Professional Expertise Distilled

Mastering AWS Development

Develop and migrate your enterprise application to the Amazon Web Services platform

Uchit Vyas

[PACKT] PUBLISHING enterprise professional expertise distilled

www.allitebooks.com

Mastering AWS Development

Develop and migrate your enterprise application to the Amazon Web Services platform

Uchit Vyas

[PACKT] PUBLISHING enterprise professional expertise distilled

BIRMINGHAM - MUMBAI

www.allitebooks.com

Mastering AWS Development

Copyright © 2015 Packt Publishing

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

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

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

First published: June 2015

Production reference: 1240615

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78217-363-2

www.packtpub.com

www.allitebooks.com

Credits

Author
Uchit Vyas

Reviewers
Goldin Evgeny
Naoya Hashimoto
Dr. Ketan Parmar

Commissioning Editor
Kunal Parikh

Acquisition Editor
Meeta Rajani

Content Development Editor
Pooja Nair

Technical Editors
Manali Gonsalves
Taabish Khan

Copy Editors
Roshni Banerjee
Adithi Shetty

Project Coordinator
Suzanne Coutinho

Proofreader
Safis Editing

Indexer
Priya Sane

Graphics
Sheetal Aute
Jason Monteiro
Abhinash Sahu

Production Coordinator
Nitesh Thakur

Cover Work
Nitesh Thakur

www.allitebooks.com

About the Author

Uchit Vyas is an IT industry veteran, a Cloud technologist at heart, and a hands-on Cloud automation lead at Opex Software, for Cloud automation and DevOps. He is responsible for the delivery of solutions, services, and product development. He explores new open source technologies and defines architecture, roadmaps, and best practices for enterprises. He has consulted and provided training on various tools and technologies, including Cloud computing, Big Data, Hadoop, ESB, infrastructure automation (Chef/Puppet/Ansible), Java-based portals, and CMS technologies to corporations around the world.

He has completed his engineering in computer science from Gujarat University. He worked as a senior associate at Infosys Limited in the Education and Research Team, during which time he worked on Big Data analytics, Cloud security, and virtualization.

He has also published books on Mule ESB, AWS Development Essentials, and AWS DynamoDB and continues to write books on open source technologies.

He hosts a blog named Cloud Magic World, where he posts tips and events about open source technologies mostly related to Cloud on `cloudbyuchit.blogspot.com`. His Twitter handle is `@uchit_vyas`.

I dedicate this book to my family and friends. A special thanks to my loving parents, Hamendra and Shreya Vyas whose words of encouragement and push for tenacity always provided the necessary inspiration. I dedicate this book to my wife Riddhi Vyas who never left my side during the whole writing process and is very special to me. I also dedicate this book to my manager and friend Dr. Manoj Manuja who has supported me throughout the process and given so much encouragement. I will always appreciate all that he has done, especially for helping me to develop my technical skills. And last but not the least, I would like to thank to my best friend and colleague Prabhakaran for being there for me throughout the book program.

---

www.allitebooks.com

About the Reviewers

Goldin Evgeny is a Ruby, Groovy, and Scala software developer who turned into an automation and release engineer to introduce order where chaos usually reigns. On an average day, all things cloud, automation, and continuous delivery get his immediate attention. Back at home, he's a father to his 2-year-old son, dreaming of a day when a proper tech talk can happen between the two. When he gets any spare time, he enjoys exploring the subjects of functional programming, Web performance, and TCP/IP networking. He's an open-source developer, speaker, and passionate advocate when it comes to tools and techniques making for smooth and painless release processes.

Naoya Hashimoto has been working on system designing, implementing, and system maintenance as an infrastructure engineer at Data Center, Management Service Provider, and Housing/Hosting Service Provider for years. After meeting public Cloud services a few years ago, his career, interests, and motives began to face toward public Cloud, including private and hybrid; services related to Cloud computing such as network, storage, orchestration, job automation, and monitoring; and open source software as well. He has worked on *Mastering AWS Development*, *Building Networks and Servers Using Beaglebone*, *PostgreSQL Cookbook*, *Icinga Network Monitoring*, and *Building a Home Security System with BeagleBone* all by Packt Publishing.

Thanks to Packt Publishing for giving me the opportunity to review this book and thanks to the author and coordinator for their contributions to this book. I did enjoy reviewing this book and studying AWS further because I got the opportunity to examine several AWS services that I've never used through this review.

I hope this book will be your first step to AWS!

---

www.allitebooks.com

Dr. Ketan Parmar (aka KPBird) has been a Java enthusiast for more than 7 years. He has worked in various technologies in Java and explored all three areas of Java (SE, ME & EE). He has provided a lot of solutions on StackOverflow and has contributed to various blogs and sites. He is passionate about Java, Android, grid computing, user interface, open-source, and Cloud computing. He has been an eminent speaker for tools and technologies related to Java Enterprise and Android Mobile. He is the leader and founder of JUGAhmedabad (Java User Group Ahmedabad).

www.allitebooks.com

www.PacktPub.com

Support files, eBooks, discount offers, and more

For support files and downloads related to your book, please visit `www.PacktPub.com`.

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

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

`https://www2.packtpub.com/books/subscription/packtlib`

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can search, access, and read Packt's entire library of books.

Why subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print, and bookmark content
- On demand and accessible via a web browser

Free access for Packt account holders

If you have an account with Packt at `www.PacktPub.com`, you can use this to access PacktLib today and view 9 entirely free books. Simply use your login credentials for immediate access.

www.allitebooks.com

www.allitebooks.com

Table of Contents

Preface **vii**

Chapter 1: Architecting in the Cloud **1**

AWS services **2**
The AWS global infrastructure **3**
Regions and Availability Zones **6**
What are AWS regions? 7
What are AWS Availability Zones? 9
How to use AWS AZs 10
AWS EC2 and IAM **11**
The AWS EC2 functionality 11
Instance types and pricing 12
Selecting an instance type 12
AWS EC2 instance numbers and pricing 17
Billing and pricing 17
Ephemeral versus persistent storage **18**
What is ephemeral storage? 18
How to use persistent storage with your instance 24
Scalability, elasticity, and bootstrapping **26**
Bootstrap your instances 27
Black belt booting 28
Identity and Access Management **28**
Accessing IAM 29
Authentication and authorization **31**
Summary **34**

www.allitebooks.com

Chapter 2: Elastic and Fault-tolerant Infrastructure 35
The AWS Elastic infrastructure by Auto Scaling 36
Working with Auto Scaling 37
Ways to access the Auto Scaling service 37
Installing and configuring Auto Scaling 38
Installing Auto Scaling prerequisites 38
Working with Auto Scaling using the CLI 42
Getting started with Auto Scaling using AWS Management Console 45
Summary 57
Chapter 3: Storage Lifecycle Management 59
Data storage scaling 59
AWS DynamoDB 60
DynamoDB data types 61
Creating the first SDK project 65
Java SDK operations 70
The DynamoDB local 76
AWS Simple Storage Service (S3) 79
Amazon CloudFront 85
Creating Amazon CloudFront Distribution 86
Amazon RDS management with CLI 91
Authorizing network access 94
Summary 96
Chapter 4: Web Application and Batch Processing Architecture 97
Alarms with Amazon CloudWatch 98
Creating an EC2 instance 101
Batch processing flow 104
Creating an IAM role 105
Creating SQS tasks 107
Creating S3 bucket 108
Launching worker nodes 109
Dispatching work and viewing results 115
Monitoring the cluster 116
Amazon CloudFormation 117
Where should I start on AWS? 122
Case study 122
LAMP on your Amazon EC2 122
Prerequisites 122
Installing and starting the LAMP server 123
File permissions 124
Testing the LAMP web server 124
Summary 125

Chapter 5: High Availability, Disaster Recovery, and Amazon VPC **127**
Disaster recovery circumstances with AWS **128**
Recovery time objective and recovery point objective 128
Backup and restore 129
Pilot light recovery in AWS 130
Warm standby solution 132
Multisite solution 135
Replication of data **137**
Architecting with Amazon VPC **139**
Launching an instance in the VPC 143
Creating a private subnet 148
Spinning a database instance in the private subnet 148
Creating a Remote Access Software VPN to your VPC 151
Launching an OpenVPN instance 151
Downloading the OpenVPN client 154
Configuring the OpenVPN server 155
Summary **156**
Chapter 6: Tools for AWS and Setup Guidelines **157**
Working with AWS SDKs and IDE toolkits **158**
Working with tools and code libraries **176**
Creating an SDK project 177
Java SDK operations 180
DynamoDB Local **185**
Command-line interface **187**
Summary **191**
Chapter 7: Interacting with AWS Using API **193**
REST-based APIs **194**
Authenticating requests using REST APIs **194**
Getting started with API tools 196
Installing API tools 197
Running your first instance 198
Example of EC2 API 198
Data format for DynamoDB **199**
HTTP requests **200**
Request header 202
Request body 203
Response header 203

Operations in DynamoDB **204**
CreateTable 205
PutItem 205
UpdateItem 206
GetItem 207
Query 208
Scan 209
DeleteItem 210
DescribeTable 210
UpdateTable 211
DeleteTable 212
ListTables 212
BatchGetItem 212
BatchWriteItem 214
Summary **215**

Chapter 8: Amazon Beanstalk, CloudTrail, and Data Warehouse Services **217**

Application deployment using AWS Elastic Beanstalk **217**
Getting started with Amazon Redshift **227**
Configuration options 228
Cluster configurations 230
Interacting with AWS Trail **232**
Features and benefits 232
Case study: migrating applications to the Cloud **238**
Summary **241**

Chapter 9: Bootstrapping and Auto-configuration **243**

Black belt booting **244**
Bootstrapping instances with AWS CloudFormation **250**
Bootstrapping Amazon instances using Chef **252**
Continuous integration and deployment **263**
Automation with Amazon SWF **265**
The workflow execution of Amazon SWF 266
Working with AWS OpsWorks **275**
Creating an OpsWorks stack 277
Creating the Rails App Server layer 278
Creating the database layer 279
Adding instances 280
Summary **286**

Chapter 10: AWS Billing and Amazon CDN Service 287
Programmatic AWS billing 287
Turning on detailed billing reports 288
Select the detailed billing reports you want to receive 289
Referencing your detailed billing report data 290
Cost allocation reporting 290
Cost control architectures 292
Controlling access to your billing report files 292
CDN service from AWS – CloudFront 293
How CloudFront works 293
Getting started with CloudFront 297
Streaming 307
Summary 307
Chapter 11: Analyzing Big Data with AWS 309
Introducing Big Data and Hadoop 310
Introducing Amazon Elastic MapReduce 310
Provisioning a Hadoop cluster on EMR 311
Hive structural design 323
Metastore 323
Compiler 324
The execution engine 324
Supporting apparatuses 324
Data types 325
Data model 326
Indexing on Hive tables 328
Amazon Kinesis 330
Kinesis terminology 330
Streams 330
Data records 330
Producers 330
Consumers 331
Shards 331
Partition keys 331
Amazon Kinesis Client Library 331
Summary 335
Chapter 12: Miscellaneous Features, AWS Security, and Troubleshooting 337
Amazon CloudSearch 337
Creating and configuring a search domain 338
Uploading and indexing the data for search 342
Searching your Amazon CloudSearch domain 344

Amazon Mechanical Turk **347**
AWS Security best practices **350**
Understanding AWS Secure Global Infrastructure 351
Regions, Availability Zones, and service endpoints 353
Managing keys in the Cloud 353
Managing patches 354
Mitigating compromise and abuse 354
The Trusted Advisor tool 354
Troubleshooting practices **355**
Ephemeral disk corruption 355
DNS concerns 356
Resizing or emptying disks 356
Host dispute 356
Security group misconfiguration 356
Summary **357**
Chapter 13: Building Applications and AWS Best Practices **359**
Application impression **359**
Tool mixture **360**
Development phase **360**
Conventions 361
Handlers 361
Starting with EduCloud 362
Handling instance entreaty 365
Instance entreaty sanction 367
Rejecting an instance entreaty 372
Using RDS and Elastic Beanstalk 374
The application of superlative AWS exercises 375
Best practices with AWS **375**
Summary **382**
Index **383**

Preface

Mastering AWS Development is a single place in which you can find solutions for all of your issues with Amazon Web Services. This book will explain how to begin and manage different services using the AWS SDKs and APIs as well as the AWS Management Console, a browser-based graphical user interface to interact with the services. It will include a significant number of examples and use cases that can be used by anyone, from an intermediate to an expert. Using the examples in this book, users can perform advanced-level programming and gain the advantages of AWS services in their SDLC at significantly lower costs on AWS.

What this book covers

Chapter 1, *Architecting in the Cloud*, covers the AWS development platform and its access and how to manage the identity for applications. Later, users will be able to state elasticity, scalability, and bootstrapping functionality using code.

Chapter 2, *Elastic and Fault-tolerant Infrastructure*, discusses how to create scalable infrastructure using EC2, EBS, and an Elastic Load Balancer and, as per the requirement from web traffic, how to scale it efficiently. Users will also learn what Auto Scaling is and launch the configuration with EC2 instances and load balancers.

Chapter 3, *Storage Lifecycle Management*, discusses how to manage the entire life cycle of storage of AWS using different services such as RDS, S3, and Redshift programmatically.

Chapter 4, *Web Application and Batch Processing Architecture*, covers how to design and develop web applications and their required infrastructure. Users will also learn an alarm mechanism and how to create an environment for a batch processing system on AWS.

Chapter 5, *High Availability, Disaster Recovery, and Amazon VPC*, discusses how to create highly available infrastructure for applications and what the vital steps and logic are that should be implemented as a disaster recovery plan. Later, users will also discover how to create Virtual Private Cloud on the AWS Management Console and CLI.

Chapter 6, *Tools for AWS and Setup Guidelines*, covers how to set up and use the AWS code library. Users will also get practical setup instructions for SDKs and IDE toolkits, which can be used during programming with AWS services.

Chapter 7, *Interacting with AWS Using API*, provides hands-on knowledge about APIs and how to connect AWS services through REST-based APIs. Also, users will learn how to authenticate and serve a request of/from API calls.

Chapter 8, *Amazon Beanstalk, CloudTrail, and Data Warehouse Services*, discusses how to migrate and host an existing/new app on AWS and how to identify appropriate services for the app. You will find out how to use the Elastic Beanstalk container service, AWS trail, CloudFormation, and how to do Auto Scaling based upon the requirements from end user traffic.

Chapter 9, *Bootstrapping and Auto-configuration*, covers how to bootstrap AWS EC2 instances with pre-configuration commands for the environment setup and how to use Chef for automation and deployment using code. Users will also learn how the AWS CloudFormation service can work seamlessly with an application and how the SWF and OpsWorks service can be used with the AWS infrastructure.

Chapter 10, *AWS Billing and Amazon CDN Service*, discusses how to do programming for AWS billing, which can be accessed from an application and how to do cost allocation reporting. Users will also learn cost control architecture designs to cut down the cost.

Chapter 11, *Analyzing Big Data with AWS*, provides brief practical knowledge about big data and Apache Hadoop on AWS Cloud. Users will also learn how to use the EMR and Kinesis services with Big Data analytics and for Hadoop solutions.

Chapter 12, *Miscellaneous Features, AWS Security and Troubleshooting*, discusses advanced services administration and programming with CloudSearch and Mechanical Turk. Also, users will discover what kind of security AWS provides and how to use those security features at the infrastructure and application level. At the end of this chapter, users will learn some best troubleshooting practices.

Chapter 13, *Building Applications and AWS Best Practices*, covers the tools/apps that are available to use with AWS for smooth development/migration and deployment. Later in this chapter, users will also learn integration techniques.

Chapter 14, *Third-party Apps and Tools Integration with AWS*, is an online chapter which can be found at `https://www.packtpub.com/sites/default/files/downloads/Chapter14.pdf`.

What you need for this book

To start using this book, you need the following software/applications to be installed on your system:

- An AWS account
- Java 1.6 or higher
- Eclipse (Juno or Kepler)
- The AWS SDK
- AWS CLI tools
- MySQL Workbench

Who this book is for

This book is ideal for programmers, developers, and architects who want to move their existing infrastructure to the AWS Cloud and start using AWS services in all the application tiers using services such as compute, storage, database, queuing, messaging, or mailing in an application, and finally, hosting this application in AWS too. Readers should have a basic knowledge and understanding of Java programs and AWS essentials.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "We can include other contexts through the use of the `include` directive."

A block of code is set as follows:

```
[default]
exten => s,1,Dial(Zap/1|30)
exten => s,2,Voicemail(u100)
exten => s,102,Voicemail(b100)
exten => i,1,Voicemail(s0)
```

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

```
[default]
exten => s,1,Dial(Zap/1|30)
exten => s,2,Voicemail(u100)
exten => s,102,Voicemail(b100)
exten => i,1,Voicemail(s0)
```

