disruptive to ongoing operations, but rolling back changes to a schema is sometimes impossible. With NoSQL, you can change the data model on the fly by adding and removing key/value pairs to items with ease.

NoSQL databases are designed with linear scalability in mind as all data is distributed across multiple nodes, which become authoritative for a certain subset of indexing keys. To retrieve the data, you usually address a common front end that then delivers the data by contacting multiple back ends and delivers documents from all of them in parallel. With a SQL database, that design is very hard to implement as the transaction usually cannot be easily distributed across multiple back ends. Unlike SQL databases, NoSQL databases usually conform to the BASE database ideology, where BASE stands for

- **Basic availability:** Availability of the database is the main requirement. The database must seem to be up all the time, and reads from/writes to the database must succeed as much as possible.

- **Soft state:** The state of the system is allowed to change over time: The database is allowed to be repartitioned (by adding or removing nodes), and the data can be expired, deleted, or offloaded. The replication system ensures that the data is replicated as soon as possible, but the availability must not be affected by the any state changes.
- **Eventual consistency:** The system will eventually (after a period of time) achieve consistency of data across all the nodes of a cluster. The data will be available even during replication, and a client requesting data could access a node with a stale piece of data. To mitigate eventual consistency, strongly consistent reads can be utilized to read data from multiple nodes to ensure that the data is always in a consistent state when being read. The read consistency must be handled by the application.

Deploying Relational Databases in AWS

Many applications require the ability to store data in a relational database. From web services, business intelligence, and analytics to infrastructure management, many different tasks require the recording of data in a database. In AWS, you have two choices:

- You can deploy an EC2 instance with a database server application installed.
- You can choose to use Amazon Relational Database Service (RDS).

Amazon RDS

The choice between a standalone EC2 instance with a database on top and RDS is essentially the choice between an unmanaged environment where you have to manage everything yourself and a managed service where most of the management tasks are automated and complete control over deployment, backups, snapshots, restores, sizing, high availability, and replicas is as simple as making an API call. When developing in AWS, it always makes sense to lean toward using a managed service as the benefits of reducing the management overhead can be numerous. Aside from simplifying the management, another business driver can be increased flexibility and automation, which can be achieved by using the AWS CLI, the SDKs, and CloudFormation to deploy the database back end with very little effort or through an automated CI/CD system. Managed services essentially empower developers to take control of the infrastructure and design services that can be easily deployed and replicated and that can have auto-healing characteristics built into them.

Example 4-6 shows how the deployment of an RDS database can be integrated in a Java application by using the AWS Java SDK, giving you the ability to deploy the database and use the database string returned to connect to the newly created database.

Key Topic

Example 4-6 *Java Script That Can Be Used to Build an RDS Database*

```
// define the credentials
AWSCredentials credentials = new BasicAWSCredentials(
  "AJDEIX4EE8UER4",
  " D3huG40jThD3huG40jThNPaAx2P3py85NPaAx2P3py85"
);
AmazonRDSClientBuilder.standard().withCredentials(credentials)  // pull the
credentials into RDS Builder
```

```
  .withRegion(Regions.US_EAST_2) // define the region as us-east-2
  .build();
CreateDBInstanceRequest request = new CreateDBInstanceRequest(); // define the
create request
request.setDBInstanceIdentifier("javadbinstance");   // give the database instance
(the server) a name
request.setDBInstanceClass("db.t3.small"); // define the size of the database
instance
request.setEngine("mysql"); // define the database engine type
request.setMultiAZ(true); // make the database highly available with MultiAZ
request.setMasterUsername("master"); // define the database master username
request.setMasterUserPassword("javadbpw"); // define the database master password
request.setDBName("masterdb");  // give the database a name
request.setStorageType("gp2");   // define the storage type - gp2 is general purpose
SSD
request.setAllocatedStorage(30); // define the storage size as 30 GB
amazonRDS.createDBInstance(request); // issue the request
```

NOTE This example breaks the rules by storing credentials in the code. This can be avoided by running the code on an EC2 instance and using a role with the permissions to create the RDS database. You need to be aware that you also have the ability to enable database authentication via IAM and generate an IAM token within your Java code to authenticate to the database without having any passwords baked in the code.

Once the script is created, you can list all your instances with the **DescribeDBInstanceResult** class. You will want to get the instance identifier and the endpoint, which is the SQL endpoint URL that you can later use to connect to the database. You can do this by including the snippet shown in Example 4-7 in your Java code.

Example 4-7 *Using the Java DescribeDBInstanceResult Class*

```
DescribeDBInstancesResult result = amazonRDS.describeDBInstances();
List<DBInstance> instances = result.getDBInstances();
for (DBInstance instance : instances) {
    String identifier = instance.getDBInstanceIdentifier();
    Endpoint endpoint = instance.getEndpoint();
}
```

Supported Database Types

Currently the RDS service supports six different database engines that can be deployed from RDS:

- MySQL
- MariaDB
- PostgreSQL
- Amazon Aurora

- Oracle
- Microsoft SQL Server

RDS for MySQL, MariaDB, and PostgreSQL

MySQL, MariaDB, and PostgreSQL are the most popular open-source relational databases used in today's enterprise environments. Being open source and requiring little or no licensing while still having enterprise-grade support available makes these databases a great choice for an enterprise looking to deploy applications in a more efficient manner. They can easily replace traditional databases that tend to have expensive licensing attached to them.

The MySQL, MariaDB, and PostgreSQL engines all have similar general characteristics and support highly available Multi-AZ deployment topologies with a synchronous master/slave pair across two availability zones. All of them also have the ability to deploy multiple read replicas in the same region or in another region. The RDS service supports the following versions of these open-source databases:

- MySQL Community Edition versions 5.5+ and 8.0
- MariaDB Server versions 10.0+
- All PostgreSQL versions (though version 9.3.5t is required for Multi-AZ and read replicas)

Figure 4-5 illustrates synchronous replication in Multi-AZ RDS deployments.

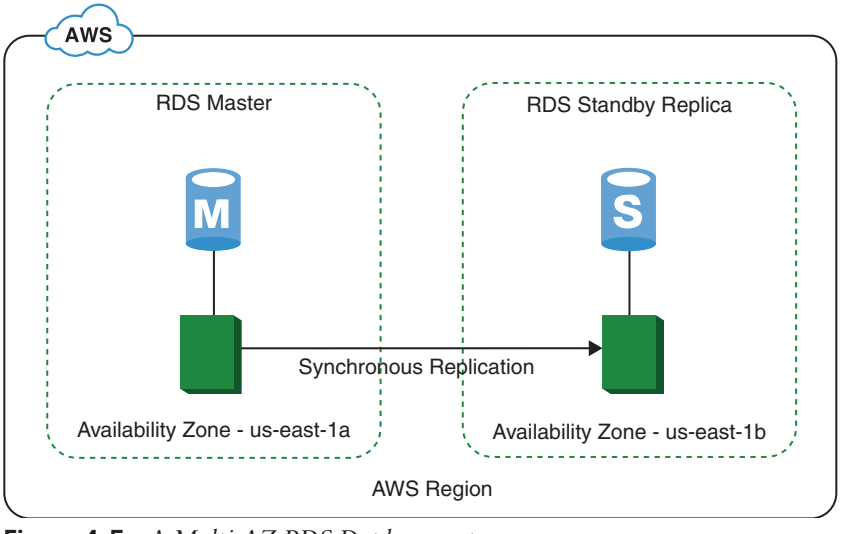

Figure 4-5 *A Multi-AZ RDS Deployment*

The MySQL, MariaDB, and PostgreSQL databases all support the use of SSL connections for the encryption of data in transit and can be configured with built-in volume encryption for data at rest.

These three database types are limited in size to 16 TB per volume and can use numerous different RDS instance types so you can scale the size of an instance from small to 8xlarge.

Amazon Aurora

Amazon Aurora is the next-generation open-source engine currently supporting the MySQL and PostgreSQL database types. The benefit of Aurora is that it decouples the processing from the storage. All the data is stored on a synchronously replicated volume in three availability zones, and the processing of SQL requests is performed on the cluster instances. The instances have no local storage, and they all access the cluster volume at the same time, so the performance of the cluster can be linearly scaled by adding nodes.

The write node in an Aurora cluster, also called the *primary instance*, is used to process all write requests. The primary instance type needs to be scaled to the write performance requirements of your application and can be easily resized by promoting a larger read replica to the primary role. All other members of the cluster are called *replica instances*, and they can respond to read requests. The primary and the replicas have different DNS names to which you send requests, which means you can simply configure your application with two FQDN targets—one for the writes and another for the reads—and do not need to handle the read/write distribution on your own.

Because the primary and replica instances have access to the same synchronously replicated cluster volume, you can also instantly promote any read replica into the primary role if the primary instance fails or if the availability zone where the primary instance is running experiences difficulties. Figure 4-6 illustrates how the Aurora design ensures synchronous writes and decouples storage from the compute layer.

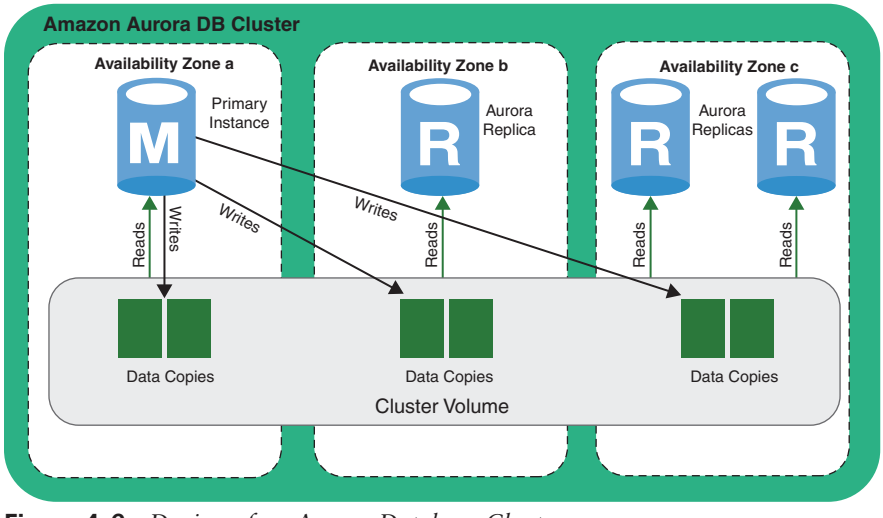

Figure 4-6 *Design of an Aurora Database Cluster*

An Aurora cluster can scale quite a bit because you can add up to 15 replicas to the primary instance while additionally adding another 16 asynchronous replicas in another region. The Aurora engine also extends the maximum cluster volume to 64 TB, delivering not only a performance advantage but also a capacity advantage over traditional open-source databases, while maintaining the ability to use SSL for encryption in transit and delivering built-in encryption at rest.

Aurora is now available in serverless on-demand mode as a pay-per-request service. This is a great option for any kind of transient SQL clusters where keeping the primary and replicas running 24/7 would cause unnecessary costs. The on-demand Aurora also handles all scaling and capacity management automatically so that you can send as many requests as you need and always get a response. This essentially allows you to also support very spiky applications where you are not sure of the performance required before the requests start rolling in.

Oracle and Microsoft SQL on RDS

Traditional enterprise databases are sometimes the only option, so RDS allows you to deploy an Oracle 11g or Microsoft 2008 or newer SQL server as a service. The cost of these two engine types can have the licensing included, so there is no need to spend large sums of money for licensing upfront. There is, of course, also an option to bring your own license for each.

While you have a lot of choice of RDS instance types to run on, the Oracle and Microsoft engines are limited to a Multi-AZ mode and provide no support for read replicas and a maximum size of 16 TB per volume. To protect data at rest and in transit, Transparent Data Encryption (TDE) is supported on both engine types.

Scaling Databases

There are four general ways to scale database performance:

- **Vertical scaling:** You can give a single database engine more power by adding more CPU and RAM.
- **Horizontal scaling:** You can give a database cluster more power by adding more instances.
- **Read offloading:** You can add read replicas and redirect read traffic to them.
- **Sharding:** You can distribute the data across multiple database engines, with each one holding one section, or *shard*, of data.

With relational databases, vertical scaling always works, but it has a maximum limit. In AWS, the maximum limit is the largest instance size that can be deployed in the service. An alternative is horizontal scaling, but generally relational databases are not the best at being able to scale horizontally. The nature of the atomicity of the SQL transactions usually means that the whole transaction must be processed by one server—or sometimes even in one thread on a single CPU.

If an RDS database is deployed in a Multi-AZ configuration, the resizing can be done transparently because the slave database is resized first, the data is synchronized, the connection fails over, and the slave becomes the master while the previous master instance is resized. When the resizing is complete, data is again synchronized, and a failover is performed to the previous master instance.

Example 4-8 uses the boto3 Python SDK to increase the instance size from db.t3.small to db-t3-medium for the instance created in the previous example.

Key Topic

Example 4-8 *Python SDK (boto3) Script That Can Be Used to Create an RDS Instance*

```
import boto3 # boto3 is the AWS SDK for python
client = boto3.client('rds') # define the RDS client to be used
response = client.modify_db_instance( # modify an existing instance
    DBInstanceIdentifier=' javadbinstance ', # specify the instance ID
    DBInstanceClass=' db.t3.medium ', # define the new size
    ApplyImmediately=True, # run the command immediately (will not impact the
availability since we set the database to be MultiAZ
)
```

Another way of scaling is to distribute the read and write transactions on multiple nodes. A typical relational database is more read intensive than write intensive, with a typical read-to-write ratio being 80:20 or even 90:10. By introducing one or more read replicas, you can offload 80% or even 90% of the traffic off your write node. Aurora excels at read replica scaling, whereas the other services that support read replicas support only asynchronous replication, which means the read data is not as easily distributed across the cluster because the data read from the replica might be stale. But even asynchronous replicas can be a great benefit for offloading your write master where historical analytics and business intelligence applications are concerned.