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

```
# cp /usr/src/asterisk-addons/configs/cdr_mysql.conf.sample
     /etc/asterisk/cdr_mysql.conf
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "clicking the **Next** button moves you to the next screen".

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to `feedback@packtpub.com`, and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

Downloading the color images of this book

We also provide you a PDF file that has color images of the screenshots/diagrams used in this book. The color images will help you better understand the changes in the output. You can download this file from: `https://www.packtpub.com/sites/default/files/downloads/3632EN_ImageBundle.pdf`.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

www.allitebooks.com

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with any aspect of the book, and we will do our best to address it.

1
Architecting in the Cloud

For many years, application/system analysts and software architects have surveyed and analyzed several important concepts, best practices, and tools to build highly-scalable and fault-tolerant applications or infrastructures. In today's "era of Big Data", these concepts or best practices may be more relevant due to "Big Bang" data, non-predictable web traffic patterns, and the demand for reduced response time. This chapter will introduce, reinforce, and reiterate some of these traditional approaches, and how they can be involved in the context of **Amazon Web Services** (**AWS**). We will also discuss some basic concepts of the AWS terminology, which will be helpful in starting to architect with AWS.

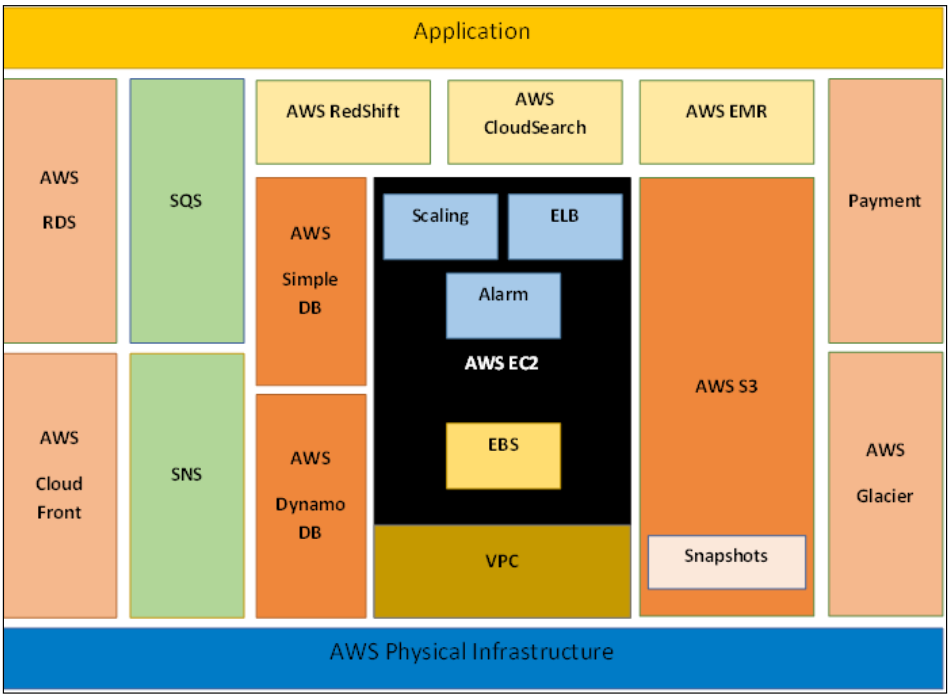
In this chapter, we will cover the following points:

- An overview of AWS services
- The AWS global infrastructure
- Regions and Availability Zones
- An overview of AWS EC2, IAM, and EBS
- Scalability, elasticity, and bootstrapping
- Authentication and authorization

Let's start by introducing AWS services.

AWS services

The AWS Cloud provides a highly-scalable and fault-tolerant infrastructure on which to deploy web-based solutions, with minimal cost and administration, and more flexibility than your in-house infrastructure or datacenter capabilities:



AWS offers a very good number of infrastructure services. The previous fundamental diagram shows you the AWS terminologies and how AWS services can interact with each other and your web-based apps, to provide solutions to build, maintain, and deploy your applications, needs, a wide range of various technological services which will help you to deploy and manage your applications. Clients and customers always ask: what demonstrates a fully managed and flexible technical infrastructure platform? To start from scratch, you can search for the AWS platform, which delivers an industry-leading infrastructure platform with all the required features that Cloud brings. Also, AWS provides knowledge about how AWS satisfies custom requirements and why users might need each service capability provided by AWS.

AWS started contributing to their highly available infrastructure platform in 2006, based on the pay-as-you-go model. After that, whatever they have garnered as services and customers till now is remarkable because they have thousands and thousands of customers across 191 countries who use AWS platform services for their initiatives, and the number is increasing in the AWS customer bucket. AWS provided around 160 features and services in 2012 and around 280 in 2013. In 2014, the number is increasing further.

The AWS global infrastructure

At present, AWS supports nine regions all over the world, which are the East Coast of the U.S., the West Coast of the U.S., Europe, Tokyo, Singapore, Sydney, Brazil, 26 redundant Availability Zones, and 56 Amazon CloudFront points-of-presence, and this number is increasing with time.

It is very crucial and important to have an option to place apps as close as possible to your customers and end users when you create and deploy apps, by ensuring the best possible lowest latency and user expected features and experience for performance. For this, AWS provides regions worldwide. Specific regions are as follows:

- US East (Northern Virginia) region
- US West (Oregon) region
- US West (Northern California) region
- EU (Ireland) region
- Asia Pacific (Singapore) region
- Asia Pacific (Sydney) region
- Asia Pacific (Tokyo) region
- South America (Sao Paulo) region
- US GovCloud

Apart from infrastructure-level highlights, AWS have plenty of managed services, which can be the cream of the AWS candy bar! The managed services bucket has the following services:

- **Security**: For every organization, security is a very vital element. For that, AWS has several remarkable security features, which distinguish it from other Cloud provides. At the moment, I am just underlining the security features at a very high level but we will discuss all the features of AWS security in *Chapter 12*, *Miscellaneous Features, AWS Security and Troubleshooting*. The security features of AWS are as follows:
    - Certifications and accreditations
    - Identity and Access Management
- **Global infrastructure**: AWS provides a fully-functional, flexible technology infrastructure platform worldwide with managed services with certain characteristics, for example:
    - Multiple global locations for deployment
    - Low-latency CDN service
    - Reliable, low-latency DNS service
- **Compute**: AWS offers a huge range of various Cloud-based core computing services, including a variety of compute instances, which can be automatically scaled to justify the needs of your users and application; a fully managed elastic load balancing service; and more fully managed desktop resources on the pathway of AWS Cloud. Some of the common characteristics of computer services include the following:
    - Broad choice of resizable compute instances
    - Flexible pricing opportunities
    - Great discounts for compute resources are always on
    - Lower hourly rates for elastic workloads
    - Wide-range of networking configuration selections
    - A widespread choice of operating systems
    - Virtual desktops
    - One can save as one grows, with the tiered pricing model

- **Storage**: AWS offers low cost with high durability and availability with their storage services. The pay-as-you-go pricing model with no commitment provides more flexibility and agility in services and processes for storage with a highly secure environment. AWS provides storage solutions and services for backup, archive, disaster recovery, and many more. They also support block, file, and object kind of storages with highly available and flexible infrastructures. A few major characteristics of storage are the following:
    - Cost-effective, high-scale storage varieties
    - Data protection and data management
    - Storage gateway
    - Choice of instance storage options
- **Content delivery and networking**: AWS offers a wide set of networking services, which enable us to create a logical isolated network that network architects define and, creates a private network connection to the AWS infrastructure, with fault-tolerant, scalable and highly available DNS services. It also provides delivery services to your end users for content by very low latency and with high data transfer speed with the AWS CDN service. A few major characteristics of content delivery and networking include the following:
    - Application and media files delivery
    - Software and large file distribution
    - Private content
    - Device detection
- **Databases**: AWS offers fully managed, distributed relational, and NoSQL types of database services. Moreover, database services are capable of in-memory caching, sharing, and scaling with/without data warehouse solutions. A few major characteristics for databases include the following:
    - RDS
    - SimpleDB and DynamoDB
    - Redshift
    - ElastiCache

- **Application services**: AWS provides a variety of managed application services with low cost application streaming and queuing, transcoding, push notifications, searching, and so on. A few major services for databases include the following:
    - AppStream
    - CloudSearch
    - Elastic Transcoder
    - SWF, SES, SNS, SQS
- **Deployment & management**: AWS offers the management of credentials to explore AWS services such as monitor services, application services, and updating stacks of AWS resources. They also have deployment and security services alongside the AWS API activity. A few major characteristics of deployment and management services include the following:
    - IAM
    - CloudWatch
    - Elastic Beanstalk
    - CloudFormation
    - Data pipeline
    - OpsWorks
    - CloudHSM
    - Cloud Trail

Additionally, there are a couple more additional important services from AWS such as support, integration with the existing infrastructure, Big Data, and ecosystem, which put them on the top of other infrastructure providers. As a Cloud architect, it is necessary to learn Cloud service offerings and their all-important functionalities. Let's look at AWS start up fundamentals and core technical concepts.

Regions and Availability Zones

AWS is a wide-ranging Cloud service provider, which empowers enterprises to start all phases of their business, ranging from small enterprise portals to large transactional data projects, and from mobile applications to gaming.

Failure can happen at any point in time and can affect the availability of instances, which reside in the same geographical locations. Although rare, if you are hosting your application in the same geographical location, you may experience this kind of failure, and your whole environment may be down at some point in time.

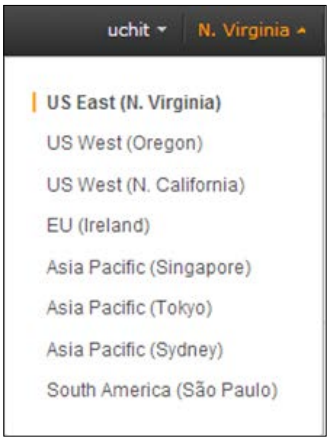
What are AWS regions?

AWS EC2 can be hosted in different locations worldwide. These multiple locations are created as regions. Amazon EC2 offers you the flexibility and ability to use resources, such as instances and databases, in multiple regions or geographical locations. However, resources won't be replicated specifically across multiple regions until and unless you do it externally.

In the following screenshot, you can see the eight currently available regions now over the globe, which can accessible from anywhere. However, as per geographical conditions, the pricing is different for particular regions among various services. AWS services are also subject to this, AWS regions as well AWS services are not supported in each and every region. To check whether a service is available in a specific region, you can go to `http://aws.amazon.com/about-aws/globalinfrastructure/regional-product-services/`. There is one more region called "GovCloud", which is for the U.S. citizens only. So, apart from U.S. citizens, nobody else can access that region.

Each AWS region is intended to be completely insulated from other AWS regions by geographical location. So this will achieve the high availability and stability with fault-tolerance. In most appropriate situations, it is good to deploy your apps as close as possible to your end users. For example, if the majority of your users are from the UK, it would be best to go with the EU (Ireland) region because it is the nearest one. Other points you need to consider when choosing the regions are legal clauses and costs.

All major AWS services (with the exception of CloudFront and Route 53) allow you to choose a region that you would like to work with. The default region will be N. Virginia. You can select the region by using the drop-down menu, as shown in the following screenshot:



After launching the resources, one can only view those resources tied to specified regions over the globe. As regions are totally insulated from other regions, resources won't replicate automatically between multiple regions.

While working with an instance using the **command-line interface** (**CLI**) or API actions, you have to declare the regional endpoint and, if you are launching an instance, you have to select an **Amazon Machine Image** (**AMI**), which resides in the same region. If your AMI is in another region, you have to first copy that AMI to your existing working region to launch it. For the AWS EC2 endpoint, please refer to the following table:

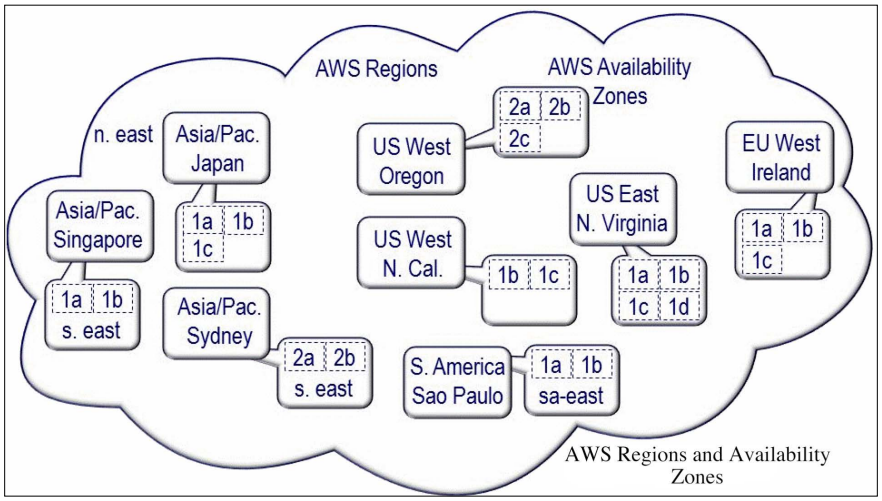
Region name	**Endpoint**
US East (Northern Virginia)	`ec2.us-east-1.amazonaws.com`
US West (Oregon)	`ec2.us-west-2.amazonaws.com`
US West (Northern California)	`ec2.us-west-1.amazonaws.com`
EU (Ireland)	`ec2.eu-west-1.amazonaws.com`
Asia Pacific (Singapore)	`ec2.ap-southeast-1.amazonaws.com`
Asia Pacific (Sydney)	`ec2.ap-southeast-2.amazonaws.com`
Asia Pacific (Tokyo)	`ec2.ap-northeast-1.amazonaws.com`
South America (Sao Paulo)	`ec2.sa-east-1.amazonaws.com`

Table 1.0 – AWS EC2 region wise endpoints

You can copy both types of AMI, AWS EBS-backed AMIs and instance-store-backed AMIs. You can copy AMIs into multiple regions whenever you want them. You can also copy an AMI into the same region for custom use. However, each AMI has a unique AMI ID, so if you copy an AMI from one region to another, it will work as a new AMI with a new unique ID. If you are communicating between multiple regions, it will be over the Internet. That's why administrators or developers have to be careful about the communication channel, and they need to use proper encryption methods to protect their data. Data transfer charges will be applicable on both ends, sending the data from an instance and receiving it at the instance end.

What are AWS Availability Zones?

With AWS EC2, you can place instances in several geographically distinct locations. Locations are combinations of regions and **Availability Zones** (**AZ**). AWS EC2 Availability Zone locations can be within regions that are designed to be isolated from other zones' failures. You can get some very good advantages such as low latency and cheap network connectivity while using EC2 Availability Zones within the same region.



The biggest advantage of, deploying your apps across multiple AZs make your architecture ready and fault-tolerant for unexpected outages. So if a breakdown occurs in a single Availability Zone and you deployed your app in multiple AZs, your app will remain accessible from different AZs. At the time of writing this chapter, there are a total of 25 AZs that exist all over the globe.

www.allitebooks.com

Availability Zone count can vary with time in regions because with time it increases based on the demand and infrastructure.

How to use AWS AZs

Every AZ will be running on its own infrastructure environment, with self-governing cooling, network with security, and power. AZs are not affected by common failures such as generators or cooling equipment failure. The great advantage of AZs is that they are physically separated so that disasters such as fire, floods, and tornadoes won't affect more than one AZ. Every AZ may have a single or multiple data centers internally as per the infrastructure availability. Each AWS Account has independently mapped AZs, which can vary between different accounts. To map an IP address with AZs, you can use the Elastic IP addresses.

To check AZ statistics with regions, you can sign in to the AWS Management Console and go to the EC2 console. In EC2, the navigation bar will show you the regions and from those regions, you can check the associated Availability Zones when an instance or **Amazon Elastic Block Storage** (**Amazon EBS**) volume is going to be launched. If you don't specify the AZ at the time of the instance launching, it will take the default AZ based on the available capacity and system health. To select and get optimal output from the AZs, you can consider the following architecture points as per the requirement. Though you can change your architectural design at any time, based on the traffic and user's behavior on the Cloud, you need to focus on the price, consistency, site downtime, and performance. The following architecture use cases may be the most suitable for you:

- Simple failover: The most reasonable failover preference contains a running deployment of your application in a primary AZ. Backup deployment is arranged for launch in a different AZ, in a situation where the primary zone fails. The expected downtime is nearly 8-10 minutes after launching the backup deployment.
- Intermediary failover: A slightly diversified deployment is set up across two AZs; this works best for production deployments that cannot afford a 10-minute downtime.
- Advanced failover: This is the best choice for a deployment strategy that needs reliability with around 99-100 percent uptime. However, of course, this is the most costly of the three. This give you a continuously running site not withstanding a deleted instance and no reaction for your application infrastructure.

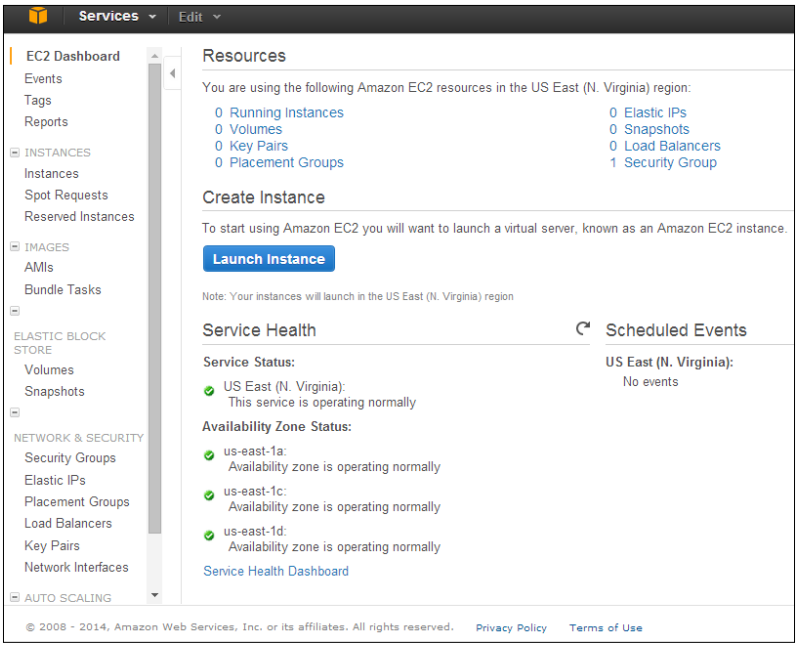
In this section, you will learn some of the business and technical specifications of AWS services, such as EC2, IAM, and the architectural terms of Amazon regarding infrastructures. Let's start with the core services of AWS: **Elastic Compute Cloud** (**EC2**) and **Identity & Access Management** (**IAM**).

AWS EC2 and IAM

Amazon EC2 is a web service (launched in 2006) that helps virtual machines run on host applications or databases. In other words, it provides a resizable compute capacity in the AWS Cloud. You can group your OS, application software, and its associated configuration settings with EC2 AMIs. An AWS user can boot AMIs to launch virtual machines. In AWS terms, it is called an "instance". A user can create, configure, launch, and delete the EC2 instances as per requirements. You can use AMIs to launch a number of virtual instances and also decommission them using web service calls to scale your environment as per your requirements.

The AWS EC2 functionality

AWS EC2 generates a fully virtualized environment, which allows you to use a web-based interface to start instances with different types of operating system and, built-in application packages with custom permissions. You can run *n* number of instances from the same AMI as required.



To work with AWS EC2, you have to:

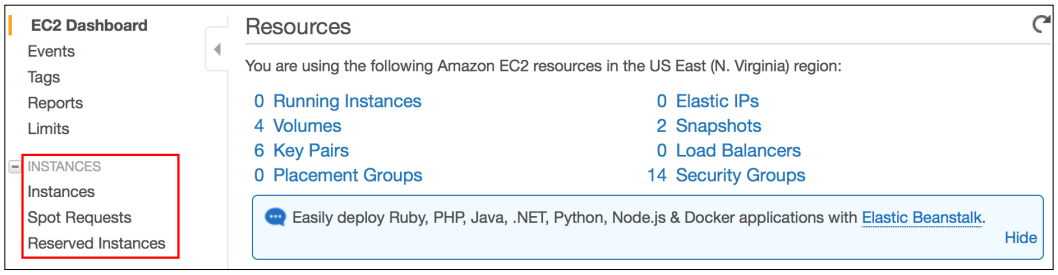
1. Choose a pre-configured template called AMI to run and make it available for use. Another option is to manually create your own custom AMIs to have your required applications and configurations.
2. Create a new or configure an existing default security group and network accessible protocols to reach your instance from the backend as well as from the frontend.
3. Select your instance type and size based on the requirement.
4. Select your instance location, addition storage, plus the Elastic IP requirement.
5. Pay only for what you use per hour.

Instance types and pricing

AWS EC2 instances are the basic building blocks required for your computing requirements on the AWS Cloud platform. Also, as it's a core component of any customer infrastructure, AWS provides a variety of instance types. Instance types comprise various combinations of memory, CPU, storage, and network capabilities, and so will give the freedom to AWS users to select the appropriate mix of resources for their web-based solutions.

Selecting an instance type

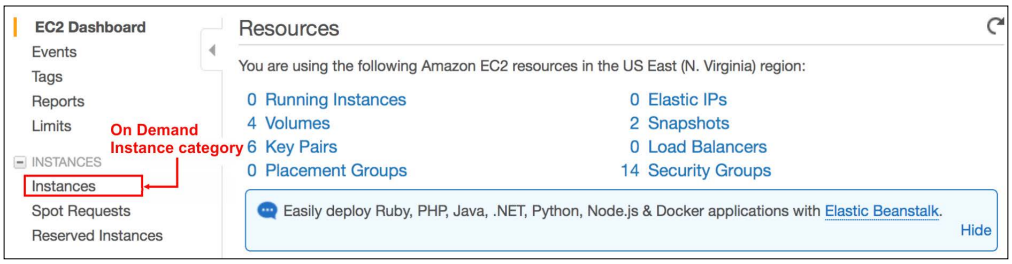
For an easier understanding of instance types and based on real-time cases, AWS has grouped together instances into families based on final application usages. The following information guide will help you to think about the instance type that would be best suited as per your requirements.



AWS EC2 instances can be differentiated via three purchasing models:

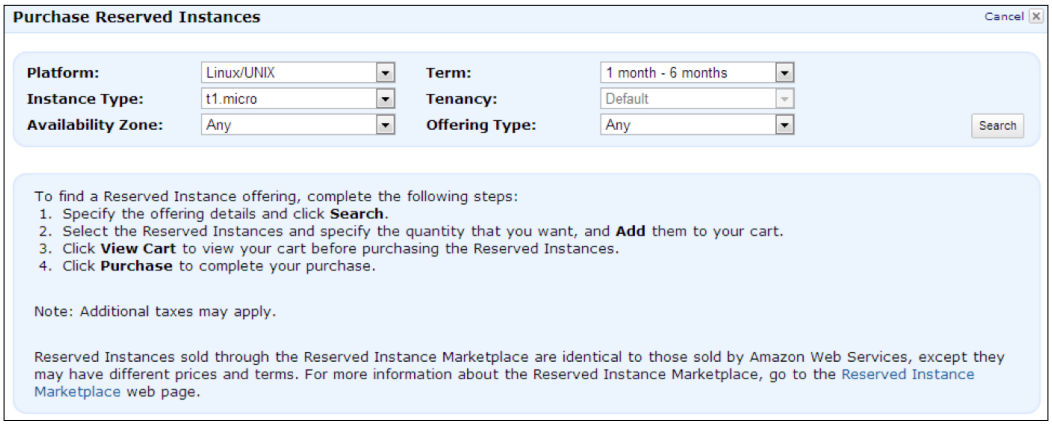
- On-demand instance
- Reserved instance
- Spot instance

At the beginning of Cloud technology, it was first defined by its properties of elasticity and supply of infrastructure with whatever configuration the consumer needs and whenever he needs. This was the primary logic behind on-demand instances.

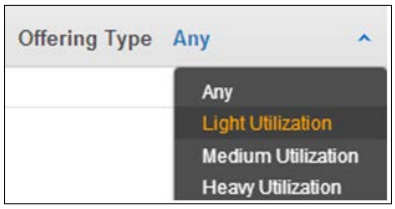


You don't need to pay any up-front cost, but only charges per hour plus how long you are going to run the AWS instance for. This purchasing model is very flexible, but it is also the most expensive one. If somebody requires dynamic throughput, the on-demand instance type can be the best choice for them.

If you know the exact duration for the instance to become available on AWS, accurate usage as per the instance types, and the geographical location of instances from where you can serve the content with low latency to your end users, you can go with reserved instances:



Reserved instances always start as on-demand instances but the only difference is the billing method. You have to pay a one-time fee to cut down the per hour charges at runtime. If you are going to use AWS EC2 instances all the time like 365x24x7, you will realize a drastic price reduction in billing.

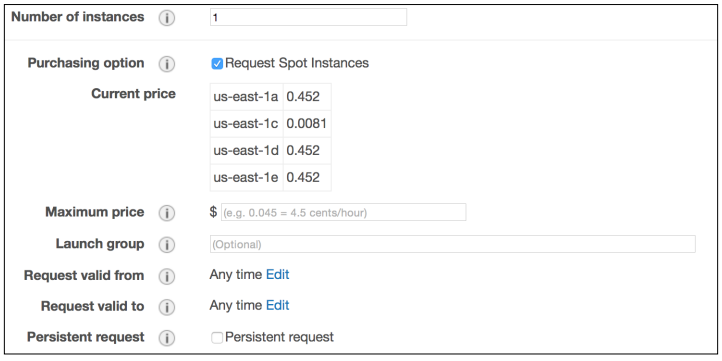


The reserved instances are classified based on their utilization, for example, light utilization, medium utilization, and heavy utilization. The basic difference between these three types is their upfront cost and per hour billing. If you select a heavy utilization type, per the hour charges will be much less but the upfront cost will be higher than the light and medium utilization types.

After purchasing the reserved instance, you cannot change the platform, AZ, instance type, term period, and offering type.

Again, if you are going for light utilization, you have to pay a smaller upfront cost but the per hour charge will be increased compared to high utilization and medium utilization. The best use case is to run light and medium utilization type instances for those who often need to run a particular instance type, but not all the time.

However, after offering resources to end users over the globe, AWS has unused infrastructure in their datacenters. So AWS has decided to rent this infrastructure by bidding it. A user will come and bid for the instance, and they will get the instance resource as long as their bid is higher than the others:



In other words, in Spot instances, you can specify the maximum price based on the per hour charge you'll pay and if there is space, you will get the instance. At any point in time, if you are outbid, your instance will be snatched away from you without any prior notice. That means you can utilize these kinds of resources to finish the process when it's most economical. Spot instances can be best suited to web applications or web servers, where if one or more instances shuts down, the process won't be affected.

The Spot instances console is the same as an on demand console except the configuration step in which a user will define the bid price.

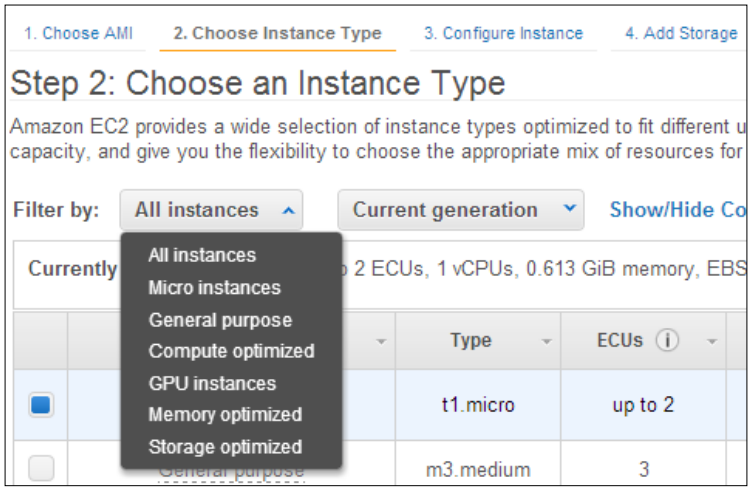
Whenever you are going to launch an instance in any region and whatever the instance type you selecting, determine the hardware of the server/instance, which will be used in your virtual infrastructure. Each and every instance type has different RAM, CPU power, and storage capacities. Among these instance types, you have to choose the most suitable instance type on which you can start hosting your software or applications. AWS EC2 gives you consistent CPU power, regardless of its whole underlying hardware infrastructure/datacenter.

AWS EC2 reserves some of the resources from its pool for the host computer, such as the CPU, memory, and some instance storage for an appropriate instance. AWS EC2 will share the remaining resources of the existing host computer, such as network and disk, among other instances. So, if any instance on a host computer tries to utilize the capacity of shared resources, each instance will receive an equal part of those shared resources. And if the resource performance and usage is under-utilized, an instance can acquire a larger share of those shared resources as it can in that period of time.

In AWS EC2, each instance will provide you with higher performance or lower performance based on that instance type and shared resource. Let's take an example. Say you select a high I/O instance type, which has a higher allocation of the shared pool of resources. Allocating a higher set of shared resources will also vary the I/O results of the instance type. For a general application that has a normal load, moderate I/O performance is sufficient. However, applications such as market trading will require a higher or more consistent I/O performance by considering an instance type that has a higher I/O performance output.

Available instance types

AWS has divided the instances based on common usage patterns, configuration families, and its configuration properties, as shown in the following image:



You can find out more about common instance categories at `http://aws.amazon.com/ec2/instance-types/`

Popular use cases for instance categories

The various instance categories are as follows:

- General purpose instance: Data processing, small size databases, enterprise applications/portals like SAP, SharePoint, and so on.
- Compute-optimized instances: Batch processing, websites that have very high traffic, GNOME analysis, ads and media serving, computational fluid serving, and video encoding.
- GPU instances: Application/software and 2D/3D application streaming, rendering and engineering design, and so on.
- Memory-optimized: Applications with larger deployment and analysis such as SAP, GNOME assemble analysis, distributed memory caches, and so on.
- Storage-optimized: Scale out transactional databases, data warehousing, and Hadoop.
- t1.micro instances: Low traffic sites, getting hands on with AWS EC2 or for some free tier stuff.

AWS EC2 instance numbers and pricing

In general, you are limited to run a total of 20 on-demand or reserved instances and can request 100 Spot instances per region. If you are a new customer, the instance limit can be lower than the limit described. Certain instance types are more specific with numbers per region and can vary based upon the account and production instance request form. You can find more details regarding limits at the following URL:

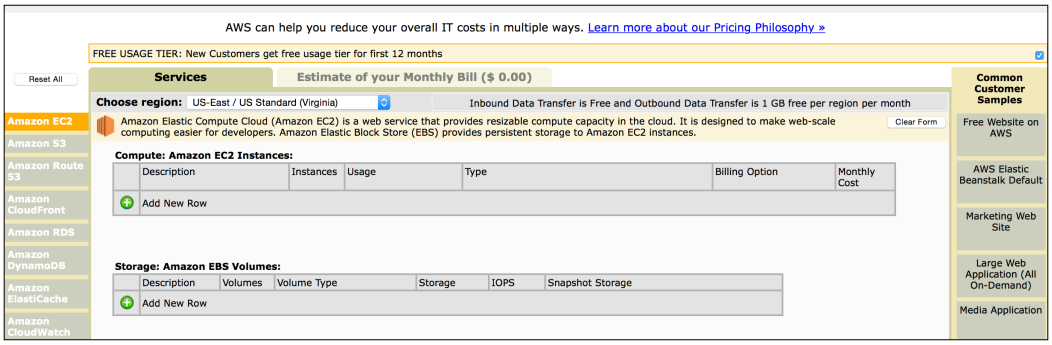
`http://aws.amazon.com/ec2/faqs/#How_many_instances_can_I_run_in_Amazon_EC2`

Billing and pricing

On AWS, you will be charged for what you use and there are no minimum charges; in other words, they follow the pay-as-you-go model. Charges will be based on per hour usage of an instance type. Partial or semi partial instance hours consumed will be counted as full hours. For example, if you are using an instance for 55 minutes, you will be charged for 60 minutes. There is no data transfer charges between two Amazon services within the same region, for example, if you are transferring some data from AWS EC2 US West to AWS RDS US West. Charges are exclusive of applicable taxes and duties, including VAT and applicable sales tax in monthly billing.

AWS services, data transferred between multiple regions, and the usage of other AWS resources will be billed separately to AWS EC2. Billing originates when Amazon EC2 initiates the boot sequence of an AMI instance. The billing cycle stops when the instance terminates, which could ensue through a web services command, by running `shutdown -h` or due to instance failure. To count the cost before initiating the new AWS resources you can check the estimate monthly billing at `http://aws.amazon.com/calculator`.

The calculator page will look like the following screenshot:



Calculator page

On the calculator, you can estimate your monthly billing based on your approximate services-based usage.

Ephemeral versus persistent storage

You learned about the EC2 instance and instance types, we saw the instance storage option in the previous table. Starting from the m1.small type, you can identify that the extra storage will be added to that instance automatically without attaching the storage externally. It can be more noticeable in larger or bigger instances such as m1.medium and so on, where you will get a large disk in the `/mnt` directory of your Linux system. And guess what? It's absolutely free for you! You won't get charged for using it to read or write.

What is ephemeral storage?

Ephemeral disk will be a temporary storage option that will be added automatically to your instance, and it depends on your instance type. The ephemeral disk size of an instance store ranges from 150 GB to 48 TB, and varies by a particular instance type. The bigger your instance, the bigger the ephemeral storage disk you will be assigned.

For certain instance types, such as c1.medium and m1.small, they practice instance storage repeatedly as SWAP as they have a restricted quantity of memory, whereas several are generally structured and mounted at the `/mnt` directory.

The ephemeral storage in an instance, is a temporary storage in nature, one should not depend on these disks to keep long-term production data or even other important data that you would not like to misplace when an instance stop happens, for example, stopping and starting an instance, failure of the underlying hardware of EC2, or terminating an instance. Because of these problems, you should keep in mind that EBS/S3 or any other insistent storage will be the best solution. Let's take an example of losing ephemeral disk data while starting and stopping an instance. The following steps test this operation practically:

1. Launch an instance that has an ephemeral disk:

	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance
☐	General purpose	t2.micro Free tier eligible	1	1	EBS only	-	Low to Moderate
☐	General purpose	t2.small	1	2	EBS only	-	Low to Moderate
☐	General purpose	t2.medium	2	4	EBS only	-	Low to Moderate
☐	General purpose	m3.medium	1	3.75	1 x 4 (SSD)	-	Moderate
☐	General purpose	m3.large	2	7.5	1 x 32 (SSD)	-	Moderate
☐	General purpose	m3.xlarge	4	15	2 x 40 (SSD)	Yes	High
☑	General purpose	m3.2xlarge	8	30	2 x 80 (SSD)	Yes	High

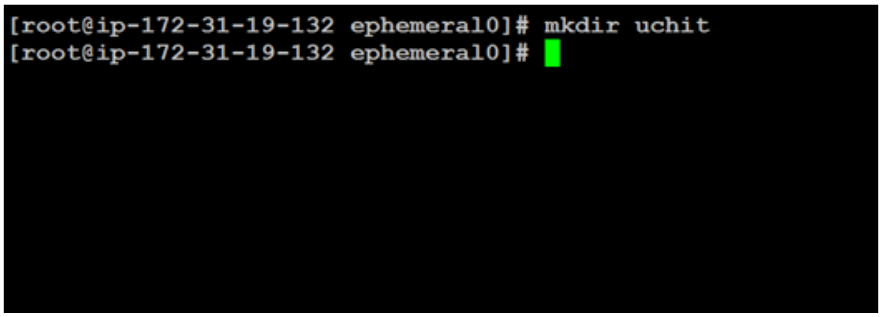
2. Log in with an instance and go to the `/mnt` or `/media` directory after checking the ephemeral storage using the following command. We get the following output:

```
# df -h
```

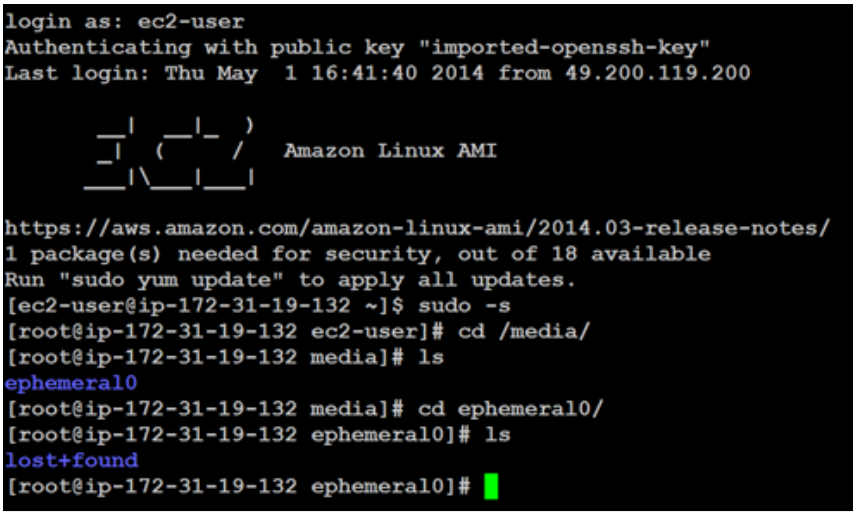
```
[root@ip-172-31-19-132 ec2-user]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/xvda1      7.8G  1.1G  6.7G  14% /
devtmpfs        813M   20K  813M   1% /dev
tmpfs           828M     0  828M   0% /dev/shm
/dev/xvdb       147G  188M  140G   1% /media/ephemeral0
[root@ip-172-31-19-132 ec2-user]#
```


www.allitebooks.com

3. Create any directory or file in the `/media` directory, as shown in the following screenshot:



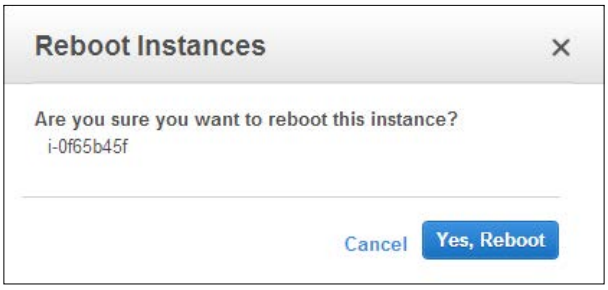
4. Now stop the instance and start it again. After starting the instance, check the `/media` directory for your content. You won't be able to see your content as we performed the stop and start operation with an instance:



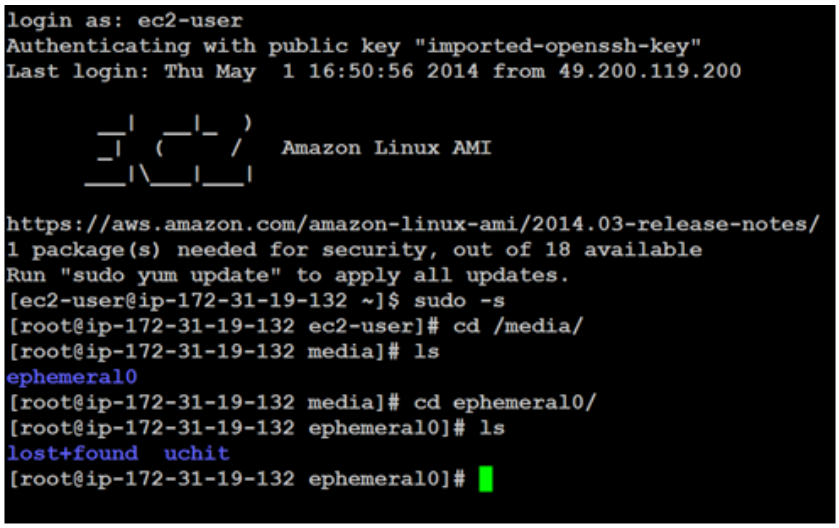
5. Now, let's check by rebooting the system after we have created some content in the `/media/ephemeral10` directory:

```
login as: ec2-user
Authenticating with public key "imported-openssh-key"
Last login: Thu May  1 16:41:40 2014 from 49.200.119.200

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2014.03-release-notes/
1 package(s) needed for security, out of 18 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-19-132 ~]$ sudo -s
[root@ip-172-31-19-132 ec2-user]# cd /media/
[root@ip-172-31-19-132 media]# ls
ephemeral0
[root@ip-172-31-19-132 media]# cd ephemeral0/
[root@ip-172-31-19-132 ephemeral0]# ls
lost+found
[root@ip-172-31-19-132 ephemeral0]# mkdir uchit
[root@ip-172-31-19-132 ephemeral0]# ls
lost+found  uchit
[root@ip-172-31-19-132 ephemeral0]#
```

6. Reboot your instance:



7. After the reboot process, check your content:



EC2 instances that use Amazon EBS for the root device do not, by default, have an instance store accessible at boot time. Also, you can't assign instance store volumes after you've launched an instance. Therefore, if you want your Amazon EBS-backed instance to use instance store volumes, you must postulate them using a block device mapping when you create your AMI or launch your instance. The block device mapping entries are `/dev/sdb=ephemeral0` and `/dev/sdc=ephemeral1`.

The following URL shows a list of ephemeral storage sizes with respect to the instance type of EC2:

`http://aws.amazon.com/ec2/instance-types/`

There are lots of theories around instances backed with EBS or instance storages concerning performance, costs, and so on. Therefore, I'd say the choice of which one to use rests on the user. Normally, you can find the subsequent facts:

1. An instance with ephemeral storage is faster than EBS for modest statistics which persistent is not.
2. You can't stop an instance in order to pay less for it; nevertheless, if you do, you will simply lose everything. So you have to create backup strategies.
3. You can't advance instance, that is you can't upgrade directly or scale vertically. So you will have to make an AMI and launch a bigger instance from the console.

And of course, there are numerous workarounds for this, but truly I would like to say that you should use these types of storages only when your application is currently designed to not store anything locally. One more thing, ensure that you have an EBS volume attached to your instance, which in turn will serve as backup or to store sensitive data that can be lost.

EBS is a persistent storage given by AWS. All data stored in persistent storage is presented after an instance is shut down and can function dynamically on device level. When you create and start an instance, the root device volume grips the image used to boot the instance. You can choose one option from the AMIs based on the Amazon EC2 instance store and the AMIs backed by Amazon EBS. Experts recommend that you use the AMIs backed by Amazon EBS because they launch faster and use persistent storage.

For example, you can detach an EBS volume from one instance and attach it to another instance. However, an EBS cannot be attached to more than one instance at the same time but multiple EBS can be assigned to one EC2 instance, and they can then be lined and/or emulated into a larger volume using **Redundant Array of Independent Disks** (**RAID**).

These two AMIs based on type reveal numerous differences, for example, in their life cycle, boot time and data persistence characteristics:

Characteristic name	**Amazon EBS-backed**	**Instance (S3) Store-backed**
Life cycle	Supports the stopping and restarting of an instance by saving the state to EBS	The instance cannot be stopped; it can either be in the running or terminated state
Data persistence	Data continues in EBS on instance failure or restart. Data can also be configured to persist when an instance is terminated.	Instance storage does not continue on instance shutdown or failure. It is possible to attach non-root devices using EBS.
Boot time	Usually less than 1 minute	Usually less than 5 minutes

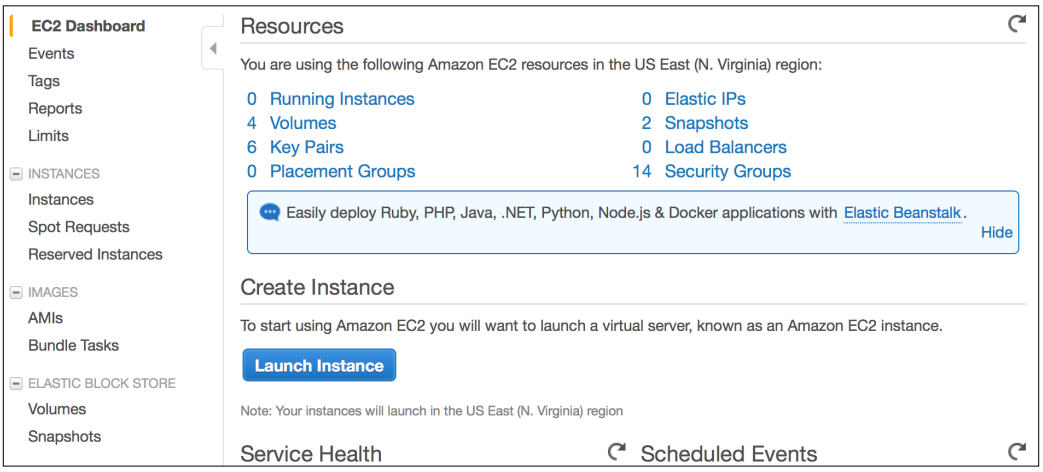
Table 1.3 – EBS-backed and instance store-backed comparison

How to use persistent storage with your instance

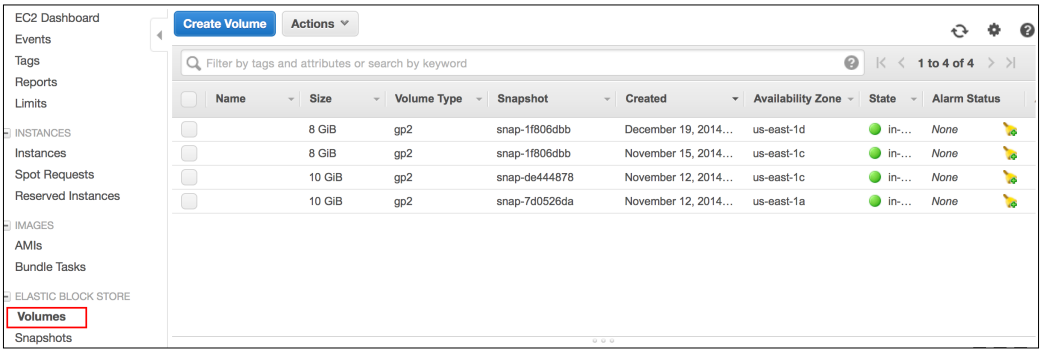
EBS is a mountable storage service; it can be mounted as a device (such as plug-and-play) to an EC2 instance.

Let's take an example to attach EBS (persistent storage) to your instance using the following steps:

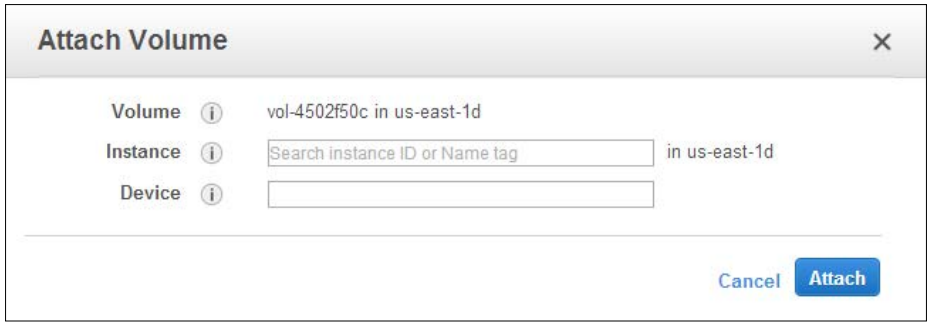
1. Go to the Amazon EC2 console, on the left-hand side of the navigation pane, you will find the **Elastic Block Store** section:



2. Click on **Volumes** in the navigation pane. The console displays a list of the current volumes:



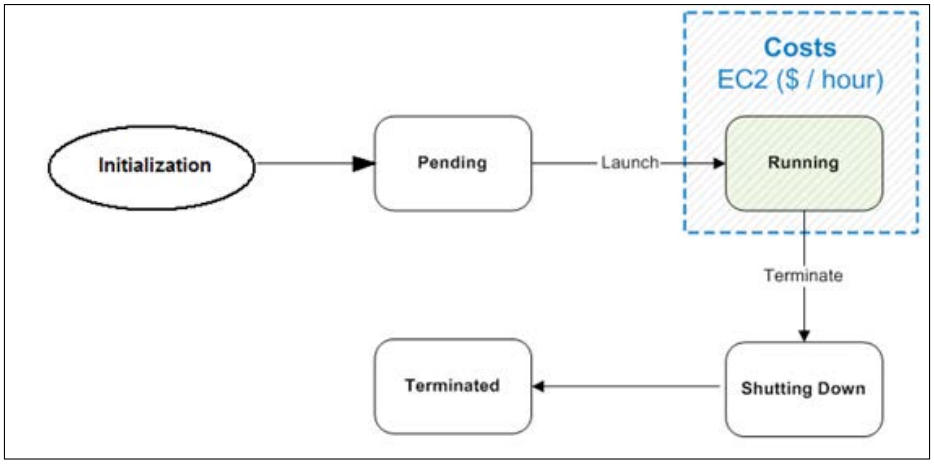
3. Select a volume and click on **Attach Volume**. Select an appropriate instance from the drop-down box. Only the instances in the same AZ as the volume will be displayed:



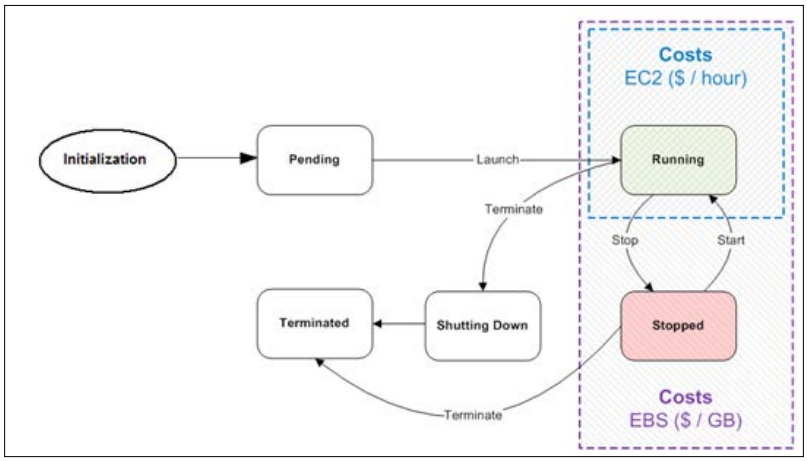
4. Fill the necessary details and click on the **Attach** button to attach the volume to the instance. The volume and instance must be in the same AZ.

Instances with Windows OS will use either Red Hat or Citrix **paravirtual** (**PV**) drivers. If you have a Windows instance with Citrix PV drivers, you can attach up to a total of 25 EBS volumes; however, Windows instances with Red Hat PV drivers are limited to 16 volumes only.

The following diagram summarizes the lifecycles of both persistent and non-persistent storages with S3 and EBS-backed EC2 instances:



EBS-backed EC2 instances present a new stopped state, which is not available in the backed-by-instance store as of now. It is essential to keep in mind that while an instance is in a stopped state, you will not experience any EC2 running charges except EBS storage charges, which are related to your instance:



The other benefit of S3-backed instances is that a stopped instance can be started another time while preserving its core state.

Scalability, elasticity, and bootstrapping

Generally, each and every Cloud is designed to gain infinite scalability. Even though the Cloud is scalable, you cannot take advantage of the scalability if your infrastructure is not scalable at any point. You have to conclude your components, which requires scalability, and you have to find those parts in which on-demand scaling won't work for your business. You have to design your application in order to get maximum output to capture the market and leverage the scalable infrastructure of the Cloud.

The characteristics of a proper scalable application are as follows:

- Growing capitals result in a comparative upsurge in the recital of performance
- A scalable service is accomplished because of conduct heterogeneity
- A climbable service is operationally effective
- An ascendable service is robust
- In general, a scalable service should get more budget with respect to actual one when it is required

These are characteristics that must be converted into an intrinsic part of the business application and if the architecture design is built with the preceding characteristics, then both the architecture and infrastructure will work in an organized manner to give you the scalability you are looking for.

As the provisioning time and upfront venture to obtain new capitals was too high, software architects never invested time and resources in augmenting hardware exploitation. It was tolerable if the hardware on which the apps run was underutilized. The concept of "elasticity" inside a design went unnoticed as the idea of having new capitals in transcripts was not conceivable.

Cloud modernizes the progression of obtaining the essential capitals; there is no requirement lengthier than to place preparations ahead of time and grip the unexploited hardware incarcerated. Instead, Cloud architects can demand whether they want mere proceedings before obtaining resources, captivating the benefit of the huge scale and rapid response time of the Cloud. The vice versa is relevant to reduce the unwanted resources when you don't need them.

Elasticity is one of the essential properties of Cloud computing in today's fast-paced demand-growing world. It is the power to scale computing capitals up and down in a straightforward manner and with nominal chafing. It is important to comprehend that elasticity will eventually initiate maximum of the paybacks of the Cloud. As a Cloud engineer, you need to adopt the concept of elasticity and implement it in your application architecture in order to get the best value from the Cloud and its services.

Bootstrap your instances

Design your instance deployment and let your instances ask you difficult questions about their existence during the boot, such as "Why am I created and what will be my role?"; each and every instance should have a specific part to play in the infrastructure and deployment environment such as database servers, replica servers, web servers, cache servers, and so on. So put your instances in a specific category with a specific role.

These role features can be passed to an instance when you boot or spin the instances from an AMI at runtime on air. At the time of booting the instance, we will download and configure the scripts and codes necessary as per the role requirement and work automatically as defined in the scripts.

The following are the advantages of bootstrapping your instances:

- Reconstruct the (development, test, and production) environment with little snaps and nominal struggle
- The preceding instances govern your abstract Cloud-based capitals

- Condense human-induced deployment mistakes
- Generates a self-healing and self-discoverable working environment, which can be more robust to hardware failure

Black belt booting

There are numerous cutting-edge methods that compromise supplementary power and flexibility when booting Linux. For example, some officialdoms preserve a sequence of the standard instances, and modify the ideas upon promotion, at the time of deployment. Communal practices include:

- Spontaneously checking for modernizers upon every boot.
- Staring at a well-known position, such as in an S3 (Simple Storage Service) bucket, for data or a script to direct the instance, which packages to load.
- Gives access of the user data to the instance to achieve each of the preceding goals, or perhaps, as an alternative to the other approaches.

Identity and Access Management

AWS has a shared security section that consist of AWS IAM. AWS IAM allows for the creation of distinct users or groups, granulated authorizations, and even precise services. Agreements can be set for any AWS service, including Amazon EC2, and letting fashionable security credentials that avoid basic users from salvaging statistics not related to their access. You can also use **Identity and Access Management** (**IAM**) to achieve security credentials such as access keys, passwords, and multi-factor authentication devices centrally. Active instantaneously, the IAM is a generally available service among other AWS services no; this means it doesn't depend on a region or Availability Zones!

IAM includes the following features:

- Complete control of users, groups, and security credentials
- Control of access based on user role, and security tokens
- Control over shared AWS resources
- Authorizations based on organizational groups and users
- Centralized networking controls

Accessing IAM

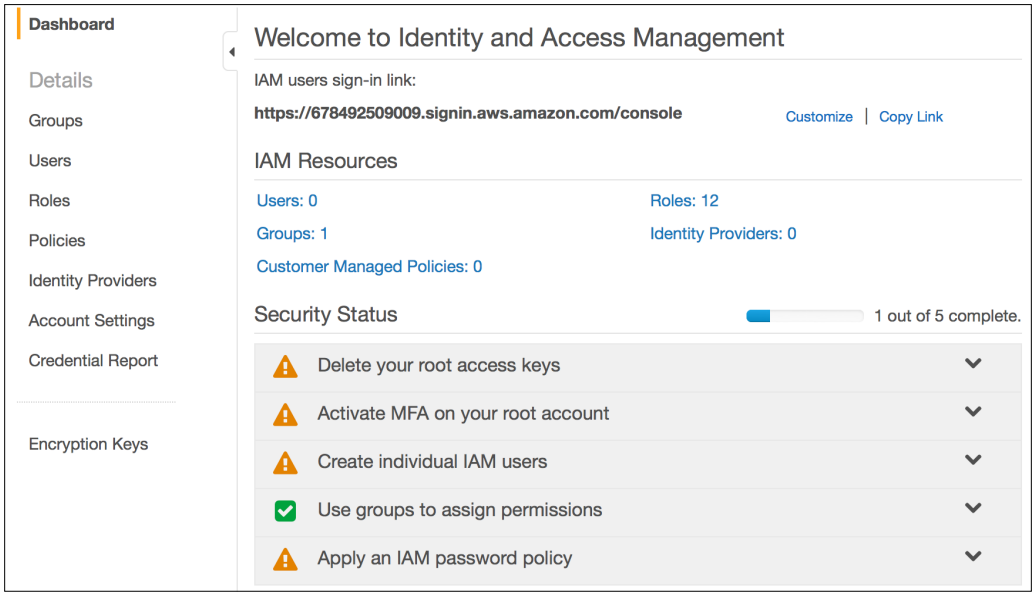
You can work with AWS Identity and Access Management (AWS IAM) by using the following methods:

- Using the AWS Management Console
- Using the AWS command-line interface
- Using the AWS API

Using any of the preceding access methods, one can accomplish IAM capitals, by:

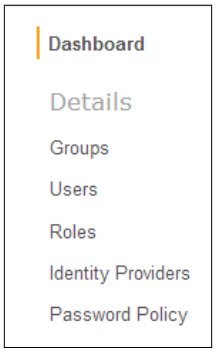
- Creating users/groups and assigning permissions to them
- Creating security credentials (roles and policies) for your users
- Assigning passwords to your users and restricting them for particular services

Here, to get familiar with AWS IAM and its usage, we will follow the first method for an overview and later in this book, we will look into IAM using CLI with other services. Let's start with a detailed overview of the IAM console:

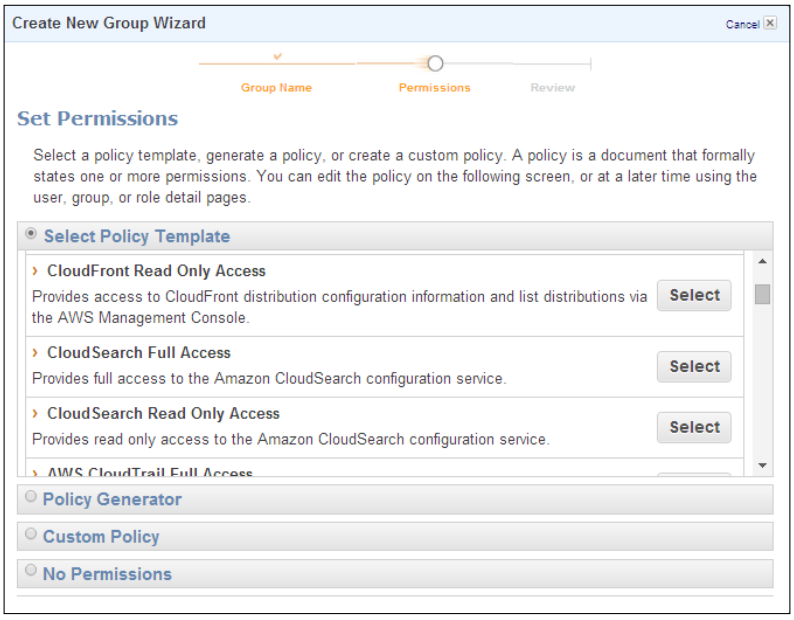


www.allitebooks.com

When a user signs in to their AWS account, they sign in via an IAM-enabled user sign-in page. For their accessibility, this sign-in page routes a cookie to evoke the user's position so that the next time a user serves to the AWS Management Console at the next login or visits the same page, the AWS Management Console calls the IAM-enabled user sign-in page automatically. This left-hand side navigation tab allows you to create and manage IAM users, groups of IAM users, their permissions, and their security credentials separately, along with other services:



You can select a policy template, which is predefined in the IAM service, or build your own custom policies using AWS Policy Generator. The permission wizard includes a specific template for every service that currently supports IAM to make it easy for you to get started and define policies:



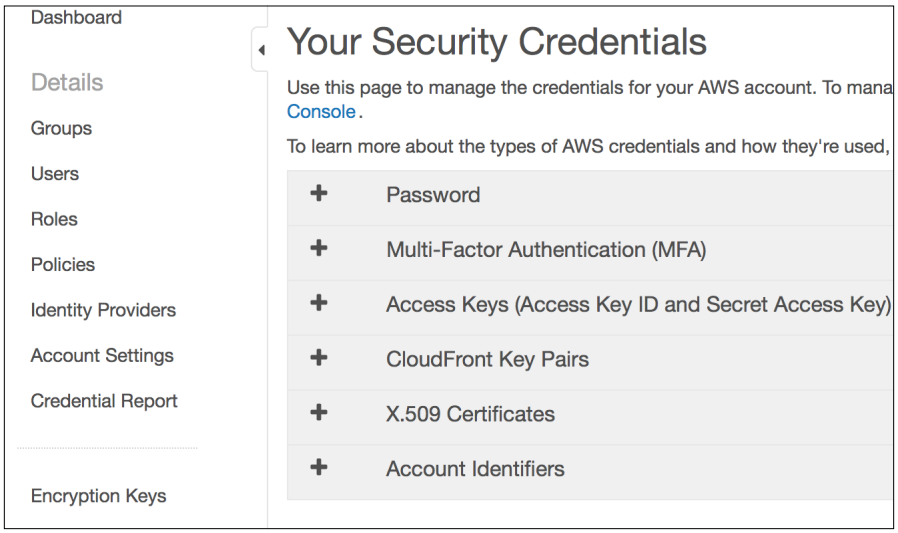
The preceding features described represent my first overview-based steps toward our long-term goals to learn about IAM and its best use cases in further chapters. However, we have a long journey ahead of us and I am looking for additional integrations, data access methods, and product based scenarios with AWS IAM.

Authentication and authorization

One of the characteristics that made me focus on AWS was my knowledge that the Cloud can be pleasant and logical in which you can figure security solutions.

Two imperative ideologies are essential to confirm that the correct people are undertaking the right things in every information system. These are as follows:

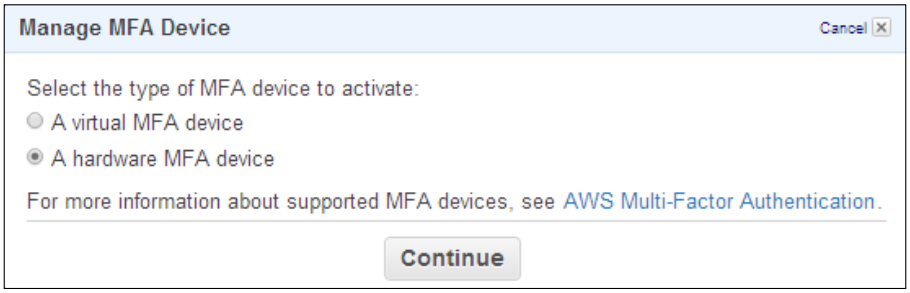
- Authentication: This is how you demonstrate your uniqueness. The computer won't accept your identity until you exhibit an acquaintance of an identity that the computer can then validate. Typically, it's your username and password; it could also be the private key (secret key and access key) associated with a digital certificate (here, X.509 in AWS). Authentication classifications never send secrets over the wire; in its place, enigmas are used to compute a difficult-to-reverse message. Since apparently only you know your secret, your claim is valid in any system.
- Authorization: This is what we're permitted to look after once the Cloud grants you access to services. Unfortunately, individuals aren't actually good at keeping secrets and often divulge secrets when they get an opportunity. The following figure shows the authorization services provided by AWS:



The **Multi-factor authentication** (**MFA**) device in AWS alleviates this base problem by demanding additional proof of the problem. Authentication factors come in many varieties:

- Something you know: A password, passphrase, key, pin, and response to a challenge.
- Something you have: A token, smartcard, mobile phone, passport, and wristband.
- Something you are or do: A meddle-resistant and theft-resistant biometric individual.

On AWS, for authentication, you can use two factor authentication devices: one for connection and the other for the retrieval of data. You can use secret keys, access keys, and X.509 certificates for authentication. You can find these under the `AWS Management Console/Security Credentials` option as described in the preceding screenshot. They also provide MFA, which can be used for two-factor authentication. The following screenshot is for the MFA device. There are two types of MFA devices: Virtual MFA and Hardware MFA.



At the following URL, you can purchase your MFA device if you want:
`http://onlinenoram.gemalto.com/`

There are two types of MFA devices available on this site, which differ in characteristics and pricing. As per your requirement, you can buy them and secure your AWS environment using two-factor authentication.

Summary

In this chapter, you learned about all the basics requirement of AWS. You also learned what AWS regions and Availability Zones are and about EC2 instances. In the EC2 section, you learned about instance types and pricing models. Later, you looked into persistent storage and ephemeral storages, and their life cycles. In the last section of the chapter, we covered what IAM is and its dashboard overview. Finally, you learned what authentication and authorization are with a high-level overview of the AWS security dashboard.

In the next chapter, you will learn how to create fault-tolerant applications with EC2, EBS, and the ELB. In this, you will dive deep into EC2 to learn about application availability and other components such as EBS and ELB along with how they work.

2
Elastic and Fault-tolerant Infrastructure

In normal business scenarios, while your internet applications are getting more traffic, you have to add more capacity either by increasing the number of servers or the size of the existing servers to handle those traffic spikes. Vice versa, if your web application traffic goes down or is normal, you have to shut down under-utilized servers or decrease the capacity of the existing servers. There are basically two types of scaling available:

- Horizontal scaling: Adding or removing the number of servers to handle traffic on demand
- Vertical scaling: Increasing the capacity of the existing servers or infrastructure (RAM, CPU, and storage)

The final decision on which scaling to use can depend on crucial factors such as cost, performance, and infrastructure.

In this chapter, we will cover the following topics:

- The AWS Elastic infrastructure by Auto Scaling
- How to use **Elastic Load Balancer** (**ELB**) with an EC2 instance

So, let's start with the AWS Elastic infrastructure.

The AWS Elastic infrastructure by Auto Scaling

For web applications configured to run on Cloud, scaling is a vital part of cost control and resource management. Auto Scaling is a web service mechanism provided by AWS to automate the process of launching and terminating AWS EC2 instances based on user-defined rules and scheduling with the health status check facility.

Auto Scaling can be applied for those applications that experience traffic on an hourly, daily, or weekly basis and that need to scale horizontally or vertically. Mostly on AWS, horizontal scaling is recommended, because to do vertical scaling with the Amazon infrastructure, you need to stop the instance and increase or decrease its capacity as per your requirement. When you are doing horizontal scaling, there is no need to stop/start your existing instances/servers. The Auto Scaling service frees you from managing your Cloud infrastructure based on your traffic spikes, accurately planning for resources in advance. So, by using Auto Scaling, you can build a fully elastic and scalable infrastructure affordably.

Let's look at an example of how scaling works. I have a blog called *Cloud Magic World* (`http://cloudbyuchit.wordpress.com`) that runs on a single EC2 small instance. An EC2 instance performs best when I have regular traffic on my blog at night. However, during the day, say from 10:00 am to 07:00 pm, the traffic to my blog increases by up to three times. When this happens, I need an additional Cloud instance to handle the increased load. For my blog application to scale in accordance with the additional load, I'll need to launch the second EC2 small instance before the increased load occurs, and I will terminate that server after the traffic goes down to normal levels. This process works best for me where the application has predictable traffic patterns; I know when to launch the extra instance and when to terminate it.

However, say, for example, that we are not aware about how big a traffic spike will hit my blog post. So, for my blog, if the traffic load is not predictable, I have to add a few more servers, although there won't be continuous traffic on those servers.

What happens in this example if I use Auto Scaling for my blog? The answer is very simple, I won't be charged for other extra instances and I don't need to run those extra instances all the time. Instead, I can define the conditions that determine the increasing traffic to my blog instance, and then I will tell Auto Scaling to launch a similar application server whenever those conditions are met and required.

In the same way, I can define policies for heavy-to-normal spikes, and then tell Auto Scaling to terminate a server when these conditions are met. AWS Auto Scaling allows you to scale resources in two ways:

- Dynamic scaling: This is based on conditions specified by you (for example, the number of users or CPU utilization).
- Predictive scaling: This is based on a schedule, which will be defined by the admin (for example, every day at 14:00:00 hours).

Working with Auto Scaling

For general web applications, such as online learning portals, one can add multiple instances for a single application to handle customer traffic over the globe. These numbers of copies of applications can be hosted on identical AWS EC2 instances to use the Auto Scaling service.

In Auto Scaling, your hosted EC2 instances on different AZs will be categorized into *Auto Scaling groups* for management and scaling activity. You can create Auto Scaling groups with different configuration parameters, such as minimum, maximum, or/and the number of EC2 instances in a particular group at any time.

In Auto Scaling groups, with the help of *launch configuration,* instances will be launched. You can create your own launch configuration with the AMI information, which will be used in Auto Scaling to launch instances. Image information would be image ID, type, security key pair, groups, and block devices. After creating a launch configuration and an Auto Scaling group, you have to create a *scaling plan* for the Auto Scaling group to define when and how to scale. One can create a scaling plan based on your requirement, such as using scheduling or using specified conditions for dynamic scaling.

Ways to access the Auto Scaling service

You can manage your Auto Scaling resources in a number of ways. AWS provides the following ways to manage Auto Scaling:

- **Auto Scaling Command Line Interface** (**CLI**): CLI silently uses the API actions to provide multifunction commands without using SDKs and APIs. The CLI commands are written in Python and include shell scripts for automation and management.
- **AWS Command Line Interface**: This can be used to organize and automate AWS services on Windows, Mac, and Linux.

- **The Auto Scaling API**: Auto Scaling uses APIs, which you can call by submitting a query request, and it is the fanciest way to access a web service directly.
- **SDKs for Auto Scaling**: This provides functions that bind an API and take care of the connection details using Java, .NET, PHP, Node.JS, Android, iOS, or the Ruby language.
- **AWS Management Console**: This is a web-based interface that you can use to manage and access Auto Scaling and other AWS services.

In this chapter, we will configure and access Auto Scaling via AWS Management Console and AWS Command Line Interface.

The limitation for Auto Scaling is that there is a default limit of 20 Auto Scaling groups and 100 launch configurations per region per AWS account. If you reach the limit, you can go to the support center at `https://aws.amazon.com/support/`, and request to raise this limit.

Installing and configuring Auto Scaling

The Auto Scaling utility can be accessed via CLI to access the Auto Scaling functionality without any of the preceding ways mentioned. It wraps the API actions to give multi-utility commands.

Installing Auto Scaling prerequisites

Proceed with the following steps to install and configure Auto Scaling Command Line Interface:

1. Set the Java variable in your system:
    1. The command-line tool reads an environment variable (`JAVA_HOME`) on the machine to locate the Java runtime. Either JRE or JDK should be fine with Version 6 or higher. To download Java, go to `http://java.oracle.com/`.
    2. Extract your downloaded Java and set the path using the variable `JAVA_HOME` to the full path of the directory, which will contain the `bin` subdirectory. For example, if you have Java installed in the `/opt` directory, the path should be `JAVA_HOME` to `/opt/jdk` for Linux and `C:\jdk` for Windows.

3. To set `JAVA_HOME`, use the following commands:

   For Linux:

   ```
   $export JAVA_HOME=/opt/jre
   ```

   For Windows:

   ```
   C:\> set JAVA_HOME=C:\java\jdk1.6.0_6
   ```

4. Include the Java directory to your system path before other versions of Java:

   For Linux:

   ```
   $export PATH=$PATH;$JAVA_HOME/bin
   ```

   For Windows:

   ```
   C:\> set PATH=%PATH%;%JAVA_HOME%\bin
   ```

5. Verify your `JAVA_HOME` settings:

   For Linux:

   ```
   $ JAVA_HOME/bin/java -version
   ```

   For Windows:

   ```
   C:\> %JAVA_HOME%\bin\java -version
   ```

2. Setting up the CLI tools: To access the Auto Scaling Command Line tools, you need to download it from the AWS site `http://aws.amazon.com/cli/` and set it up with your AWS credentials within an instance. You need to just download it and unzip it. No installation is required; it will come as a `.zip` bundle.

   1. The CLI also depends on the environment variable, so, again, we have to set its path with `AWS_AUTO_SCALING_HOME`:

      For Linux:

      ```
      $ export AWS_AUTO_SCALING_HOME=/usr/local/as-1.0.12.0

      $ export PATH=$PATH:$AWS_AUTO_SCALING_HOME/bin
      ```

      For Windows:

      ```
      C:\> set AWS_AUTO_SCALING_HOME=C:\CLIs\as-1.0.12.0
      C:\> set PATH=%PATH%;%AWS_AUTO_SCALING_HOME%\bin
      ```


www.allitebooks.com

2. The way to use Python for the AWS CLI setup is very easy and became very popular in the developer community as it requires only a single command called `pip`. It will install AWS CLI tools on your instance. So, the same can be possible for AWS Auto Scaling tools.

   ```
   pip install awscli
   ```

Your environment variables of the Windows machine may reset when you close the terminal window. You may want to set them undyingly using the `setx` command, which is the same as `set`.

3. Authenticate your AWS account with the CLI tools:

   1. After signing in, you need to create access keys and secret keys for your account. You have to provide these keys to your CLI tools. You can create your AWS secret keys and access ID from `https://console.aws.amazon.com/iam/home?#security_credential`.
   2. Create one new file called `CredentialFile` and save your access key ID and secret access key to the file.
   3. Provide `600` (read and write for owner only) permission to `CredentialFile` if you are a Linux user using the following command:

      ```
      $ chmod 600 CredentialFile
      ```

   4. Set the `AWS_CREDENTIAL_FILE` variable based on your file location.

Now, you are almost done with setting up the AWS CLI tools for Auto Scaling. So, now you can test the tools on a Windows or Linux machine using the following command:

```
as-cmd
```

You should see the output as follows, for your reference:

```
Command Name                                    Description
------------                                      -----------
as-create-auto-scaling-group                    Create a new Auto Scaling
group.
as-create-launch-config                         Creates a new launch
configuration.
```

```
as-create-or-update-tags                    Create or update tags.
as-delete-auto-scaling-group                Deletes the specified
Auto Scaling group.
as-delete-launch-config                     Deletes the specified
launch configuration.
as-delete-notification-configuration        Deletes the specified
notification configuration.
as-delete-policy                              Deletes the specified
policy.
as-delete-scheduled-action                  Deletes the specified
scheduled action.
as-delete-tags                                Delete the specified
tags
as-describe-adjustment-types                Describes all policy
adjustment types.
as-describe-auto-scaling-groups             Describes the specified
Auto Scaling groups.
as-describe-auto-scaling-instances          Describes the specified
Auto Scaling instances.
as-describe-auto-scaling-notification-types  Describes all Auto Scaling
notification types.
as-describe-launch-configs                  Describes the specified
launch configurations.
as-describe-metric-collection-types         Describes all metric
colle... metric granularity types.
as-describe-notification-configurations     Describes all
notification...given Auto Scaling groups.
as-describe-policies                          Describes the specified
policies.
as-describe-process-types                   Describes all Auto
Scaling process types.
as-describe-scaling-activities              Describes a set of
activities belonging to a group.
as-describe-scheduled-actions               Describes the specified
scheduled actions.
as-describe-tags                              Describes tags
as-describe-termination-policy-types        Describes all Auto Scaling
termination policy types.
as-disable-metrics-collection               Disables collection of Auto
Scaling group metrics.
```

```
as-enable-metrics-collection                    Enables collection of Auto
Scaling group metrics.
as-execute-policy                                  Executes the specified
policy.
as-put-notification-configuration               Creates or replaces
notifi...or the Auto Scaling group.
as-put-scaling-policy                         Creates or updates an Auto
Scaling policy.
as-put-scheduled-update-group-action            Creates or updates a
scheduled update group action.
as-resume-processes                            Resumes all suspended
Auto... given Auto Scaling group.
as-set-desired-capacity                          Sets the desired capacity
of the Auto Scaling group.
as-set-instance-health                              Sets the health of the
instance.
as-suspend-processes                             Suspends all Auto Scaling
... given Auto Scaling group.
as-terminate-instance-in-auto-scaling-group   Terminates a given
instance.
as-update-auto-scaling-group                     Updates the specified
Auto Scaling group.
help
version                                          Prints the version of the
CLI tool and the API.
For help on a specific command, type 'commandname --help'
```

If you are getting the preceding output, you are done with the setup and are now ready to access Auto Scaling using CLI.

Working with Auto Scaling using the CLI

You will understand the procedure of creating your basic Auto Scaling infrastructure within EC2-Classic or the default **Virtual Private Cloud** (**VPC**) using the Auto Scaling command line interface using the upcoming example. So, by now, you should be able to understand that to start with Auto Scaling, you need to create:

1. The launch configuration
2. An Auto Scaling group

The launch configuration determines the model that Auto Scaling uses to initiate Amazon EC2 instances for your applications. This template will have all the required data for the Auto Scaling configuration, which will be required to launch the EC2 instances that have your apps.

The `as-create-launch-config` command can take the following arguments:

Name	Image-id
Instance-type	Associate-public-ip-address
Spot-price	IAM-instance-profile
Block-device-mapping	EBS-optimized
Monitoring-enabled	Kernel
Key	User-data
User-data-file	Ram disk
Group	Other general option

Table 2.0-as-create-launch-config arguments

To start the Auto Scaling launch configuration, you have to perform the following:

1. Open your terminal and run the following command:

```
as-create-launch-config Uchit-test --image-id ami-0078da69
--instance-type m1.small
```

2. You should get the following output, if you passed the request successfully:

```
OK-Created launch config
```

Now, you are done with launching the configuration called `Uchit-test`, which launches an "m1.small" instance using the `ami-0078da69` AMI.

Auto Scaling groups are very important for Auto Scaling because they are a collection of EC2 instances. It is important because billing is concerned with this since you will define numbers for instances here!

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

To create an Auto Scaling group, run the following CLI command:

```
as-create-auto-scaling-group
```

This command needs your Auto Scaling group, a launch configuration, one or more AZs, a minimum group size, and maximum group size details as arguments.

As of now, for this demo launch configuration, I am going to specify the following options:

- Auto Scaling group name: `uchit-test-asg`
- Launch configuration name: `uchit-test`
- Optional: Availability Zone: `us-east-1a`
- Minimum size: `1`
- Maximum size: `5`
- Desired capacity: `1`

Enter the following command to launch your Auto Scaling group within EC2-Classic:

```
as-create-auto-scaling-group uchit-test-asg --launch-configuration uchit-test --availability-zones us-east-1a --min-size 1 --max-size 5 --desired-capacity 1
```

If you are going to launch the Auto Scaling group in default VPC, there is no need to specify the Availability Zone.

If your command runs successfully, you will get an output like the following:

```
OK-Created AutoScalingGroup
```

With the reference of your commands to the preceding arguments, the `uchit-test-asg` Auto Scaling group and the `uchit-test` launch configuration, Auto Scaling will launch one EC2 instance in the `us-east-1a` AZ that has type m1.small.

Use the following command to verify the Auto Scaling instances setup:

```
as-describe-auto-scaling-instances --headers
```

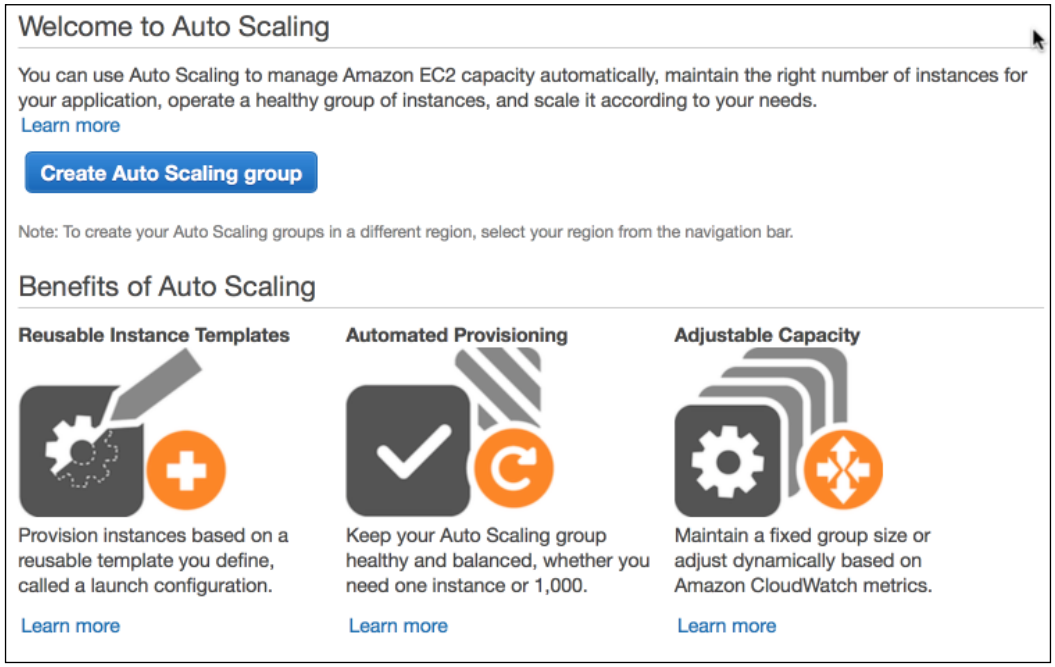
Also, one can check the Auto Scaling groups creation using the following command:

```
as-describe-auto-scaling-groups uchit-test-asg
--headers
```

In this way, you can configure and launch your elastic applications using AWS Auto Scaling services from CLI directly, without touching the AWS Management Console. It is best suited for developers to launch and configure their application from CLI or any IDE. Now, you will see how to configure the Auto Scaling service from AWS Management Console for your web application to get high availability and handle terrific traffic load.

Getting started with Auto Scaling using AWS Management Console

Using AWS Management Console, you can launch configurations and Auto Scaling using a single mouse click, and you can also bid for spot instances within this console when required.



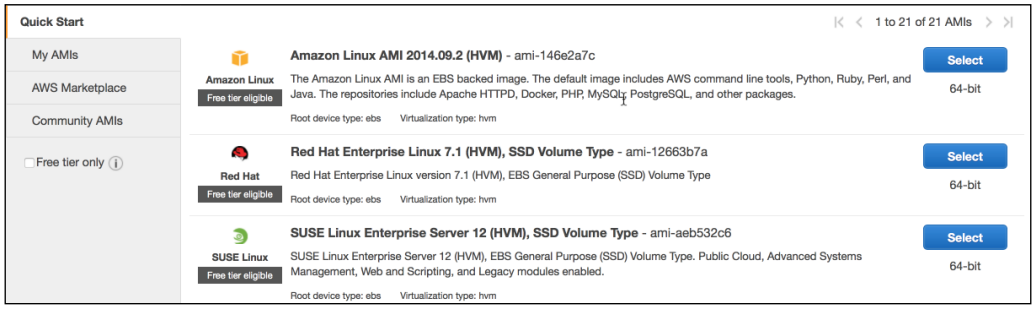
The welcome page shows some major benefits, such as the following:

- Fully automated provisioning
- Reuse instance templates
- Capacity adjustment

To start with the configuration, please follow the ensuing steps carefully:

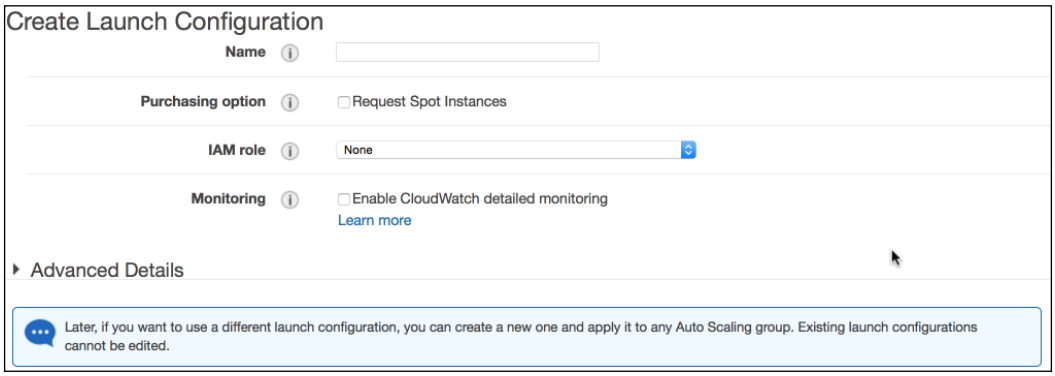
1. Creating launch configuration.

So, here we have to select the AMI and its relevant architecture, which suits our application.

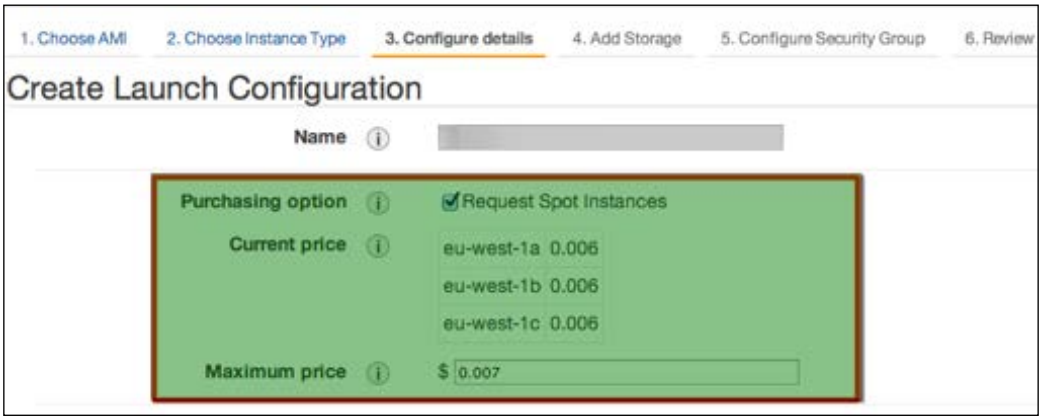


You can select your desired AMI from the given options from AWS. Here, you have to give the instance type as per your requirement. For example, you can choose m1.small, t1.micro, and so on.

2. Provide configuration details:

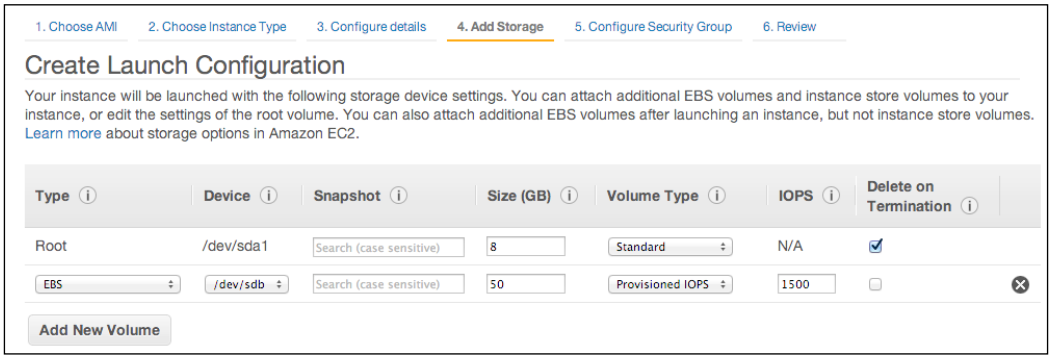


In the preceding screenshot, you can provide information in the **Advanced Details** section; it lets you choose AMI's **Kernel ID** and **Ram Disk ID** and optionally fill in a **User data** block. **User Data** can be regular data to be read by an application or an executable bash script, which will be run by an EC2 instance on its start. You can provide spot instance bid details, if you want:

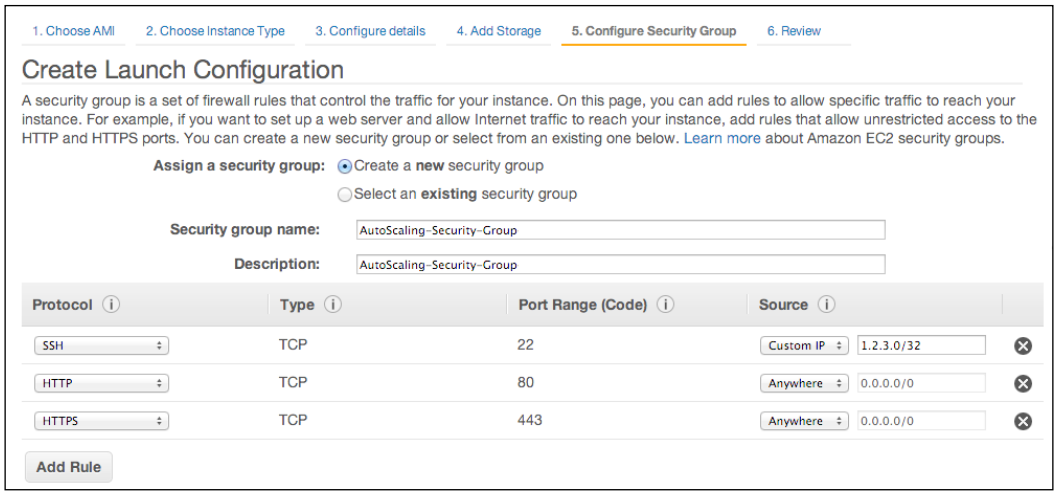


Spot instance bid details

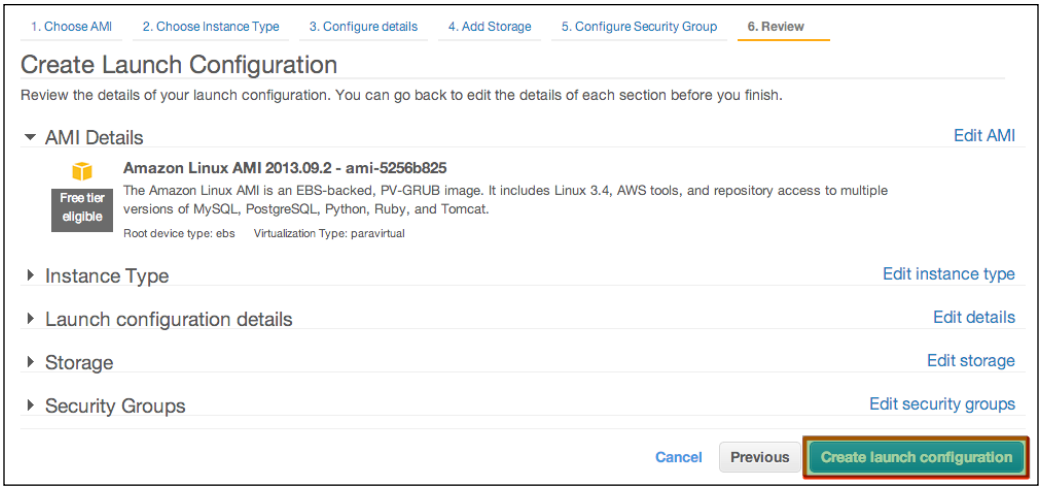
3. Provide your necessary storage options in the form of the root device size, EBS volume size, and so on.



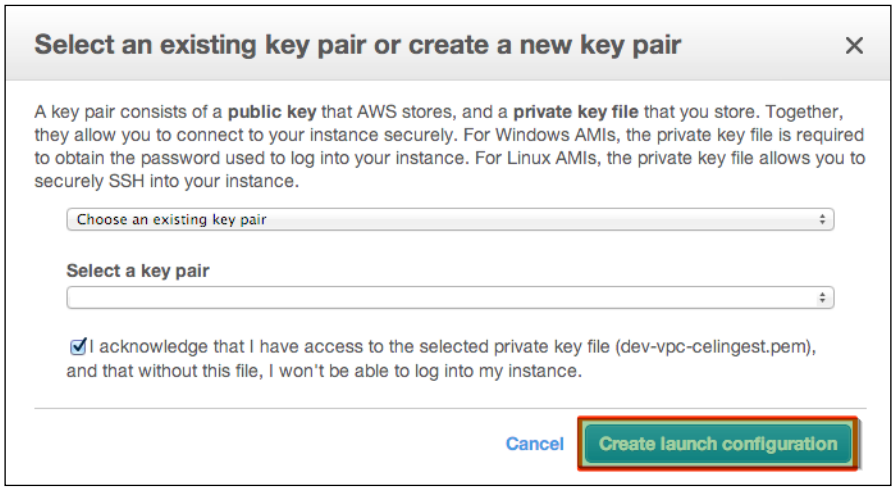
4. You can select an appropriate security group in the **Configure Security Group** tab, or just add a new customized one, as shown in the following screenshot, where you restrict the **SSH** access to your IP while leaving the **HTTP** and **HTTPS** access open to be used from anywhere around the globe:



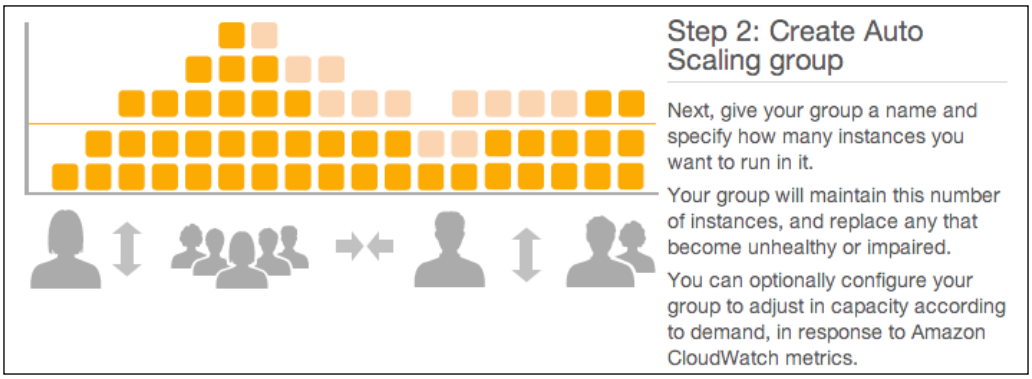
5. Finally, the last tab allows you to review and eventually change previous choices, if you need to:



Lastly, you have to choose the key pair to access your instances and click on **Create launch configuration**:

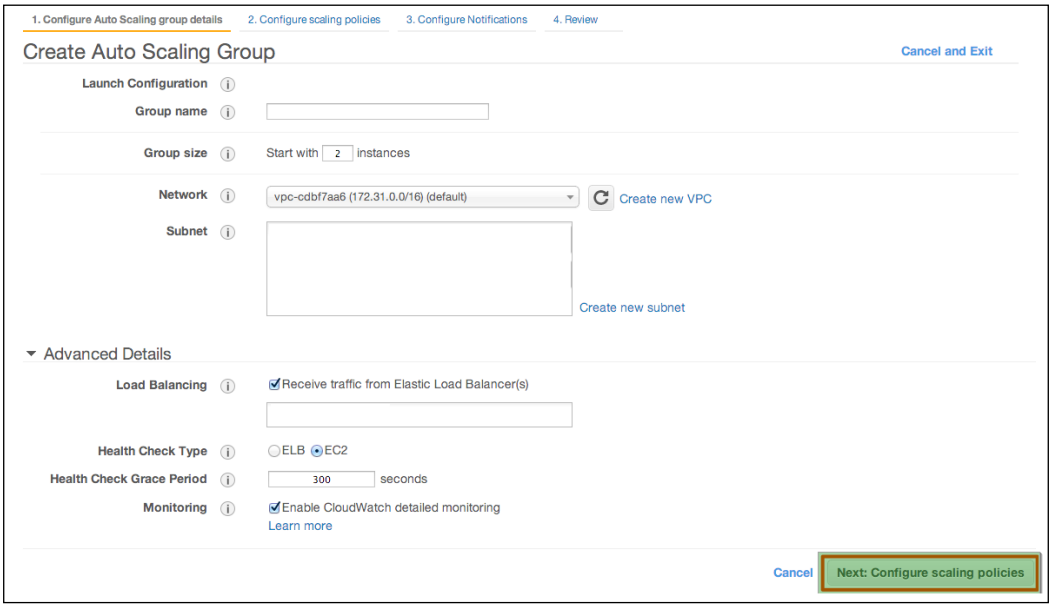


You are now done with the launch configuration for the AWS Auto Scaling service, but you have to configure the Auto Scaling group based on a schedule or policy. So, let's create an Auto Scaling group for your web application.



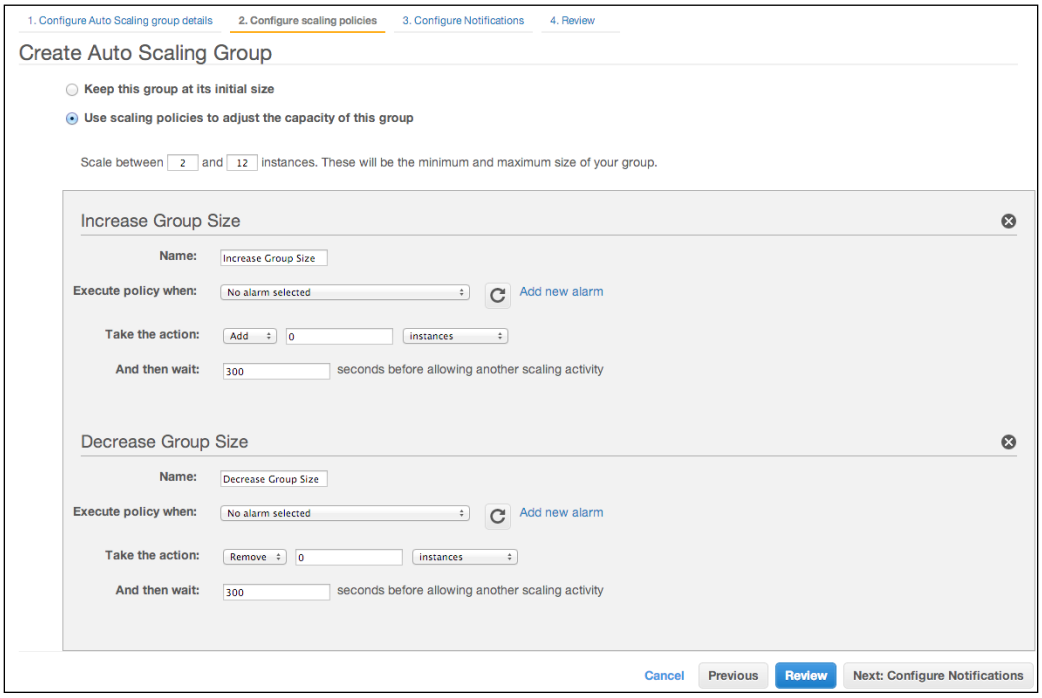
www.allitebooks.com

1. In the very first tab, you have a few choices, for example, group name, size, network, subnet, and so on, with some advanced details. In advanced details, you can call AWS Load Balancer, which will route the traffic based on policies to your scaled instances. Fill the appropriate details and click on **Next: Configure scaling policies**.

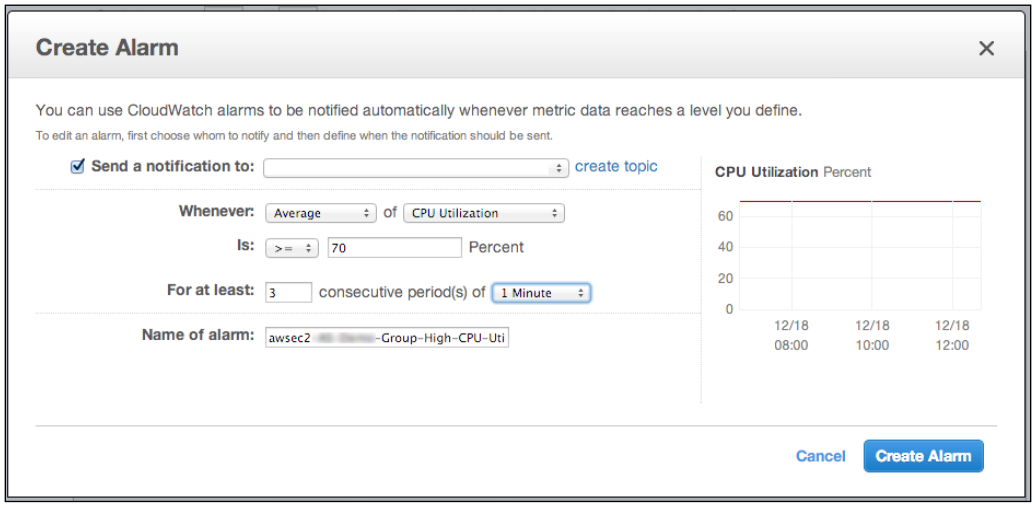


2. The next tab is very important as it allows you to select between two fundamental scaling plans. Here, there are two options:
    - **Keep this group at its initial size**, which will ensure that your group figures out a number of healthy instances equal to the initial size you mentioned. At any point of time, if an instance fails the checks, it will be replaced automatically.

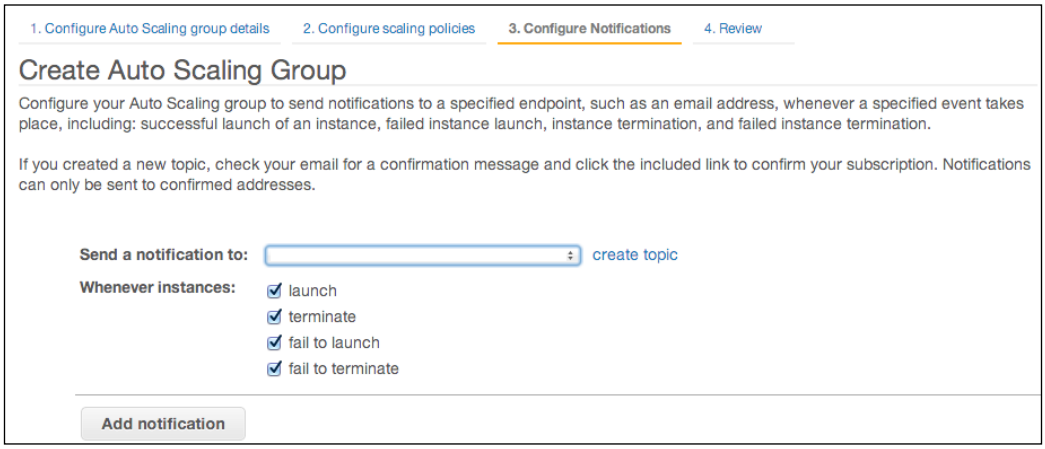
- **Use scaling policies to adjust the capacity of this group**. You can select the **CloudWatch** alarm based on some policies (for example, add 1 instance if CPU usage is more than 60% or remove 2 instances if CPU usage is less than 40%) to increase or decrease the number of instances.



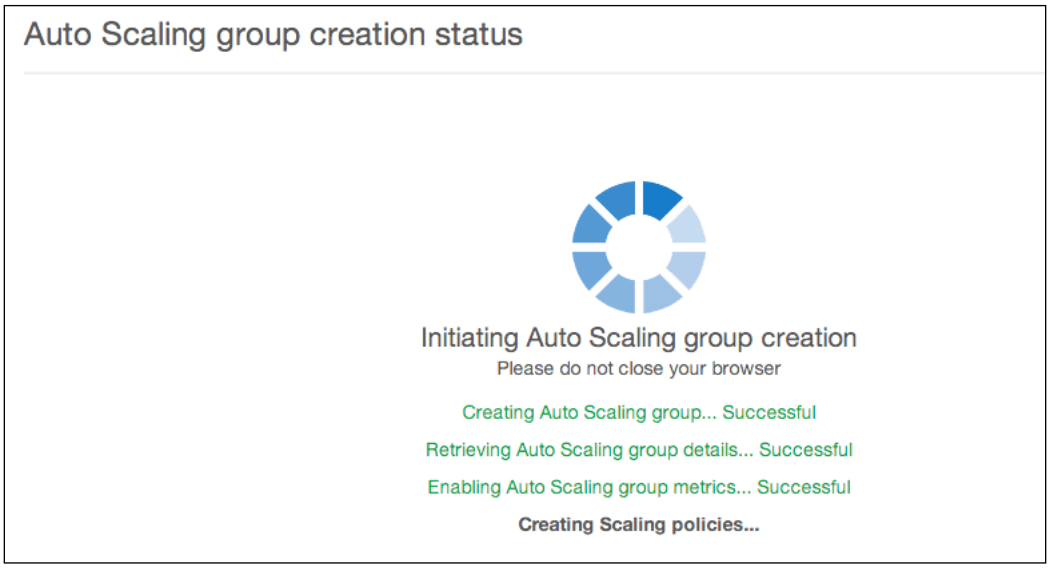
3. You can set policies for the minimum and maximum number of instances initiating your Auto Scaling group. You can add CloudWatch alarms easily after some time, or during the middle of deployment, but, here, we'll add the one to allow your group to increase its size, as shown in the following screenshot; you can also configure to decrease the group size.



4. You are almost done, so now it's time to configure the notifications in the **Configure Notifications** tab using **Simple Notification Service** (**SNS**) relative to your group to get updates:



5. Finally, the **Review** tab will be there to review the configuration, and after clicking on the **Create Auto Scaling Group** button, you will be redirected to a status screen showing the creation of your resources. If something fails, you're prompted to retry the single resource initiation again with correction.



So, after initiating, you can see the new scaling activity on your EC2 dashboard. And, whenever required, you can make changes in these configurations.

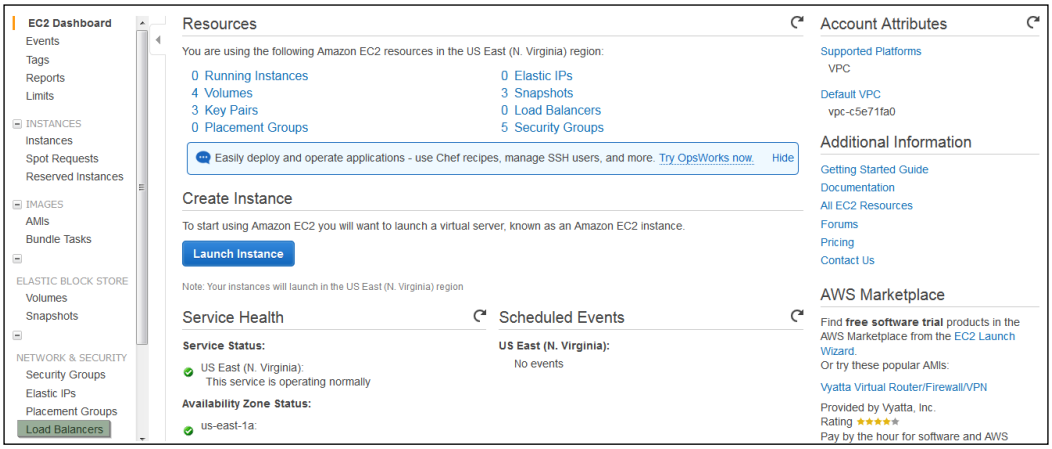
Here, you are done with Auto Scaling via GUI and CLI. So, now, you will see how we can configure AWS ELB with our instances to balance and route the load as per instance health check and utilization.

To start with ELB, you have to make sure that you create your ELB in the same region in which you have your EC2 instances created. I am going to list the summary steps from which you can easily understand the flow to launch ELB from your AWS Management Console. The following steps outline how to create a basic ELB in EC2-Classic:

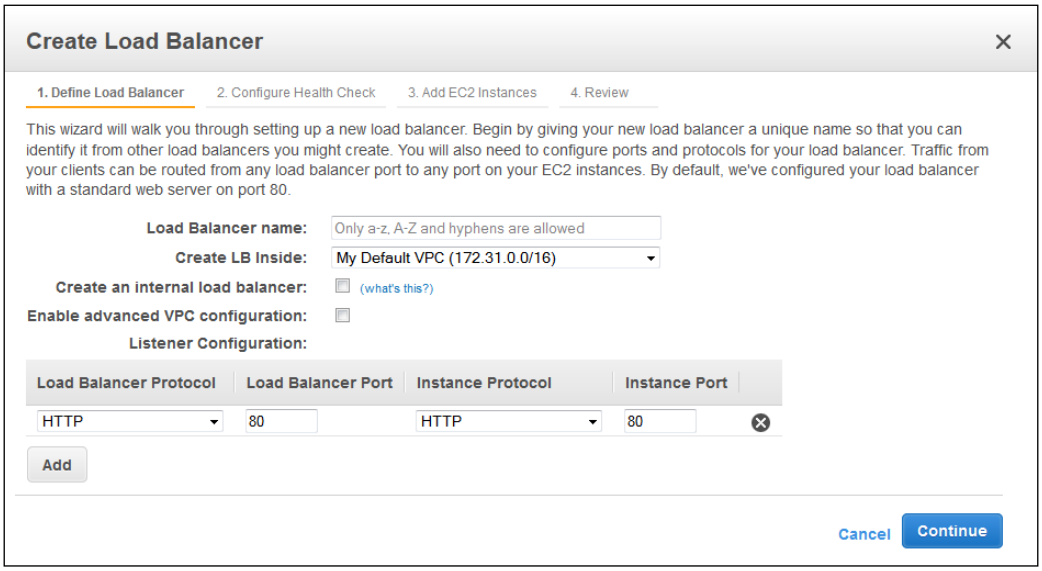
1. Configure the listeners.
2. Configure a health check.
3. Review and create load balancer.
4. Register instances with the load balancer.
5. Verify that your ELB is created and working.

Let's look at each step one by one and see how they can be configured.

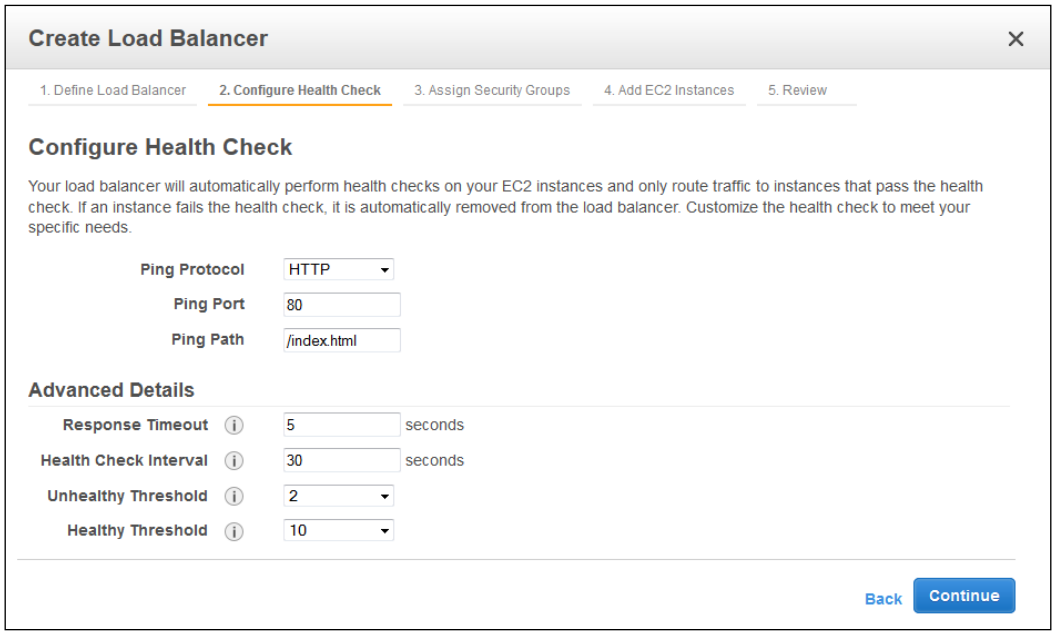
1. To configure the listeners, start your ELB wizard and click on **Create Load Balancer**:



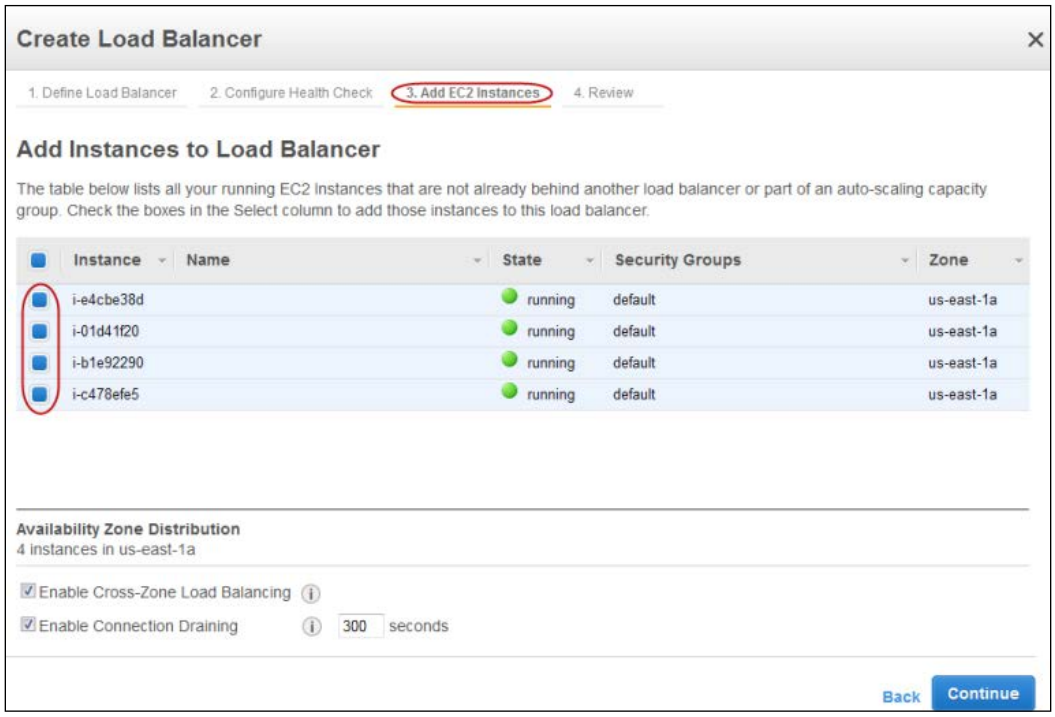
2. You will be navigated to the first tab, which is the **Define Load Balancer** tab. Here, you have to set **Load Balancer name**, description, and, most importantly, the port numbers. So, based on your port number configuration, the ELB will route the traffic to the given port for your application.



3. After clicking on the **Continue** button, you will be taken to the next tab called **Configure Health Check**. So, on this tab, you have to define your ping path and port number on which the ELB will ping to check the status. If the given consecutive count for unhealthy or healthy numbers is set and your instance does not match with that, the ELB will take action like to remove or add instance back again into ELB. So, it will check your instance health based on the defined interval.



4. After clicking on the **Continue** button, you will be taken to **Assign Security Groups** to help your desired security groups communicate with your instances. Finally, the important **Add EC2 Instances** tab will be there on your screen in which you will add your instances that will be under ELB.



5. Finally, the **Review** tab will open to check the configurations made. You can see the load balancer on the load balancer wizard, and it will take some time to populate your configurations and register your EC2 instances as well. You can even configure your third-party SSL certificates with the AWS ELB at the time of creation or later. Moreover, based on your Auto Scaling, you can define your load balancer in it. At the time of creation of the Auto Scaling group, you can define your load balancer, if you want.

6. If you are configuring your Auto Scaling using CLI, you can define your ELB as an argument, so that it will automatically create a load balancer for you, and later on, you can configure the necessary route details, or you can make configurations that your ELB can communicate with EC2 instances at the same time.

Summary

In this chapter, we discussed how by using Auto Scaling and ELB services, you can set up your web application with EC2 instances and scale them based on your requirements. Using Auto Scaling and ELB, you won't face any downtime for your applications, and that's the power of Auto Scaling on AWS, which is awesome!

In the next chapter, you will see the overall architecture of AWS, with a focus on breaking down the problem into discrete systems, and deciding how to physically separate those systems. You will also learn practical setup instructions for SDK and IDE toolkits, which can be used during programming with AWS services.

3
Storage Lifecycle Management

At the time of development, many storage systems are very similar to linear storage systems, but if you want to upgrade an application or its storage, for example, an application needs to be scaled with additional capacity which is more focused on supporting a large number of users with large file capacity and more attached data servers, it may become difficult to upgrade, as increasing the number of servers requires additional storage resources. In this chapter, we'll go through the following topics:

- Data storage scaling
- AWS DynamoDB
- AWS **Simple Storage Service** (**S3**)
- AWS CloudFront

So let's start with the basics of data storage scaling.

Data storage scaling

A storage system is one that uses a scaling method to create a dynamic storage environment that will support stable data growth whenever required to scale. Storage scaling means increasing or decreasing resources for a particular application, which includes two major categories: vertical scaling (scale-up) and horizontal scaling (scale-out).

If we add resources to a single machine such as adding CPUs or memory, it's called scale-up storage. The problem with this storage is that the storage capacity is added to the existing machine, but the bandwidth and computer power provided for that machine will be the same so it degrades the performance. To overcome this problem, scale-out storage systems are used, in that individual storage components (nodes) are added to the system. Each of these nodes contain capacity, computer power, and storage I/O bandwidth. A node added to the storage system comprising of these three resource in the system will be raised at the same time for extra storage requirement. Removing resources from a single machine or removing a node from the server is called scale-down storage.

So let's look at how AWS storage services are helpful to data, scaling, and their behavior.

AWS DynamoDB