Typically the last resort for scaling relational databases is to shard the data. Essentially this means that a dataset is sliced up into meaningful chunks and distributed across multiple masters, thus linearly increasing write performance.

NOTE The performance in sharding increases linearly only under the utmost perfect conditions, where data distribution across shards is equal. In real-world scenarios, achieving equal distribution of data across shards and retaining meaningful pieces of data on the same server is the biggest challenge.

For example, imagine a phone directory in a database with names from A to Z. When you need more performance, you can simply split up the database into names starting with A to M and N to Z. This way, you have two databases to write to, thus theoretically doubling the performance. Figure 4-7 illustrates the principle of sharding RDS databases to achieve better performance.

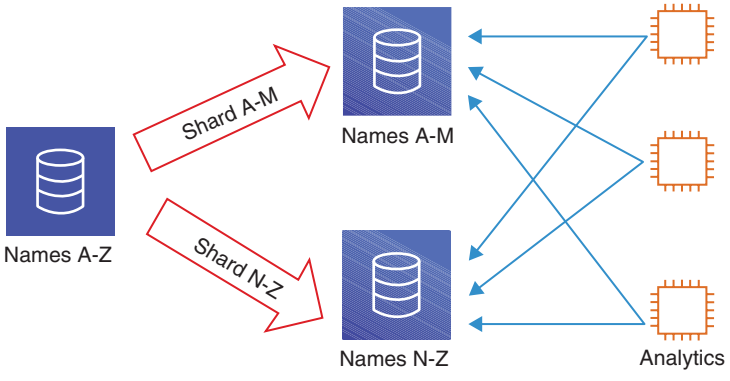

Figure 4-7 *Sharding a Phone Directory into Two Databases*

However, the limitation of sharding is immediately apparent when you try to perform analytics as you need to access two databases, join the two tables together, and only then perform the analytics or BI operation. Figure 4-8 illustrates tables from sharded databases being joined to an analytical database.

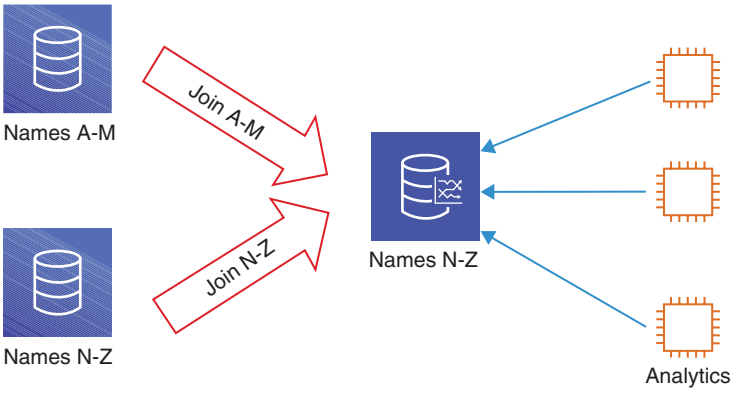

Figure 4-8 *Steps Required for Analytics on Sharded Databases*

Handling Nonrelational Data in AWS

As you saw with the different data models discussed earlier in this chapter, not all data fits well into a traditional relational database. Some cases are more suitable for a NoSQL back end than a standard SQL back end—such as where data requires a flexible schema, where data is being collected temporarily, where data consistency is not as important as availability, and where consistent, low-latency write performance is crucial. AWS offers several different solutions for storing NoSQL data, including the following:

- **DynamoDB:** A NoSQL key/value storage back end that is addressable via HTTP/HTTPS
- **ElastiCache:** An in-memory NoSQL storage back end
- **DocumentDB:** A NoSQL document storage back end
- **Neptune:** A NoSQL graphing solution for storing and addressing complex networked datasets
- **Redshift:** A columnar data warehousing solution that can scale to 2 PB per volume
- **Redshift Spectrum:** A serverless data warehousing solution that can address data sitting on S3
- **TimeStream:** A time series recording solution for use with IoT and industrial telemetry
- **Quantum Ledger:** A ledger database designed for record streams, banking transactions, and so on

As you can see, you are simply spoiled for choice when it comes to storing nonrelational data types in AWS. This chapter focuses on the first two database types, DynamoDB and ElastiCache, as they are important both for gaining a better understanding of the AWS environment and for the AWS Certified Developer–Associate exam.

Amazon DynamoDB

DynamoDB is a serverless NoSQL solution that uses a standard REST API model for both the management functions and data access operations. The DynamoDB back end is designed to store key/value data accessible via a simple HTTP access model. DynamoDB supports storing any amount of data and is able to predictably perform even under extreme read and write scales of 10,000 to 100,000 requests per second from a single table at single-digit millisecond latency scales. When reading data, DynamoDB has support for eventually consistent, strongly consistent, and transactional requests. Each request can be augmented with the JMESPath query language, which gives you the ability to sort and filter the data both on the client side and on the server side.

A DynamoDB table has three main components:

- Tables, which store items
- Items, which have attributes
- Attributes, which are the key/value pairs containing data

Tables

Like many other NoSQL databases, DynamoDB has a distributed back end that enables it to linearly scale in performance and provide your application with the required level of performance. The distribution of the data across the DynamoDB cluster is left up to the user. When creating a table, you are asked to select a primary key (also called a hash key). The primary key is used to create hashes that allow the data to be distributed and replicated across the back end according to the hash. To get the most performance out of DynamoDB, you should choose a primary key that has a lot of variety. A primary key is also indexed so that the attributes being stored under a certain key are accessible very quickly (without the need for a scan of the table).

For example, imagine that you are in charge of a company that makes online games. A table is used to record all scores from all users across a hundred or so games, each with its own unique identifiers. Your company has millions of users, each with a unique username. To select a primary key, you have a choice of either game ID or username. There are more unique usernames than game IDs, so the best choice would be to select the username as the primary key as the high level of variety in usernames will ensure that the data is distributed evenly across the back end.

Optionally, you can also add a sort key to each table to add an additional index that you can use in your query to sort the data within a table. Depending on the type of data, the sorting can be temporal (for example, when the sort key is a date stamp), by size (when the sort key is a value of a certain metric), or by any other arbitrary string.

A table is essentially just a collection of items that are grouped together for a purpose. A table is regionally bound and is highly available within the region as the DynamoDB back end is distributed across all availability zones in a region. Because a table is regionally bound, the table name must be unique within the region within your account.

Figure 4-9 illustrates the structure of a DynamoDB table.

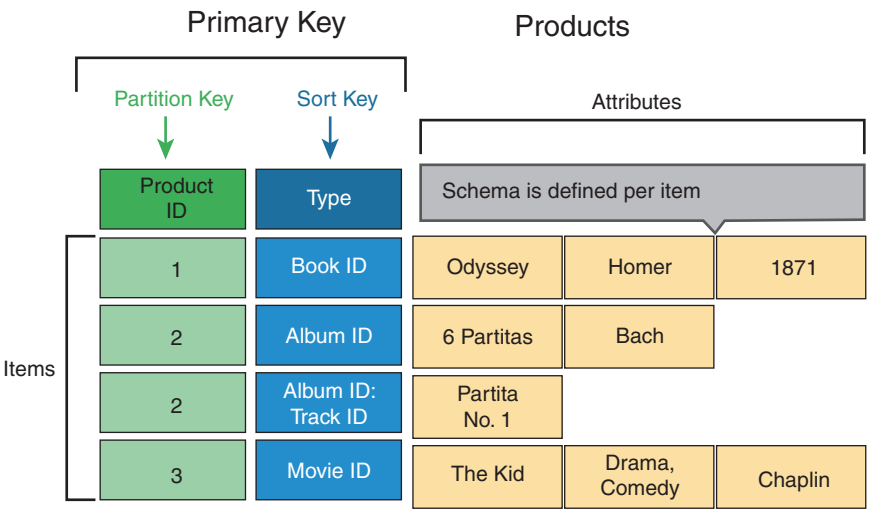

Figure 4-9 *Structure of a DynamoDB Table*

DynamoDB also has support for reading streams of changes to each table. By enabling DynamoDB Streams on a table, you can point an application to the table and then continuously monitor the table for any changes. As soon as a change occurs, the stream is populated, and you are able to read the old value, the new value, and both the new and old values. This means you can use DynamoDB as a front end for your real-time processing environment and also integrate DynamoDB with any kind of security systems, monitoring systems, Lambda functions, and other intelligent components that can perform actions triggered by a change in DynamoDB.

When creating a table, you need to specify the performance mode and choose either provisioned capacity or on-demand mode. Provisioned capacity is better when there is a predictable load expected on the table, and on-demand mode can be useful for any kind of unknown loads on the table. With provisioned capacity, you can simply select AutoScaling for the capacity, which can increase or decrease the provisioned capacity when the load increases or decreases.

Encryption is also available in DynamoDB at creation; you can select whether to integrate encryption with the KMS service or with a customer-provided key.

Items

An item in a table contains all the attributes for a certain primary key or the primary key and sort key if the sort key has been selected on the table. Each item can be up to 400 KB in size and is designed to hold key/value data with any type of payload. The items are accessed via a standard HTTP model where PUT, GET, UPDATE, and DELETE operations allows you to perform create, read, update, and delete (CRUD) operations. Items can also be retrieved in batches, and a batch operation is issued as a single HTTP method call that can retrieve up to 100 items or write up to 25 items with a collective size not exceeding 16 MB.

Key Topic

Attributes

An attribute is a payload of data with a distinct key. An attribute can have one of the following values:

- A single value that is either a string, a number, a Boolean, a null, or a list of values
- A document containing multiple nested key/value attributes up to 32 levels deep
- A set of multiple scalar values

Scalar Type Key/Value Pairs

For each attribute, a single value or a list of arbitrary values exists. In this example, the list has a combination of number, string, and Boolean values:

```
{
 "name" : Anthony,
 "height" : "6.2",
 "results" : ["2.9", "ab", "false"]
}
```

These attributes would be represented in a DynamoDB table as illustrated in Table 4-3.

Table 4-3 Scalar Types

name	height	results
Anthony	6.2	2.9, ab, false

Document Type: A Map Attribute

The document attribute contains nested key/value pairs, as shown in this example:

```
{
"date_of_bith" :  [ "year" : 1979, "month" : 09, "day" : 23]
}
```

This attribute would be represented in a DynamoDB table as illustrated in Table 4-4.

Table 4-4 Table with an Embedded Document

name	height	results	date_of_birth
Anthony	6.2	5, 2.9, ab, false	year:1979
			month:09
			day:23

Set Type: A Set of Strings

The set attribute contains a set of values of the same type—in this example, string:

```
{
 "activities" :
 ["running", "hiking", "swimming" ]
```

}

This attribute would be represented in a DynamoDB table as shown in Table 4-5.

Table 4-5 Table with an Embedded Document and Set

name	height	results	date_of_birth	activities
Anthony	6.2	5, 2.9, ab, false	year:1979 month:09 day:23	running, hiking, swimming

Secondary Indexes

Sometimes the combination of primary key and sort key does not give you enough of an index to efficiently search through data. You can add two more indexes to each table by defining the following:

- **Local secondary index (LSI):** The LSI can be considered an additional sort key for sifting through multiple entries of a certain primary key. This is very useful in applications where two ranges (the sort key and the secondary index) are required to retrieve the correct dataset. The LSI consumes some of the provisioned capacity of the table and can thus impact your performance calculations in case it is created.
- **Global secondary index (GSI):** The GSI can be considered an additional primary key on which the data can be accessed. The GSI allows you to pivot a table and access the data through the key defined in the GSI and get a different view of the data. The GSI has its own provisioned read and write capacity units that can be set completely independently of the capacity units provisioned for the table.

Planning for DynamoDB Capacity

Careful capacity planning should be done whenever using provisioned capacity units to avoid either overprovisioning or underprovisioning your DynamoDB capacity. Although AutoScaling is essentially enabled by default on any newly created DynamoDB table, you should still calculate the required read and write capacity according to your projected throughput and design the AutoScaling with the appropriate limits of minimum and maximum capacity in mind. You also have the ability to disable AutoScaling and set a certain capacity to match any kind of requirements set out by your SLA.

When calculating capacities, you need to set both the read capacity units (RCUs) and write capacity units (WCUs):

- One RCU represents one strongly consistent 4 KB or two eventually consistent 4 KB reads.
- One WCU represents one write request of up to 1 KB in size.

For example, say that you have industrial sensors that continuously feed data at a rate of 10 MB per second, with each write being approximately 500 bytes in size. Because the write capacity units represent a write of up to 1 KB in size, each 500-byte write will consume 1 unit, meaning you will need to provision 20,000 WCUs to allow enough performance for all the writes to be captured.

As another example, say you have 50 KB feeds from a clickstream being sent to DynamoDB at the same 10 MB per second. Each write will now consume 50 WCUs, and at 10 MB per second, you are getting 200 concurrent writes, which means 10,000 WCUs will be sufficient to capture all the writes.

With reads, the calculation is dependent on whether you are reading with strong or eventual consistency because the eventually consistent reads can perform double the work per capacity unit. For example, an the application is reading at a consistent rate of 10 MB per second and performing strongly consistent reads of items 50 KB in size, each read consumes 13 RCUs of 4 KB, whereas eventually consistent reads consume only 7 RCUs. To read the 10 MB per second in a strongly consistent manner, you would need an aggregate of 2600 RCUs, whereas eventually consistent reads would require you to only provision 1400 RCUs.

Global Tables

In DynamoDB, you also have the ability to create a DynamoDB global table (see Figure 4-10). This is a way to share data in a multi-master replication approach across tables in different regions. To create a global table, you need to first create tables in each of the regions and then connect them together in the AWS console or by issuing a command in the AWS CLI to create a global table from the previously created regional tables. Once a global table is established, each of the tables subscribes to the DynamoDB stream of each other table in the global table configuration. This means that a write to one of the tables will be instantly replicated across to the other region. The latency involved in this operation will essentially be almost equal to the latency of the sheer packet transit across one region to another.

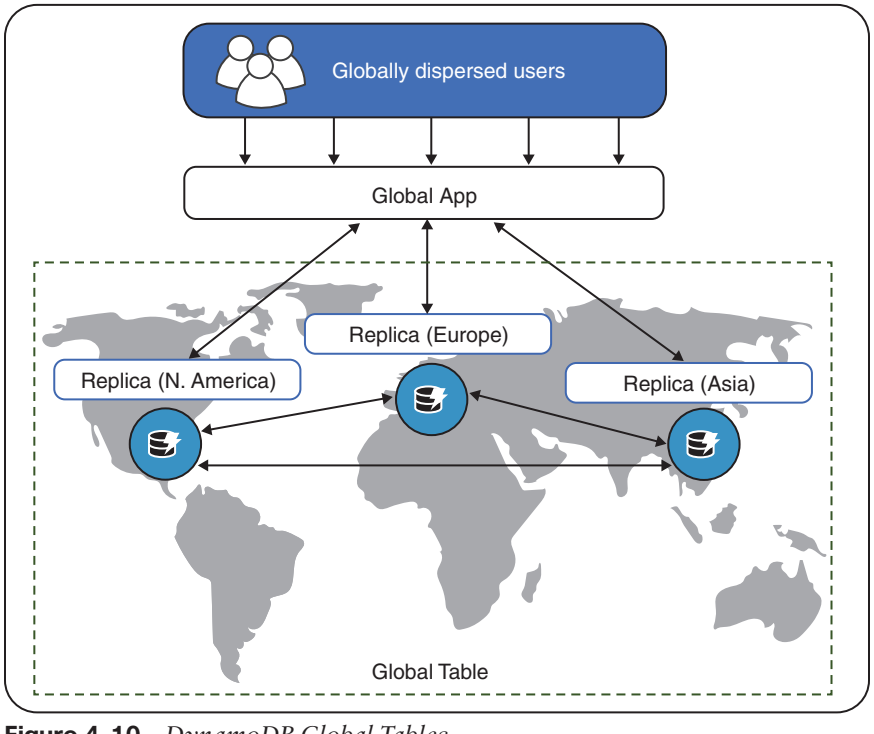

Figure 4-10 *DynamoDB Global Tables*

Accessing DynamoDB Through the CLI

It is possible to interact with a DynamoDB table through the CLI. Using the CLI is an effective way to show how each and every action being performed is simply an API call. The CLI has abstracted shorthand commands, but you can also use direct API calls with the JSON attributes.

In this example, you will be creating a DynamoDB table to create a table called vegetables and define some attributes in the table. To create the table, you use the **aws dynamodb create-table** command, where you need to define the following:

- **--table name:** The name of the table
- **--attribute-definitions:** The attributes with the name of your attribute (**AttributeName**) and the type of value (**AttributeType**: **S** = string, **N** = number, or **B** = binary)
- **--key-schema:** The primary and sort key to use
- **KeyType:** The primary key (**HASH**), and the sort key (**RANGE**)
- **--provisioned-throughput:** The RCUs and WCU

The command should look like so:

```
aws dynamodb create-table \
--table-name vegetables \
--attribute-definitions \
AttributeName=name,AttributeType=S AttributeName=type,
AttributeType=S \
--key-schema \
AttributeName=name,KeyType=HASH AttributeName=type,KeyType=RANGE \
--provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=10
```

NOTE The CLI commands in DynamoDB can be quite extensive, and for the sake of clarity, the preceding example uses the \ newline Linux command-line separator.

After the table is created, you can use the **aws dynamodb put-item** command to write items to the table:

- **--table-name:** The table you want to write into
- **--item:** The item, as a key/value pair
- **"key": {"data type":"value"}:** The name, type, and value of the data

The command should look something like this:

```
aws dynamodb put-item --table-name vegetables \
--item '{ "name": {"S": "potato"}, "type": {"S": "tuber"}, "cost":
{"N": "1.5"} }'
```

If you are following along with the instructions, you can create some more entries in the table with the **put-item** command.

When doing test runs, you can also add the **--return-consumed-capacity TOTAL** switch at the end of your command to get the number of capacity units the command consumed in the API response.

Next, to retrieve the data from the DynamoDB table for your user with the primary key **potato** and sort key **tuber**, you issue the following command:

```
aws dynamodb get-item --table-name users \
--key '{ "name": {"S": "potato"}, "type": {"S": "tuber"} }
```

The response from the query should look something like the output in Example 4-9.

Example 4-9 *Response from the aws dynamodb get-item Query*

```
HTTP/1.1 200 OK
 x-amzn-RequestId: <RequestId>
 x-amz-crc32: <Checksum>
 Content-Type: application/x-amz-json-1.0
 Content-Length: <PayloadSizeBytes>
 Date: <Date>
 {
"Item": {
         "name": { "S": ["potato"] },
       "type": { "S": ["tuber"] },
       "cost": { "N": ["1.5"] },
      }
 }
```

User Authentication and Access Control

Because DynamoDB provides a single API to control both the management and data access operations, you can simply define two different policies to allow for the following:

- Administrative access with permissions to create and modify DynamoDB tables
- Data access with specific permissions to read, write, update, or delete items in specific tables

You can also write your application to perform both the administrative and data access tasks. This gives you the ability to easily self-provision the table from the application. This is especially useful for any kind of data where the time value is very sensitive, also it is useful for any kind of temporary data, such as sessions or shopping carts in an e-commerce website or an internal report table that can give management a monthly revenue overview.

You can provision as many tables as needed. If any tables are not in use, you can simply delete them. When data availability is required, you can reduce the RCU and WCU provisioning to 5 units (the lowest possible setting). This way, a reporting engine can still access historical data, and the cost of keeping the table running is minimal.

For a sales application that records sales metrics each month, the application could be trusted to create a new table every month with the production capacity units but maintain the old tables for analytics. Every month, the application would reduce the previous monthly table's capacity units to whatever would be required for analytics to run.

Because policies give you the ability to granularly control permissions, you can lock down the application to only one particular table or a set of values within the table by simply adding a condition on the policy or by using a combination of allow and deny rules.

The policy in Example 4-10 locks down the application to exactly one table by denying access to everything that is not this table and allowing access to this table. This way, you can ensure that any kind of misconfiguration will not allow the application to read or write to any other table in DynamoDB.

Key Topic

Example 4-10 *IAM Policy Locking Down Permissions to the Exact DynamoDB Table*

```
{
 "Version": "2012-10-17",
 "Statement":[{
 "Effect":"Allow",
 "Action":["dynamodb:*"],
 "Resource":["arn:aws:dynamodb:us-east-1:111222333444:table/vegetables"]
 },
 {
 "Effect":"Deny",
 "Action":["dynamodb:*"],
 "NotResource":["arn:aws:dynamodb:us-east- 1:111222333444:table/ vegetables "]
 },
 ]
 }
```


Caching Data in AWS

Caching is an important feature of designing an application in the cloud as caching offers a double function. On one hand, you can think of caching as a temporary database that can automatically expire and delete stale data; on the other hand, caching can be considered as a system that can deliver frequently used data to the user in a much faster manner.

As a simple analogy, consider your refrigerator and the supermarket. Your fridge is basically your local cache, it takes you seconds to get to the fridge and retrieve a yogurt. But it might take you several minutes or even tens of minutes to go buy a yogurt from the supermarket. You can also introduce several layers of cache. For example, your fridge has the lowest capacity but the fastest retrieval rate, whereas the supermarket has the highest capacity but the slower retrieval rate. In some cases, going to a nearby convenience store might make sense instead of going all the way to the supermarket. So the cost of storage of the yogurt is the most expensive in your fridge, whereas the cost of storage at the supermarket is the cheapest. When caching, you are essentially trying to balance the low cost of keeping the yogurt at the supermarket with a higher cost of the fridge, where you want to keep just enough yogurt to feed the family for a few days.

Amazon ElastiCache

ElastiCache is a managed service that helps simplify the deployment of in-memory data stores in AWS. With in-memory data stores, you can perform caching of frequently retrieved responses, maintain session state, and, in some cases, run SQL-like databases that support transaction type queries through scripting.

One of the primary uses for ElastiCache is simple database offloading. Your application is likely to have a high read-to-write ratio, and some requests are possibly made over and over and over again. If all of these common requests are constantly being sent to the back-end database, you might be consuming more power in that database than needed, and it might become very expensive. Instead of constantly retrieving data from the database, you can ensure that frequent responses are cached in an intermediary service that is faster to respond and that can help you reduce the size of the database server. No matter whether your application requires just a simple place to store simple values that it retrieves from the database or whether it requires a scalable, highly available cluster that offers high-performance complex data types, ElastiCache can deliver the right solution for the right purpose.

Memcached

Memcached is a high-performance, distributed, in-memory caching system. The basic design of the Memcached system is meant for storing simple key/value information. The Memcached service differs from the DynamoDB back end in the fact that each key has only one value. Of course, you can nest multiple values into the value of the key, but there is no index to the data because all the data is stored in memory and retrievable with microsecond latency.

Memcached is perfectly suited for simple caching such as offloading of database responses where the key is the query and the value is the response. It is also perfectly suited for storing session information for your web application, where the cookie ID can be used as the key and linked with the session state as the value.

ElastiCache offers an easy way to deploy a Memcached cluster in a single availability zone.

Redis

When a more advanced in-memory database is required, Redis is the solution. Redis supports running an in-memory database in a more classical approach, with a Multi-AZ pair and read replicas in the cluster. It supports more complex datasets and schema-type data, has the ability to be used as a messaging back end, and gives some transactional data access support through Lua scripting.

Amazon DynamoDB Accelerator

The DynamoDB Accelerator (DAX) service is designed to store hot data from a DynamoDB table in memory, which accelerates the read performance of a DynamoDB database up to 10 times. DAX supports millions of requests per second and reduces the latency for each read request from single-digit milliseconds down to microseconds. DAX has a completely transparent read model; essentially, all reads to your table can be redirected to DAX, and no modification is required on the application side.

Amazon CloudFront

CloudFront is a serverless content delivery network that can enhance the user experience of any application running in the AWS cloud, outside the cloud, or on premises. CloudFront

provides you with the ability to cache common responses from your HTTP/HTTPS web application by caching the responses to GET, HEAD, and OPTIONS HTTP methods. The data is cached at the AWS edge locations, which are distributed closer to densely populated areas in more than 100 different locations. Figure 4-11 illustrates the AWS regions and edge location distribution across the globe.

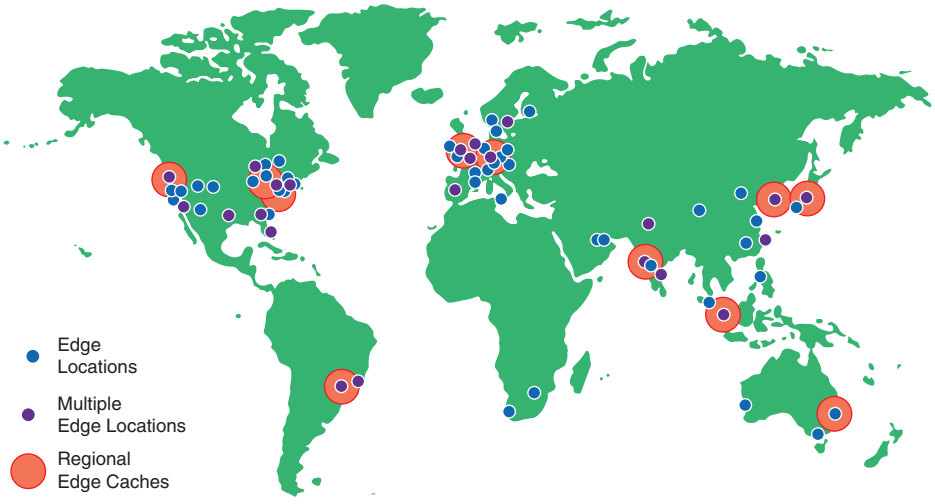

Figure 4-11 *CloudFront Global Points of Presence*

CloudFront also has the ability to establish connections for incoming requests, including PUT, POST, UPDATE, PATCH, and DELETE, thus making it seem as if the application front end is much closer to the user than it actually is. Here is a breakdown of the HTTP methods CloudFront supports:

- **GET:** A read operation that retrieves a document from the web server
- **HEAD:** A read operation that retrieves only the header of the document
- **OPTIONS:** A read request to determine the communication options available on the web server
- **POST:** A write operation that is used to send text-based content to a web server
- **PUT:** A write operation that is used to send a file or data blob to a web server
- **PATCH:** A write operation as an extension to a PUT that enables you to modify an existing file
- **DELETE:** A write operation that deletes a file or some content on a web server

The following settings are supported on a CloudFront distribution:

- **GET and HEAD:** Standard caching for documents and headers. Useful for static websites.
- **GET, HEAD, and OPTIONS:** Adds the ability to cache OPTIONS responses from an origin server.

- **GET, HEAD, OPTIONS, PUT, PATCH, POST, and DELETE:** Terminates all HTTP/HTTPS sessions at the CloudFront edge location and can increase the performance of both read and write requests.

In addition, you can control the time-to-live (TTL) of your cache. By controlling the TTL, you can set a custom way of expiring content when it should be refreshed. CloudFront distributions support the following options for setting TTL:

- **Min TTL:** When forwarding all headers, this is a required setting. Determines the minimum cache lifetime for your CloudFront distribution and determines the shortest interval for CloudFront to check the origin for newer versions of the document.
- **Max TTL:** This optional value defines the longest possible period that objects can stay in the cache. It is used to override any cache-control headers being sent out by the origin.
- **Default TTL:** This optional value works only when no specific TTL is set in the headers coming from the origin. It allows the origin to control its own cache behavior and override the default with cache-control headers.

CloudFront offers the capability to both improve the performance of an application and decrease the cost of content delivery. For example, when delivering content from S3, the transfer costs can add up. With CloudFront, the transfer cost for your data is cheaper per gigabyte. This makes a lot of difference when content that goes viral is hosted on S3. Imagine a video-sharing service where videos tend to go viral and are getting millions of views per day. If each video is 10 MB in size, each million views would carry 10 TB of transfer costs from S3. To achieve the same performance from S3, you can turn on Transfer Acceleration, which increases the delivery speed of content to remote regions. The cost of delivery of the content doubles this way. So with CloudFront you can get initial savings, which can translate to less than 50% of the cost of delivering from S3 with Transfer Acceleration, while also reaping the benefit of having the content cached much closer to the user, who will benefit from the decreased latency of your service. Figure 4-12 illustrates the operation of the CloudFront cache.

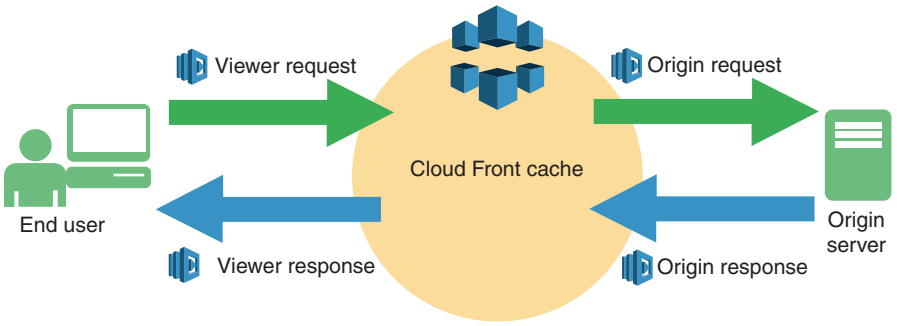

Figure 4-12 *Basic Operation of the CloudFront Service*

Latency is not something to dismiss. Web pages load content somewhat sequentially, and even small increases in back-end performance can add up dramatically. Amazon did a study on latency versus sales performance and discovered that a mere increase of 100 ms in web page load latency would directly influence sales on amazon.com by 1%. Even worse, a study

performed by Google found that the traffic to a typical website decreases by 20% if the latency of the web page load is increased by 500 ms. A typical website sequentially loads anywhere between 10 and 100 objects when delivering a web page, and that can translate to a site loading anywhere from a few seconds (less than 3 seconds is considered good) up to tens of seconds for the worst-performing sites. If the latency to request each of those objects is about 100 ms, that alone adds a whole second for each of those 10 objects. Using CloudFront can bring down the request latency times to single-digit or low double-digit milliseconds, thus drastically improving the performance of a web page's load time even without any site content optimization. It should be noted, though, that optimizing the site content makes the biggest difference; however, optimization can require quite a lot of effort, whereas turning on CloudFront can accelerate a site within minutes.

Another great feature that can help you develop and tune the content delivery is the fact that CloudFront is addressable via the API. This means you can easily control the behavior of the caching environment from within the application. You have complete control over how the headers are forwarded to the origin, you have control over compression, you can modify the responses coming directly out of CloudFront, and you can detect the client type within the cache.

To add some processing power to CloudFront, a distribution can be integrated with Lambda@Edge, which executes predefined functions at the edge location, thus allowing you to include some dynamic responses at the cone of access to your application. The Lambda@Edge execution performance will have the same low latency as the content being delivered from CloudFront and can significantly increase the user experience with your application.

CloudFront Security

CloudFront is secure and resilient to L3 and L4 DDoS attacks when used with AWS Shield Standard. The AWS Shield Advanced service on your CloudFront distribution gives you a 24/7 response team look after your site, allows for custom DDoS mitigation for advanced higher-layer DDoS attacks, and protects you from incurring additional costs associated with the increase in capacity when absorbing a DDoS attack. CloudFront can also be integrated with the AWS Web Application Firewall (WAF), which can help mitigate other types of attacks, such as web address manipulations, injection attacks, and web server vulnerabilities (known and zero-day attacks), and provides the ability to implement different types of rules for allowed patterns, sources, and methods.

To secure data in transit, you can use a TLS endpoint over HTTPS. CloudFront seamlessly integrates with the AWS Certificate Manager (ACM) service, which can automatically provision, renew, and replace an HTTPS certificate on your distribution at no additional cost. This service provides a great benefit to your web application because you never need to worry about renewing, replacing, or paying for an X.509 certificate from a public certificate authority.

You can also use CloudFront to offload all in-transit encryption by sending data to an HTTP origin. When sensitive data is involved, you can use field-level encryption, which only encrypts chosen fields being sent to the server, as with a payment form where the credit card details are encrypted but the rest of the information (such as customer name and address) are sent in clear text to the origin. Field-level encryption uses a set of public and private keys to asymmetrically encrypt and decrypt data across the network and keep the data secure, as illustrated in Figure 4-13.

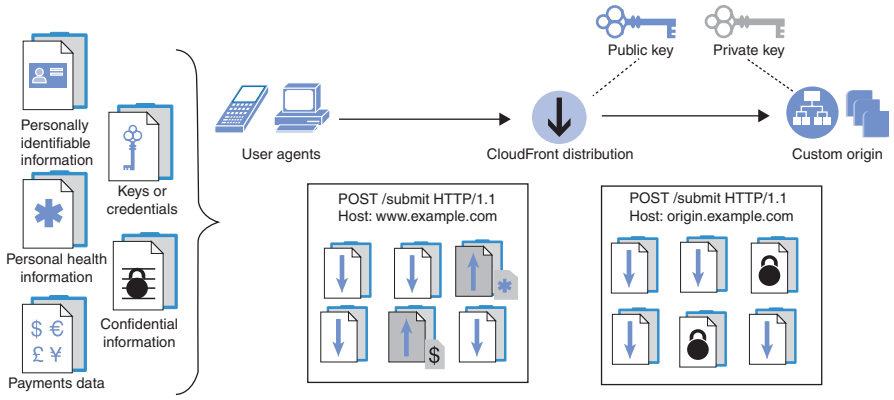

Figure 4-13 *Field-Level Encryption in an AWS CloudFront Distribution*

All data being cached by CloudFront is also automatically encrypted at rest through encrypted EBS volumes in the CloudFront distribution servers.

CloudFront also offers the ability to restrict access to your content in three different ways:

- Restricting access to your application content with signed URLs or cookies
- Restricting access to content based on geolocation
- Restricting access to S3 buckets using Origin Access Identity (OAI)

Key Topic

This example shows how to create an OAI and allow access only to a specific S3 bucket through the identity. The command needs two arguments:

- CallerReference, which ensures that the request can't be replayed (like a timestamp)
- Comment, which essentially gives a friendly name to your distribution

To try this example, run the following:

```
aws cloudfront create-cloud-front-origin-access-identity \
--cloud-front-origin-access-identity-config \
CallerReference=20190820,Comment=everyonelovesaws
```

Make sure to capture the OAI ID from the response because you will be using it in your configuration.

Now that you have created the origin access identity, you need to add the identifier of the origin access identity to the bucket policy that you will protect with the origin access identity. The following policy allows only the origin access identity with the ID **E37NKUHHPJ30OF** to access the **everyonelovesaws** bucket. You apply this bucket policy to the S3 bucket that you previously made public. Example 4-11 shows a policy that allows access for the origin access identity.

Example 4-11 *Bucket Policy for a CloudFront Origin Access Identity*

```
{
    "Version": "2008-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::cloudfront:user/CloudFront Origin Access
Identity E37NKUHHPJ30OF"
            },
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::everyonelovesaws/*"
        }
    ]
}
```


This policy makes your bucket unavailable until you create the distribution in CloudFront. You can test this by trying to access the bucket through its URL.

To create the CloudFront distribution, you can use the cloudfront.json file shown in Example 4-12 as an input to the AWS CloudFront **create-distribution** command. In the file you need to define at least the following sections:

- **CallerReference:** A file that prevents replays
- **DefaultRootObject:** The index file to be used in the distribution
- **Id:** The friendly name for the distribution
- **OriginAccessIdentity:** The ID of the origin access identity
- **DomainName:** The FQDN of the S3 bucket or your custom origin
- **DefaultCacheBehavior:** Want to cache
- **Comment:** A comment, which is required but can be left blank
- **Enabled:** A Boolean that can be set to **true** to enable the distribution or **false** to disable it

Example 4-12 *CloudFront Distribution Configuration File in JSON*

```
{
  "CallerReference": "20190820",
  "Aliases": {
    "Quantity": 0
 },
  "DefaultRootObject": "index.html",
  "Origins": {
    "Quantity": 1,
    "Items": [
```

```
      {
        "Id": "everyonelovesaws",
        "DomainName": "everyonelovesaws.s3.amazonaws.com",
        "S3OriginConfig": {
          "OriginAccessIdentity": "origin-access-identity/cloudfront/E37NKUHHPJ30OF"
        }
      }
    ]
  },
  "DefaultCacheBehavior": {
    "TargetOriginId": "everyonelovesaws",
    "ForwardedValues": {
      "QueryString": true,
      "Cookies": {
        "Forward": "none"
      }
    },
    "TrustedSigners": {
      "Enabled": false,
      "Quantity": 0
    },
    "ViewerProtocolPolicy": "allow-all",
    "MinTTL": 0
  },
  "Comment": "",
  "Enabled": true
}
```

Save this file to where you are running the CLI command and run the **aws cloudfront create-distribution** command as follows:

```
aws cloudfront create-distribution \
--distribution-config file://cloudfront.json
```

This command returns the complete set of JSON settings from the cloudfront.json file, but the most important thing it returns is the distribution FQDN. Look for the following string in the response from the last command:

```
"DomainName": "d1iq7pwkt6nlfb.cloudfront.net"
```

Now you can browse the **d1iq7pwkt6nlfb.cloudfront.net** FQDN and see that your S3 bucket is accessible only from the CloudFront origin access identity. This FQDN can also be used as a CNAME for your website so you can serve your content with your custom domain name.

Exam Preparation Tasks

To prepare for the exam, use this section to review the topics covered and the key aspects that will allow you to gain the knowledge required to pass the exam. To gain the necessary knowledge, complete the exercises, examples, and questions in this section in combination with Chapter 9, "Final Preparation," and the exam simulation questions in the Pearson Test Prep Software Online.

Review All Key Topics

Review the most important topics in this chapter, noted with the Key Topics icon in the outer margin of the page. Table 4-6 lists these key topics and the page number on which each is found.

Key Topic

Table 4-6 Key Topics for Chapter 4

Key Topic Element	Description	Page Number
Section	Working with S3 in the AWS CLI	114
Section	Hosting a Static Website	116
Example 4-1	Example of a bucket policy	116
Table 4-2	Relational database table example	121
Example 4-3	JSON structure for NoSQL database examples	121
Example 4-6	Building an RDS database in the Java SDK	123
Example 4-8	Resizing an RDS database using the boto3 Python SDK	128
Section	Attributes	132
Section	Accessing DynamoDB through the CLI	135
Example 4-10	DynamoDB IAM policy example	137
Tutorial	Creating a CloudFront distribution with OAI	142

Define Key Terms

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

storage, static asset, dynamic asset, relational database, nonrelational database, ACID, BASE, SQL, NoSQL, S3, web server, HTTP, HTTPS, SSL, TLS, object storage, RDS, RDS instance, RDS database instance, RDS connection string, IAM, DynamoDB, ElastiCache, Redis, Memcached, CloudFront, CDN, OAI, CloudFront distribution, origin, Multi-AZ

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. **True or false:** In most cases, it is not possible to determine the type of storage to use simply by looking at the data structure.
2. **True or false:** A video being delivered via a streaming service should be considered a static asset.
3. What is the maximum file size that can be sent to the S3 service in one PUT command?

4. Which types of security documents allow you to limit the access to the S3 bucket?
5. Which types of database engines are supported on RDS?
6. Can an RDS database be resized without service disruption?
7. **True or false:** In a DynamoDB database, both the management and data access are available through the same DynamoDB API.
8. **True or false:** A DynamoDB database always requires you to specify the RCU and WCU capacities and use AutoScaling.
9. Which service would you recommend to cache commonly returned responses from a database?
10. What is an origin access identity in CloudFront?

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W