DynamoDB is a fully accomplished NoSQL database with a spotlight on scalability, reliability, and performance. The data and traffic will be distributed to the table globally over multiple servers to serve the load described by the customer to provide consistent and fast performance, and durability. Data items will be stored on a **Solid State Disk** (**SSD**) with replication over multiple AZs. DynamoDB has generated a lot of enthusiasm for the obvious reason that Amazon is an authoritative figure in the NoSQL space now.

In RDBMS, a table is organized as rows and columns, but in DynamoDB, we will never use these (rows and column) words, except in this paragraph. Even if it is used mistakenly, please note that rows are items and columns are attributes.

Do you know that whenever you build a DynamoDB table, an index is generated automatically? This index is titled as the primary index. The primary index will contain primary key attributes (both hash and range keys). The index produced using hash keys is an unordered hash index. It measures the items in the table with equivalent hash keys that will be grouped together and arranged adjacent to each other, which benefits the retrieval of items with identical hash key attributes faster (using the scan operation). However, there will not be any assembling in the item on the hash key attribute.

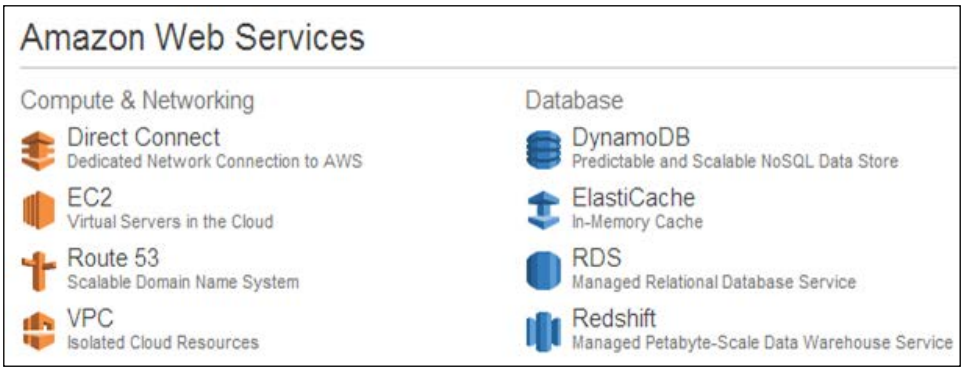
Throughout the creation of the DynamoDB table, it is better to postulate the secondary index attributes, hash and range key attributes. It is not conceivable to stipulate other attributes (mentioned earlier as optional attributes) throughout the creation of the table. In fact, excluding hash and range key attributes, all supplementary attributes are part of the items (rows). This is why we don't postulate these optional attributes while producing the table. Let's look at an example for AWS DynamoDB to learn its inner workings using Eclipse with reference to Java. You can use any language listed on AWS:

1. Take the AWS toolkit for Eclipse from `http://aws.amazon.com/eclipse/` and follow the setup steps, which are given on the download page. These tools are mostly a combination of preconfigured templates to construct apps.
2. Next, you have to specify a credential file that you fashioned in the preceding chapter for Auto Scaling. You can practice Auto Scaling it in the unchanged file here. Use the following code to set credentials:

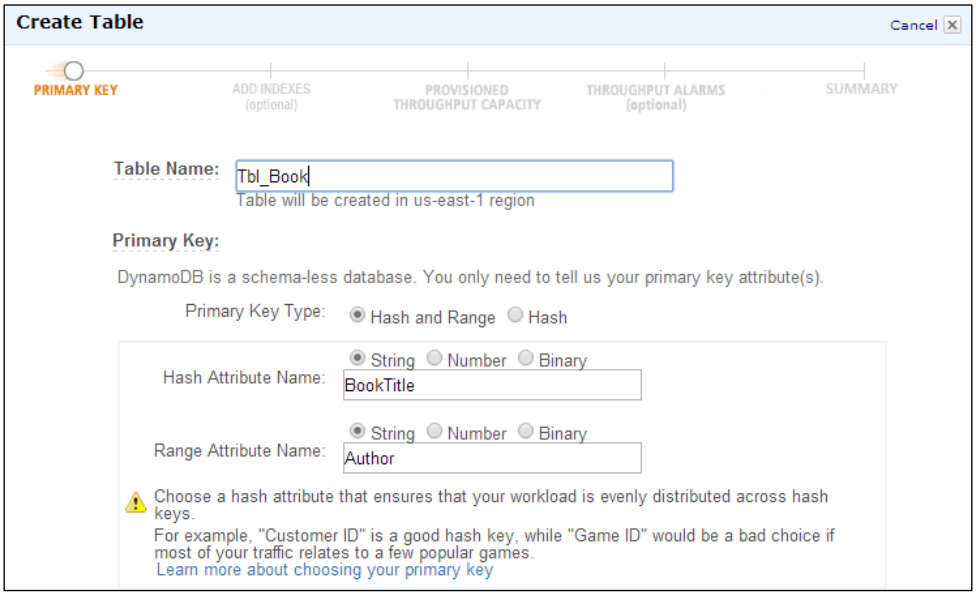
```
[default]
aws_access_key_id = <Your Access Key ID>
aws_secret_access_key = <Your Secret Key>
```

DynamoDB data types

DynamoDB provides six data types which are: String, Number, Binary, StringSet, NumberSet, and BinarySet. To comprehend it better, we will get help from the AWS Management Console. Once we have signed up with AWS, in our Management Console will appear the **DynamoDB** icon under the **Database** section, presented as follows:



Clicking on the DynamoDB icon the first time will take us to the startup page that has procedures to start with DynamoDB. We should click on the Create Table icon to create the first table. We'll then get the following window:



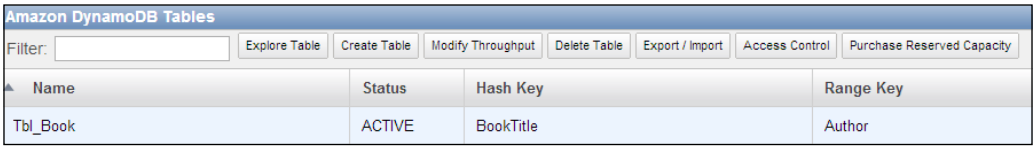
Now, we will create the `Tbl_Book` table. We will insert only one item into the `Tbl_Book` table. The table and its items are as follows:

BookTitle	Author	Publisher	PubDate	Language	Edition
String (hash)	String (range)	String	String	StringSet	Number
`SCJP`	`Kathy`	`TMH`	`28-Dec-09`	`{"English", "German"}`	`1`

As discussed, during the formation of the table, we need to stipulate only the primary key attributes alongside the table name. In this table, both the key foundations are of type String.

If we need to create a naive hash primary key (without range key), then we can choose the **Hash** radio button instead of the **Hash and Range** button.

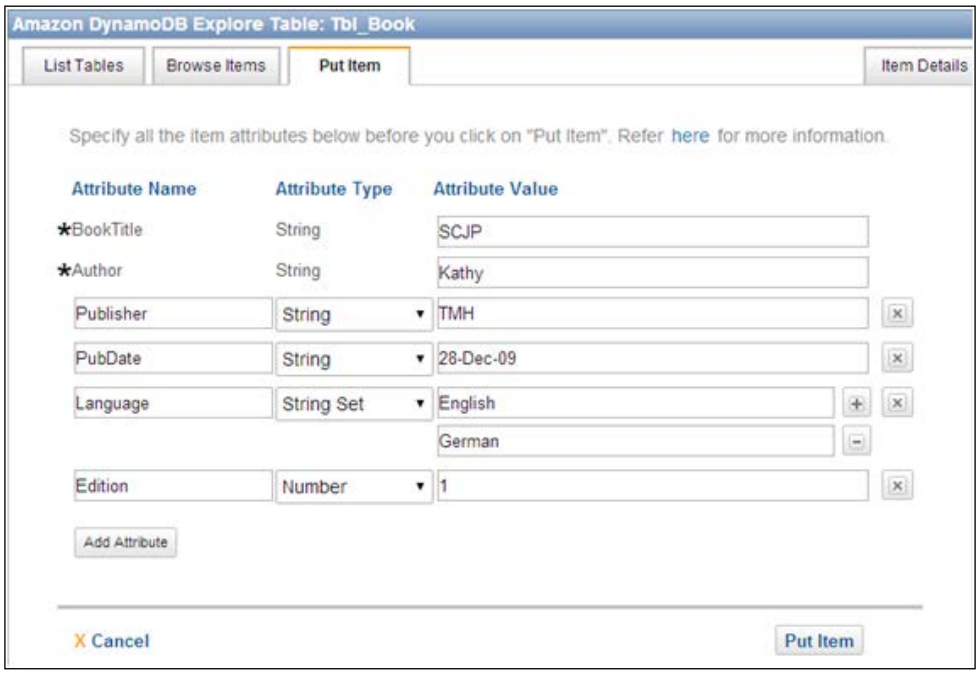
The next page will deliver a selection to produce secondary indexes, which we need not worry about much now. Once we advance with all the command buttons in the browser, we will see the following page:



Initially, the status will be **CREATING**. Once it becomes **ACTIVE**, then we can click on **Explore Table** (as shown to insert (or scan) items into the table. Clicking on the **Explore Table** will open the following screen:

Once we have clicked on **Explore Table**, we should click on the

button to supplement an item. By clicking on this button, the subsequent window will open (we have already populated it for saving paper).



The mandatory attributes, which are name and type, will already be set and we cannot alter it. However, we can enhance the attribute values (which must be distinctive).

In addition to this, we can click on unfilled textboxes (below the hash and range key attribute name) to improve an item's precise attribute name, type, and value.

Here, the first four attributes are of the type String, so we can insert the corresponding values into the attribute value.

In the case of inserting a set (StringSet for the `Language` field), stipulate numerous strings or numbers by clicking on the **+** symbol to the right of the value text box. Once all the attributes are inserted, click on the Put Item button, which will place this item in the `Tbl_Book` table.

To view the inserted item, click on the **Browse Items** tab, choose the **Scan** radio button and, click on the **Go** button. Now, we will be able to check the table content as shown in the following screenshot:

The string attribute values are bound in double quotes and set attribute values are bound by curly braces. Number attribute values won't be surrounded by a character.

There are some rules while using set data types. These are listed as follows:

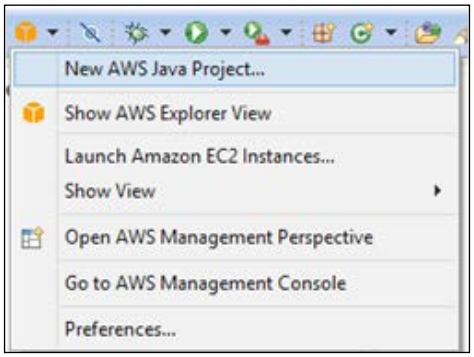
- A set must have a non-zero number of elements (blank sets are not allowed)
- A set must not have supplementary values (the **Language** set will not take "English","English")

There is a distinct data type called Binary, which is responsible for storing base64 encoded values. It is also used to store images or pictures in a base64 encoded structure.

Creating the first SDK project

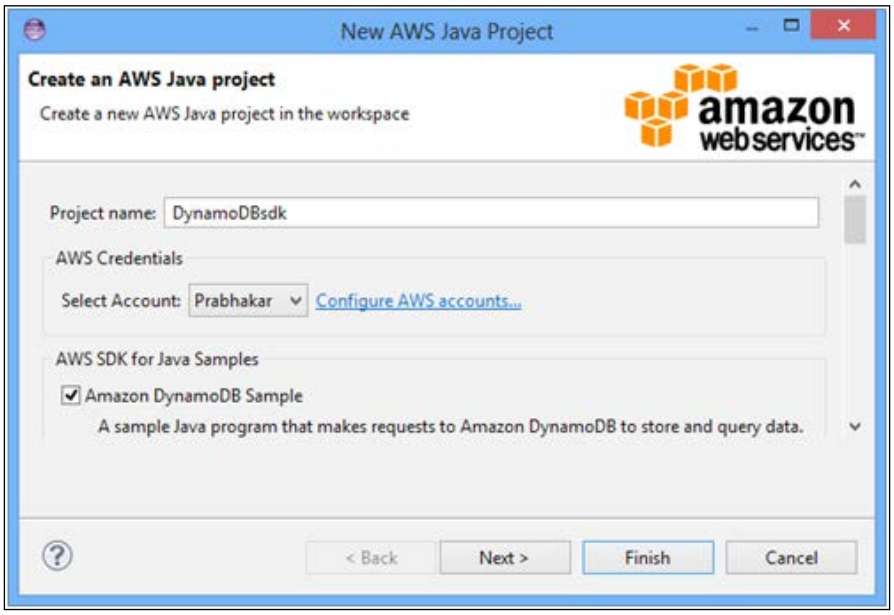
If we have already installed the Eclipse plugin and can see the credentials file created correctly, then we are ready to fly on the SDK plane.

1. By clicking on the AWS toolkit for Eclipse icon , we will see the option to create a new AWS project, as shown in the following screenshot:

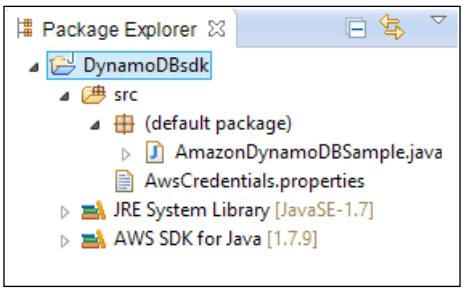


2. Here, we need to select **New AWS Java Project...**. Clicking on this option will open the following window. Clicking on this option for the first time will get a few sample codes from AWS and will ask whether we want these sample codes to be part of the project.
3. It is recommended that you check the **Amazon DynamoDB Sample** checkbox for the first time, to familiarize yourself with the syntax of the DynamoDB table operations.
4. Once done, select the AWS account that is already configured or configure a new AWS account.

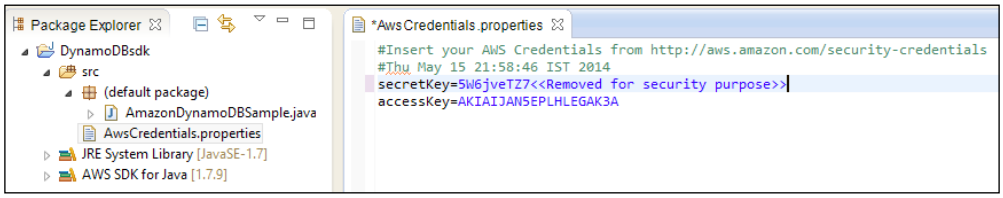
5. Click on the **Next** button to proceed:



6. Clicking on the **Next** button will create a new project with the name specified in the previous window. The project structure is as shown in the following diagram:
7. In the `src` folder of the project, the credentials file will be made available by default. The sample DynamoDB code will also be available in the default package of this `src` folder in a file named `AmazonDynamoDBSample.java`.

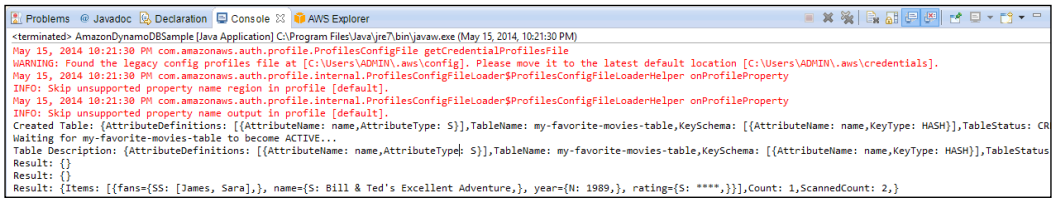


8. We can take a look at our credentials file by double-clicking on the `AwsCredentials.properties` file. The file content is as shown here (I removed my complete secret key for security purposes. You are not supposed to share your keys with anyone. It is like opening the gate of your safe; anyone with your key can pretend to be you):



Even though the properties file is located in the project, while running any program through Eclipse, it will always fetch the configuration information located at `$USER_HOME/.aws/config`. However, it is not safe to place the credentials file at this location. AWS always wants us to keep this information at `$USER_HOME/.aws/credentials` (take a look at line two of the following screenshot.)

Even if we provide invalid credential information in the project's properties file, it will fetch the correct ones from the default location while running the project. Running the sample code will yield us the following output:

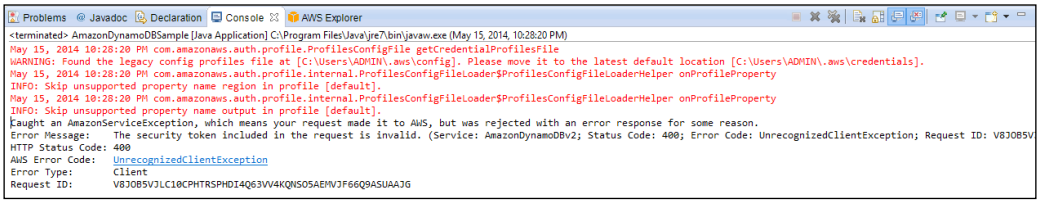


Output of the AwsCredentials.properties fie

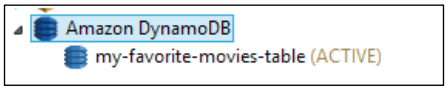
All the red-colored lines are logger messages (warning, information, or error) and the black-colored lines are system output that are the print statements mentioned in the sample code.

I have deliberately modified the access key of the credentials file located at `$USER_HOME/.aws/config`. After this, if we try running the sample code it will give `UnrecognizedClientException`.

This exception will be thrown only if the project is not able to instantiate the DynamoDB client, which is a clear indication that there is something wrong with the credentials:

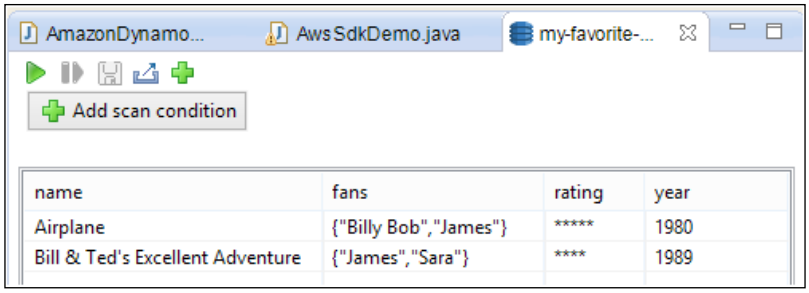


Since the sample code has been provided by AWS themselves, I don't want to get into trouble by providing the code here. So we will see what this sample code does. First and foremost, it creates a table named `my-favorite-movies-table` in the `US_WEST_2` region. Once we have run this code, we need to open AWS Explorer and refresh Amazon DynamoDB shown as follows:



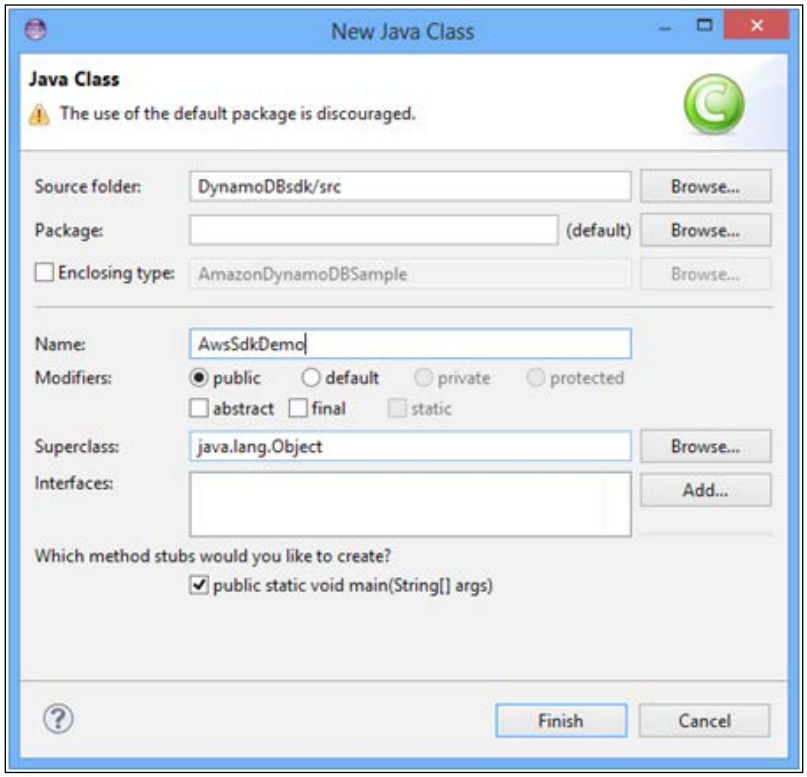
Make sure that you're selecting the correct region (`US_WEST_2`) in the AWS Explorer, otherwise we cannot see the table that is created.

Double-clicking on the table name will open the following window showing the content of the table:



I hope that there is nothing I need to explain about the table's attribute names and types. In this table, **name** is the only key attribute.

The sample code provided will not have code to create any indexes. In the following topics of this chapter, we will discuss everything in detail. First, we will create a new class named `AwsSdkDemo` in the same project:



In this DynamoDB class (named `AwsSdkDemo`), we can perform the following DynamoDB operations:

- Initializing our AWS credentials
- Defining table attributes
- Defining key schema (of table and indexes)
- Creating local and secondary indexes
- Defining the provisioned throughput
- Creating a table with the preceding parameters
- Describing a table
- Updating the DynamoDB table
- Table status check
- Adding (placing) items to the table

Java SDK operations

There are eight user-defined private local functions that are being invoked in the following code. We will see each and every function in detail:

```
public class AwsSdkDemo {

static AmazonDynamoDBClient client;
initializeCredentials();
String tableName = "Tbl_Book";

if (Tables.doesTableExist(client, tableName)) {
  System.out.println("Table " + tableName + " already EXISTS");
}
else {
  ArrayList<AttributeDefinition> attributeDefinitions =
getTableAttributes();
  ArrayList<KeySchemaElement> keySchemaElements =
getTableKeySchema();
  LocalSecondaryIndex localSecondaryIndex =
getLocalSecondaryIndex();
  ArrayList<LocalSecondaryIndex> localSecondaryIndexes = new
ArrayList<LocalSecondaryIndex>();
  localSecondaryIndexes.add(localSecondaryIndex);
  GlobalSecondaryIndex globalSecondaryIndex =
getGlobalSecondaryIndex();
  ProvisionedThroughput provisionedThroughput =
getProvisionedThroughput();

  CreateTableRequest request = new CreateTableRequest()
    .withTableName(tableName)
    .withAttributeDefinitions(attributeDefinitions)
    .withKeySchema(keySchemaElements)
    .withProvisionedThroughput(provisionedThroughput)
    .withGlobalSecondaryIndexes(globalSecondaryIndex);
      request.setLocalSecondaryIndexes(localSecondaryIndexes);

  CreateTableResult result = client.createTable(request);

  System.out.println("Waiting for " + tableName + " to become
ACTIVE...");

  Tables.waitForTableToBecomeActive(client, tableName);
```

```
    TableDescription tableDescription = client.describeTable(
      new DescribeTableRequest()
      .withTableName(tableName))
      .getTable();

    System.out.println("Created Table: " + tableDescription);

    UpdateTableRequest updateTableRequest =
  getUpdateTableRequest(tableName);
    UpdateTableResult updateTableResult =
  client.updateTable(updateTableRequest);
    putItems(tableName);
  }}
```

For the complete code, please refer to `3632EN_03_01.txt` in the code bundle.

The first chunk of code is to load AWS credentials and authenticate ourselves to AWS to run the program and perform the DynamoDB operation.

For the kind of DynamoDB operations we wish to perform, they must be done through this client:

```
static AmazonDynamoDBClient client;
```

The following block will initialize the table name to the local variable. Then, the `if` condition will check whether the table already exists with this name (on the client configured region) and returns the Boolean value. If the table already exists, then the system output message will be printed as follows:

```
String tableName = "Tbl_Book";
if (Tables.doesTableExist(client, tableName)) {
  System.out.println("Table " + tableName + " already EXISTS");
}
```

Refer to `3632EN _03_02.txt` in the code bundle for the complete code.

The following block will create `CreateTableRequest` with attributes such as `tablename`, attribute definitions, key schema, provisioned throughput, and indexes:

```
CreateTableRequest request = new CreateTableRequest()
  .withTableName(tableName)
  .withAttributeDefinitions(attributeDefinitions)
  .withKeySchema(keySchemaElements)
  .withProvisionedThroughput(provisionedThroughput)
  .withGlobalSecondaryIndexes(globalSecondaryIndex);
```

Refer to `3632EN_03_03.txt` in the code bundle for the complete code.

The following line will submit the table creation request through the DynamoDB client:

```
client.createTable(request);
```

The following line of code will pause the further execution of the code, until the table becomes active (most probably used before adding items to the table).

```
Tables.waitForTableToBecomeActive(client, tableName);
```

The following code will request the user to describe the table name passed as a parameter to the client:

```
client.describeTable(new DescribeTableRequest()
  .withTableName(tableName))
  .getTable();
```

Refer to `3632EN_03_04.txt` in the code bundle for the code snippet.

The following code will update a table with the `UpdateTableRequest` instance passed:

```
client.updateTable(updateTableRequest);
```

As we are already aware, we have kept our credentials file at `$USER_HOME/.aws/config`, which the SDK will easily identify (even though the default location is `$USER_HOME/.aws/credentials`). In that code chunk:

- The first line of try block will load the default AWS credentials
- The next line will configure the DynamoDB client with the loaded credentials
- The next line will initialize the region to `US-WEST-2`, which is `Oregon`
- The last line of the `try` block will set the region for the DynamoDB client to `US-WEST-2`

In case of an improper location of the credentials file, this exception will be thrown:

```
private static void initializeCredentials() throws Exception {
AWSCredentials credentials = null;
try {
  credentials = new
ProfileCredentialsProvider().getCredentials();
  client = new AmazonDynamoDBClient(credentials);
```

```
    Region usWest2 = Region.getRegion(Regions.US_WEST_2);
    client.setRegion(usWest2);
  } catch (Exception e) {
    throw new AmazonClientException(
      "Invalid location or format of credentials file.",e);
  }
  }
```

Refer to `3632EN_03_05.txt` in the code bundle.

The following function will prepare `ArrayList`, which adds all `AttributeDefinition` to it. Each `AttributeDefinition` will take two parameters: the first is the attribute name and the other is the attribute type. In the following code, we define five attributes:

```
private static ArrayList<AttributeDefinition> getTableAttributes()
{
  ArrayList<AttributeDefinition> attributeDefinitions = new
ArrayList<AttributeDefinition>();
  attributeDefinitions.add(new AttributeDefinition()
    .withAttributeName("BookTitle")
    .withAttributeType("S"));
  attributeDefinitions.add(new AttributeDefinition()
    .withAttributeName("Author")
    .withAttributeType("S"));
  attributeDefinitions.add(new AttributeDefinition()
    .withAttributeName("PubDate")
    .withAttributeType("S"));
  attributeDefinitions.add(new AttributeDefinition()
    .withAttributeName("Publisher")
    .withAttributeType("S"));
  attributeDefinitions.add(new AttributeDefinition()
    .withAttributeName("Edition")
    .withAttributeType("N"));
  return attributeDefinitions;
}
```

Refer to `3632EN_03_06.txt` in the code bundle.

The following method will return `ArrayList` of the `KeySchemaElement` type. Inside this function, we instantiate `ArrayList` of the `KeySchemaElement` type. To this `ArrayList`, we add two `KeySchemaElement`. The first element is for setting the attribute `BookTitle` as the `HASH` key type and the second element is to set the attribute `Author` as the `RANGE` key type. Finally, we return this `ArrayList`:

```
private static ArrayList<KeySchemaElement> getTableKeySchema() {
  ArrayList<KeySchemaElement> ks = new
           ArrayList<KeySchemaElement>();
```

```
    ks.add(new KeySchemaElement()
      .withAttributeName("BookTitle")
      .withKeyType(KeyType.HASH));
    ks.add(new KeySchemaElement()
      .withAttributeName("Author")
      .withKeyType(KeyType.RANGE));
    return ks;
  }
```

Refer to`3632EN_03_07.txt` in the code bundle.

The following method will return the `ProvisionedThroughput` instance with the populated write and read throughput capacities for our table. The long number (`2L`) here means the maximum read or write data size per second. This is usually measured in KBps. So here, we are restricting the read-write speed to 2 KBps:

```
private static ProvisionedThroughput getProvisionedThroughput() {
  ProvisionedThroughput provisionedThroughput = new
ProvisionedThroughput()
      .withReadCapacityUnits(2L)
      .withWriteCapacityUnits(2L);
  return provisionedThroughput;
}
```

Refer to `3632EN_03_08.txt` in the code bundle.

The following code tries to update the table that has already been created. Here, we modify only the `ProvisionedThroughput` capacity units. During the table creation, we have set this value to `2L`; now we update it to `4L`. The `UpdateTableRequest` request will allow us to change only a few parameters, such as changing the provision throughput capacity of the table and (if needed) the secondary indexes throughput capacity. If you have a question like can we update (add or remove) secondary indexes of the table? The answer is no, we cannot add or remove secondary indexes using `UpdateTableRequest`. The reason lies in the *DynamoDB data types* section in this chapter.

```
private static UpdateTableRequest getUpdateTableRequest(String
tableName) {
  ProvisionedThroughput upt = newProvisionedThroughput()
    .withReadCapacityUnits(4L)
    .withWriteCapacityUnits(4L);

  UpdateTableRequest updateTableRequest = new
                    UpdateTableRequest()
```

```
        .withTableName(tableName)
        .withProvisionedThroughput(upt);
    return updateTableRequest;
}
```

Go through `3632EN_03_09.txt` in the code bundle for the complete code.

In the following function, we will try to put two items (`item1` and `item2`, each of the type `Map<String, AttributeValue>`) into the table (whose name is taken as an input parameter). As we discussed earlier (the `getTableKeySchema` method), every item must have primary key attributes (the `BookTitle` and `Author` attributes). So both the items have these two attributes.

In the first item (`item1`) we are totally adding four attributes, namely `Publisher` (String type), `PubDate` (String type), `Language` (StringSet type) and `Edition` (`Number` type). In order to add the attributes, we must call the correct method of the `AttributeValue` class depending on the type of attributes we need to pass as an argument.

In the second item (`item2`) too, we add the same attributes for another book, with an additional attribute named `Pages` (Number type):

```
private static void putItems(String tableName) {
  Map<String, AttributeValue> item1 = new HashMap<String,
              AttributeValue>();
  Item1.put("BookTitle", new AttributeValue().withS("SCJP"));
  Item1.put("Author", new AttributeValue().withS("Kathy"));
  Item1.put("Publisher", new AttributeValue().withS("TMH"));
  Item1.put("PubDate", new AttributeValue().withS("28-Dec-09"));
  Item1.put("Language", new AttributeValue()
    .withSS(Arrays.asList("English", "German")));
  Item1.put("Edition", new AttributeValue().withN("1"));
  PutItemRequest putItemRequest = new PutItemRequest()
    .withTableName(tableName)
    .withItem(item1);
  client.putItem(putItemRequest);

  Map<String, AttributeValue> item2 = new HashMap<String,
              AttributeValue>();
  item2.put("BookTitle", new AttributeValue().withS("Inferno"));
  item2.put("Author", new AttributeValue().withS("DanBrown"));
  item2.put("Publisher", new AttributeValue().withS("TMH"));
  item2.put("PubDate", new AttributeValue().withS("28-Jul-12"));
  item2.put("Language", new AttributeValue()
```

```
        .withSS(Arrays.asList("English")));
    item2.put("Edition", new AttributeValue().withN("1"));
    item2.put("Pages", new AttributeValue().withN("623"));
    PutItemRequest putItemRequest1 = new PutItemRequest()
      .withTableName(tableName)
      .withItem(item2);
    client.putItem(putItemRequest1);
  }
```

Refer to `3632EN_03_10.txt` in the code bundle.

In the `getTableAttributes` method defined earlier in this section, have we specified or defined the attributes `Pages` and `Language`? The answer is 'No'.

While creating the table, we must specify the primary key and index attributes; all other optional attributes can be specified during item insertion. `Pages` and `Language` being non-key and non-index attributes, they are not part of the attribute definition in the `getTableAttributes` method.

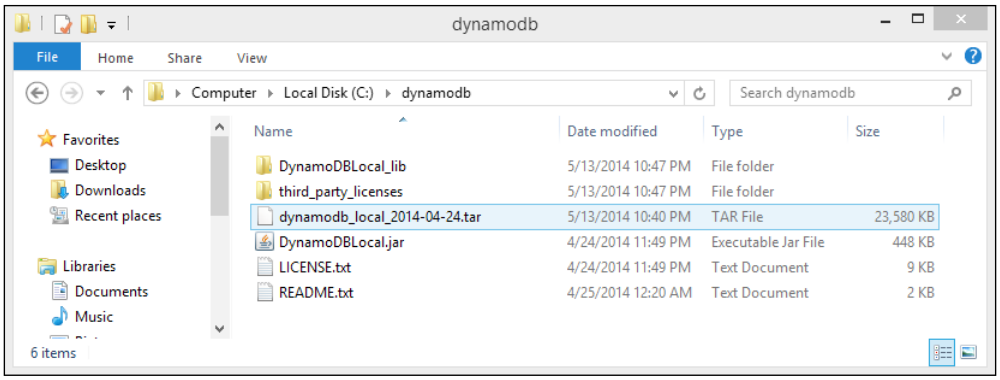
There are two more local functions invoked to create indexes, which we will see in the next chapter.

The DynamoDB local

The DynamoDB local is a local client-side database, which emulates the DynamoDB database in our local system. This is pretty helpful when developing an application that uses DynamoDB as the backend. After writing a module, we need to connect to Amazon and run it to check whether the code works fine. This will consume a lot of bandwidth along with a few dollars. Moreover, it will increase the speed of development as well as it's easy for unit testing. To avoid this, we can test the code locally with the help of the DynamoDB local. Once the testing is done, we can make our application use the AWS DynamoDB service. This requires only three things, which are as follows:

- Downloading DynamoDB local from `http://dynamodb-local.s3-website-us-west-2.amazonaws.com/dynamodb_local_latest`
- Starting the DynamoDB local service (JRE6 or later)
- Pointing the code to use the DynamoDB local port

We don't need to discuss more about how to download a file from the Internet. So let's go directly to the second point. The downloaded file might be a zipped one (`tar.gz` or `.zip` or `.rar`). We need to extract it to a location. You can extract it to `C:\dynamodb`, as shown in the following screenshot:



Starting the DynamoDB local is very easy. First, we need to change the working directory using the `cd` command, and then we can start the DynamoDB local on port 8888 by using the following command:

```
java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar -port
8888
```

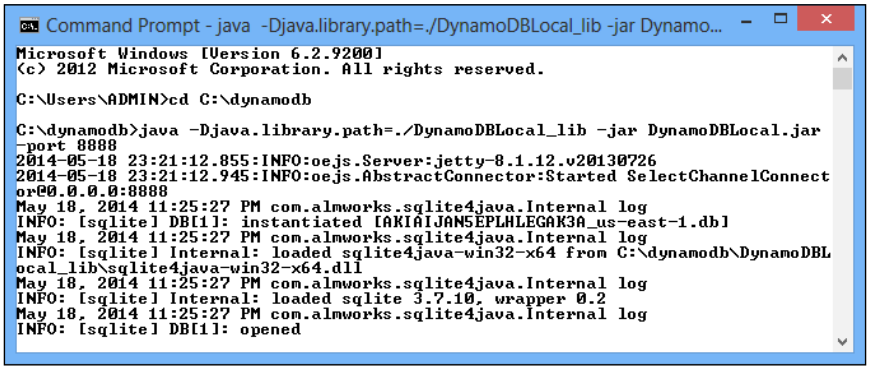
Even the command `java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar` is enough to start the DynamoDB local, but it starts on port 8000, which is occupied by my PC. That's why I used port 8888.

Once the DynamoDB local has started, it's easier to configure the client. We need to make changes to three lines of our initializeCredentials method (discussed earlier in the chapter). We need to insert a new line pointing to the DynamoDB local host and port using the `client.setEndpoint()` method (shown as follows). Then, we need to remove other client-related setters such as `setRegion` as shown in the following code:

```
private static void initializeCredentials() throws Exception {
  AWSCredentials credentials = null;
  try {
    credentials = new  ProfileCredentialsProvider()
      .getCredentials();
  } catch (Exception e) {
    throw new AmazonClientException(
    "Invalid location or format of credentials file.",e);
  }
  client = new AmazonDynamoDBClient (credentials);
  client.setEndpoint("http://localhost:8888");
  //Region usWest2 = Region.getRegion(Regions.US_WEST_2);
  //client.setRegion(usWest2);
}
```

You'll find this code snippet in `3632EN_03_11.txt` in the code bundle.

After this, if we run the `AwsSdkDemo` class, it would give the following output in the console (where DynamoDB local has started):

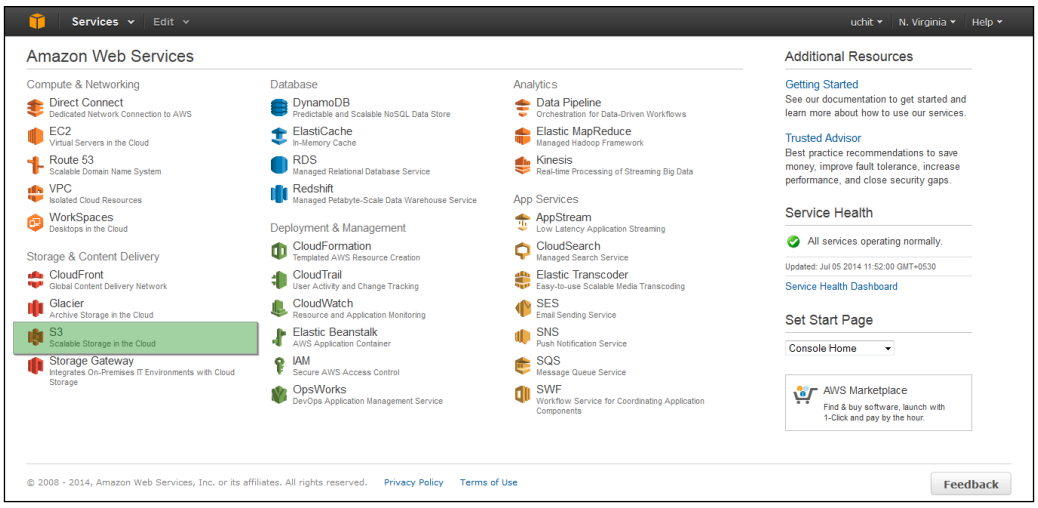


AWS Simple Storage Service (S3)

To store an object in Amazon S3, you can upload the file you need to store to a bucket. When you upload a file, you can set permissions on the object as well as some metadata.

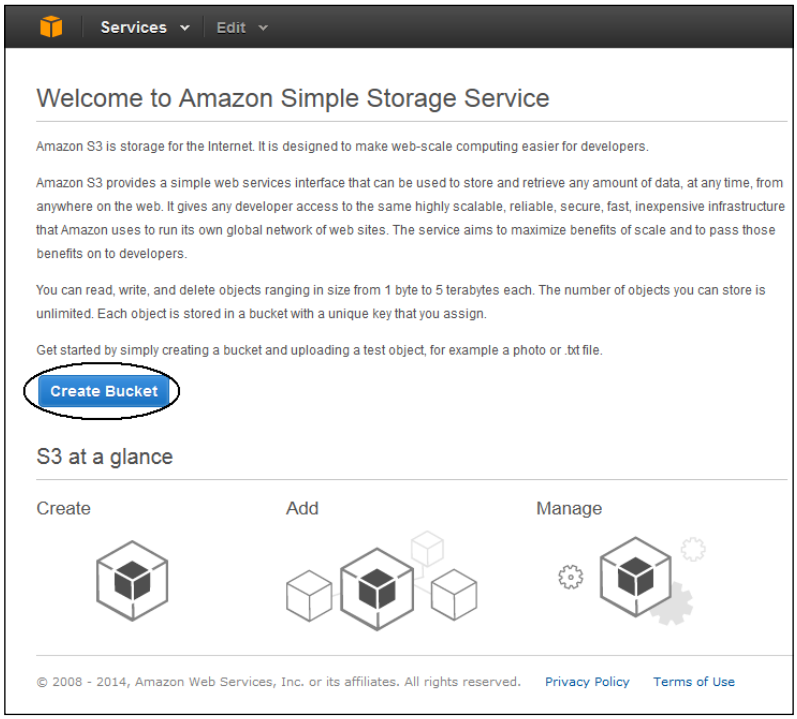
Buckets will work as the containers for your contents (objects). You can have one or more buckets as per your requirement. For each bucket, you can control access to it (who can generate, delete, and list objects in the bucket), view access logs for it and its objects, and select the geographical region where Amazon S3 will store the bucket and related contents.

Amazon S3 can store any data as objects within storage containers called buckets. An object can be a file and may be any metadata that describes that file. You can find the following window for the AWS S3 dashboard to understand more specifically:



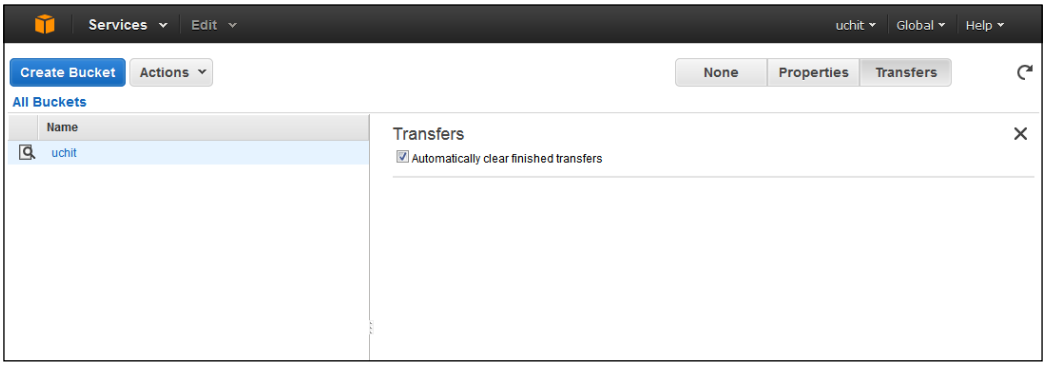
AWS S3 dashboard

By selecting S3, you will be on the S3 dashboard, which will look something like the following:



Just click on the **Create Bucket** button and you will be prompted by a pop-up box to create your first bucket with S3, its region name to get the bucket creation dashboard to see the power of AWS S3 and its configurations (available options).

The bucket name should be unique all over the globe, otherwise, it won't allow you to create a bucket with that name. After the creation of the bucket, you will see the following window:



For bucket-related operations, you can perform them via CLI, API, or from the dashboard directly. In this chapter, you will learn about bucket operations and configurations from SDK. To start with SDK, you will require the AWS SDK and AWS account with an access key and private key to connect to Amazon S3. The object controls all the engagements through which you can interact with the AWS S3 instance:

```
const string AWS_ACCESS_KEY = "your_AWS_access_key";
const string AWS_SECRET_KEY = "your_AWS_secret_key";
AwsS3 client = new AwsS3(AWS_ACCESS_KEY, AWS_SECRET_KEY)
```

Normally, these keys are stored in `web.config` and we access them by code in web-based applications:

```
// In your application config file, set this
<appSettings>
    <add key="AWSAccessKey" value="AWS_access_key"/>
    <add key="AWSSecretKey" value="AWS_secret_key"/>
<appSettings>
```

You need to create a function to access your S3 credentials:

```
// Function to get credentials
public static AwsS3 GetS3Client()
{
    NameValueCollection appConfig =
ConfigurationManager.AppSettings;
```

```
        AwsS3 s3Client = AWSClientFactory.CreateAmazonS3Client(
                appConfig["AWSAccessKey"],
                appConfig["AWSSecretKey"]
                );
        return s3Client;
    }
```

Refer to `3632EN_03_12.txt` for this code snippet in the code bundle.

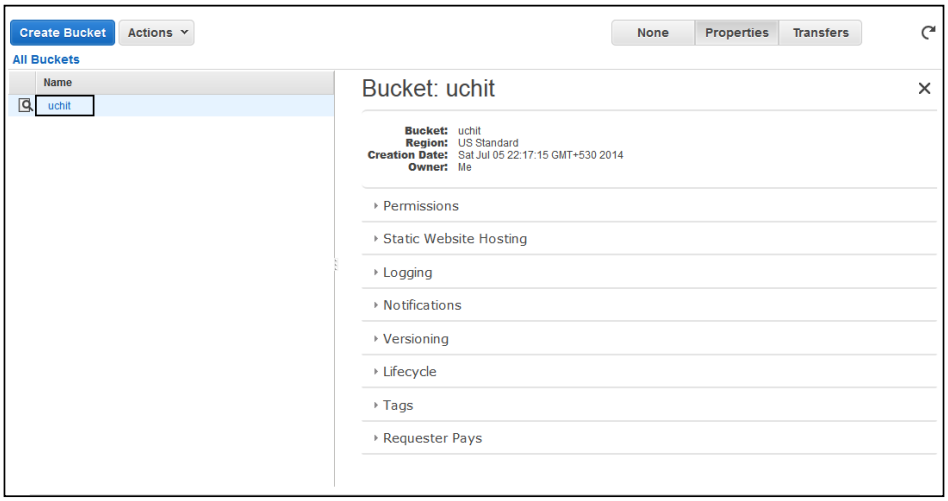
This way, you can start work with AWS S3. Next, you will see how to upload and retrieve content from the S3 bucket in the following example. If you want to store data in S3, you need to create a bucket. It is similar to a root folder in Windows. In Amazon S3, the maximum number of buckets is 100 and the names of buckets are unique globally.

You are allowed to create a maximum of 100 buckets per account and the bucket names should be unique. Using the following code, you can create a sample bucket on AWS S3:

```
string BUCKET_NAME = "uchit";
ListBucketsResponse response = client.ListBuckets();
boolean found = false;
foreach (S3Bucket bucket in response.Buckets)
{
    if (bucket.BucketName == BUCKET_NAME)
    {
        found = true;
        break;
    }
}
if (found == false)
{
    client.PutBucket(new
PutBucketRequest().WithBucketName(BUCKET_NAME));
}
```

You'll find this code snippet in `3632EN_03_13.txt`.in the code bundle.

After successful execution of the preceding code, you will be able to see a new bucket named **uchit** on the AWS S3 dashboard:



The S3 bucket: uchit

To create a new file in the S3 bucket there are plenty of ways, but here we will introduce some generic simple ways. In the first way, you need `FileKey`, which will be unique to the full path and content body that contains information. To create a directory, use `FileKey` with the special character `/` at the end to show that you want to create a specific directory:

```
String S3_KEY = " Demo Create folder/";
PutObjectRequest request = new PutObjectRequest();
request.WithBucketName(BUCKET_NAME);
request.WithKey(uchit_KEY);
request.WithContentBody("");
client.PutObject(request);
```

Refer to `3632EN_03_14.txt`, which contains this code snippet.

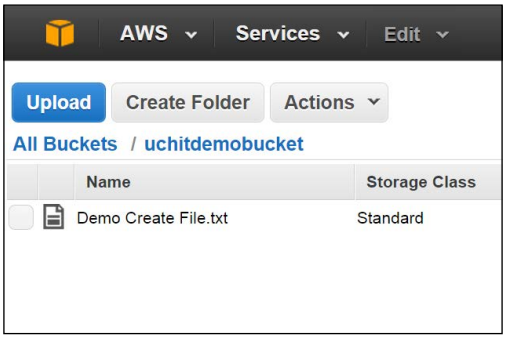
The new directory will be created, which will look like the following screenshot:

To create a new file, there is a minor change in the code from the directory creation code that you can see with following code:

```
String S3_KEY = "Demo Create File.txt";
PutObjectRequest request = new PutObjectRequest();
request.WithBucketName(BUCKET_NAME);
request.WithKey(uchit_KEY);
request.WithContentBody("This is content of S3 object in Demo
file.");
client.PutObject(request);
```

For this code snippet, see `3632EN_03_15.txt` in the code bundle.

After successful execution of the preceding code, you will be able to see the file created in the bucket:



If you want to create a file within the directory, you have to change the `FileKey` to include the directory name:

```
String S3_KEY = "Demo Create folder/" + "Demo Create File.txt";
PutObjectRequest request = new PutObjectRequest();
request.WithBucketName(BUCKET_NAME);
request.WithKey(uchit_KEY);
request.WithContentBody("This is content of S3 object in Demo
file.");
client.PutObject(request);
```

Check `3632EN_03_16.txt` in the code bundle for the complete code.

After successful execution of the preceding code, you will be able to see the file created in the defined directory within the bucket. Next, you will see how to upload files from the local machine to the S3 bucket with the absolute path using the following code:

```
//uchit_KEY is name of file we want to upload
PutObjectRequest request = new PutObjectRequest();
request.WithBucketName(BUCKET_NAME);
request.WithKey(uchit_KEY);
request.WithFilePath(pathToFile)
client.PutObject(request);
```

Take a look at `3632EN_03_17.txt` for the code snippet.

Amazon S3 has so many useful features and is very useful for every developer who wants to develop a web app that needs to store data online with some knowledge of the AWS SDK.

Amazon CloudFront

CloudFront is a **Content Delivery Network** (**CDN**) service from AWS to deliver the entire web application, including streaming or interactive content, and dynamic or static content with the AWS global delivery network of edge locations. Content request will automatically be routed to the nearest edge location, so content will be delivered with the best performance. It is optimized to work seamlessly with other AWS services such as AWS S3, Amazon EC2, Amazon ELB, Route 53, and also with a non-AWS origin server, which will store the definitive versions of your files. We will see how to leverage Amazon CloudFront using examples as it will be more beneficial for developers and you can read theoretical information on the AWS website as well. Mainly, there are various options and areas in which AWS CloudFront can work effectively, such as the following:

1. Web distribution and RMTP distribution
2. Private content distribution
3. Dynamic and static content
4. HTTP and HTTPS protocols

In this chapter, you will look at the example of creating a CDN for the WordPress blog with CloudFront and S3. A CDN is a farm of servers that are distributed around the world, which will have copies of your content. In this way, users visiting your site get a response from the server nearest to their location. Hence, decreasing data travel distance, time, and intermediaries, results in faster access to the content. To enable your WordPress blog to take advantage of Amazon CloudFront, you have to make the following changes:

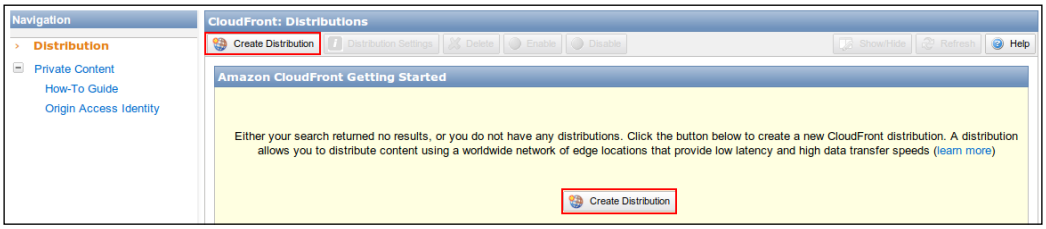
1. Store your site images outside the web server, or in other words, offload any image-related work from your application server, let's say from EC2 instances.
2. Allow your browsers to download your images at the same time they download the textual content. For this, you have to create an Amazon S3 bucket.
3. You have to create a CloudFront Distribution by clicking on **Create Distribution** button, which will cache your images from S3 to the configured edge locations over the globe.
4. You can install the WordPress plugin, which will serve the purpose of uploading your images to S3 and will give you a CloudFront link to the images.

You already learned how to create a bucket so you can start directly with the second stage mentioned earlier, if you have the S3 bucket available to you. If not, please create your bucket in the same region in which you are going to create CloudFront Distribution.

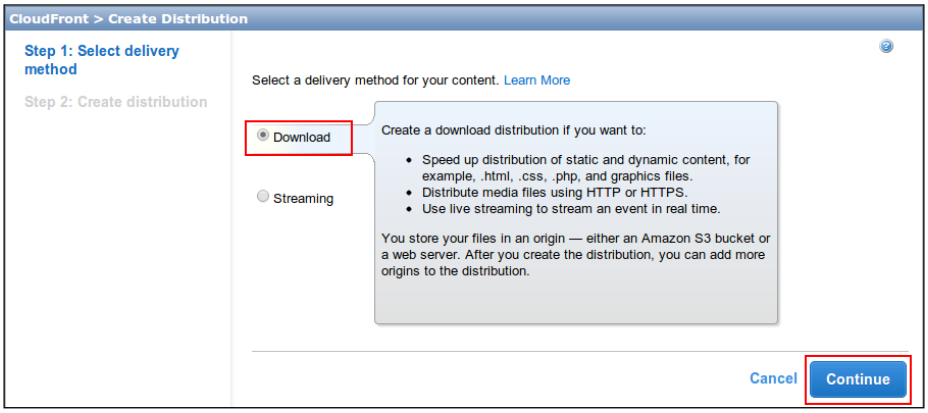
Creating Amazon CloudFront Distribution

You can create your CloudFront Distribution from CLI, the AWS Console, and using APIs. However, for convenience at the first stage, here you will be creating using the console.

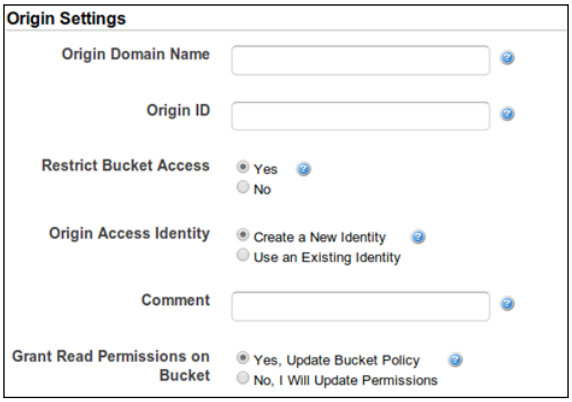
1. Go to the AWS services dashboard and select **CloudFront** from the listed services and click on the **Create Distribution** button:



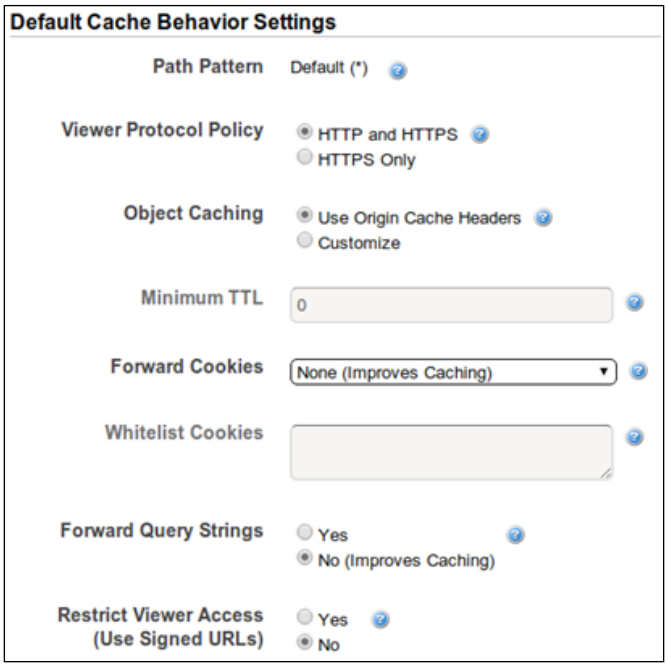
2. CloudFront supports you to create static content as well as streaming content distributions. You are going to distribute blog-related content, so select the first option ("**Download**") and click on the **Continue** button:



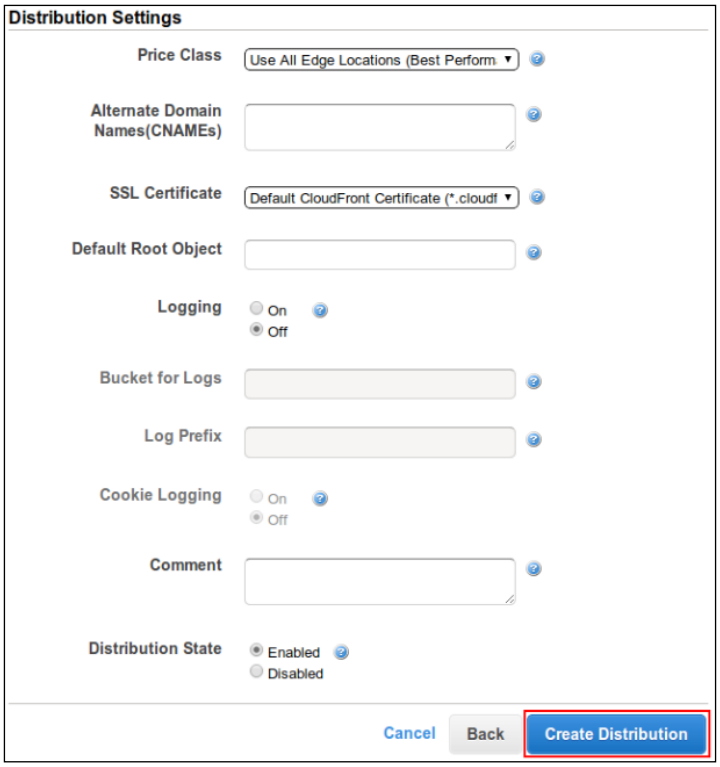
3. Select your distribution's content origin. You need to specify the source of the content that you want in the **Origin Domain Name** field. As seen in the previous section, you have already created the S3 bucket, so your console will help you in that case to identify the bucket and its related **Origin ID**, which will just be the nickname assigned to the origin.
4. I recommend enabling the **Restrict Bucket Access** option to ensure your content is not accessible directly from S3 but via CloudFront only. This implies that you need to create an IAM identity and edit the bucket policies but Amazon will do it for you here, you only have to choose **Create new identity** and enable **Yes, Update Bucket Policy**:



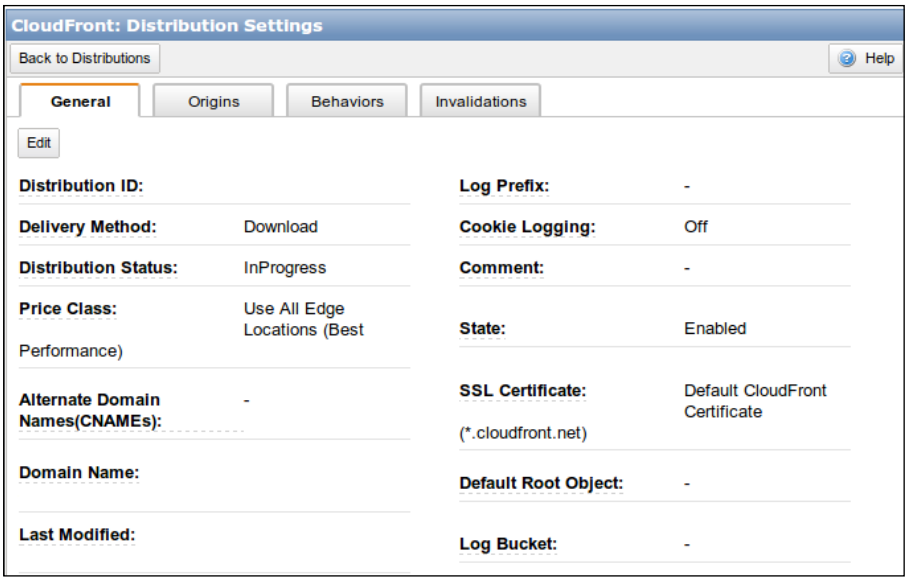
5. Choose **Default Cache Behavior Settings** option. In this section, you can enforce HTTPS usage. You can also select the cache behavior by policy or in the content header. You can also forward cookies and URL query strings to your origin, or restrict access for your content using signed URLs:



6. Choose **Distribution Settings**. In this section, you have to select **Price Class** to select regions and edge locations where your content will be cached. The pricing is dependent on the regions and edge locations, so pick up the correct and appropriate locations based on your traffic analysis and requirement. You can even set an alias on your DNS to your CDN. For this, you have to declare name(s) as **Alternate Domain Names (CNAMEs)**. Once done, click on **Create Distribution** button:



7. Once the distribution is in the **Enabled** state on the AWS console, you are almost done. You can check the settings in which you will be able to find the domain name for your content. Just note that you will have to tell the WordPress blog where your CDN will be located:



8. Now you have to create an IAM user who can manage your S3 bucket and assign a custom policy to it. Optionally, you can integrate Amazon S3 with Amazon CloudFront. For this, you will need the WordPress plugin, which is available on `http://wordpress.org/plugins/amazon-s3-and-cloudfront/`. So after the configuration and installation of this plugin, you will see a new segment with the CloudFront URL where the content will be served from.

To remove the content from the edge locations, you can use the invalidating functionality of CloudFront. For this, you have to do the following:

1. Traverse to the CloudFront section on the AWS console.
2. Choose your distribution.
3. Click on the **options** icon.
4. From here, move to the **Invalidations** tab and then **Create Invalidation**.
5. Place the names of the files to be invalidated.
6. Click on the **Invalidate** button.

Once this procedure is complete, the object cached on the edge locations will be removed and the latest version will be cached at the subsequent time it is requested.

Amazon RDS management with CLI

Amazon **Relational Database Service** (**RDS**) is a web service that makes it easy to set up, operate, and scale a relational database on the AWS Cloud platform. It has access to the full capabilities similar to MySQL, Oracle, MSSQL, PostgreSQL, and Aurora databases. So this means the code, applications, and tools you already use today with your existing databases work seamlessly with Amazon RDS as well. There are a number of advantages to Amazon RDS, which includes the following:

1. Accelerated deployment
2. Managed and scalable
3. Reliable and compatible
4. Automated backup and multi AZ deployment
5. Secure

To manage the Amazon RDS, you have to first set up tools for it. In our exercise, we will install tools and the setup environment, database instance creation and listing, and connect database instances. So let's gear up with the environment setup and configuration. To install tools, you need Java 1.6 or higher and the Amazon RDS command-line toolkit. To start with Java:

1. Set the Java variable in your system:
    1. The command-line tool reads an environment variable (`JAVA_HOME`) on the machine to locate the Java runtime. Either JRE or JDK should be fine with the Version 6 or higher. To download Java, go to `http://java.oracle.com/`.
    2. Extract your downloaded Java and set the path using variable `JAVA_HOME` to the full path of the directory, which will contain the `bin` subdirectory. For example, if you have Java in the `/opt` directory, the path should be `JAVA_HOME` to `/opt/jdk` for Linux and `C:\jdk` for Windows.
    3. To set `JAVA_HOME`, use the following commands:

       For Linux:

       ```
       $export JAVA_HOME=/opt/jre
       ```

For Windows:

```
C:\> set JAVA_HOME=C:\java\jdk1.6.0_6
```

4. Include the Java directory to your system path before other versions of Java:

   For Linux:

   ```
   $export PATH=$PATH;$JAVA_HOME/bin
   ```

   For Windows:

   ```
   C:\> set PATH=%PATH%;%JAVA_HOME%\bin
   ```

5. Verify your `JAVA_HOME` settings:

   For Linux:

   ```
   $ JAVA_HOME/bin/java –version
   ```

   For Windows:

   ```
   C:\> %JAVA_HOME%\bin\java –version
   ```

2. Setting up the RDS CLI tools. To access the RDS command-line toolkit, you need to download it from the AWS site `http://aws.amazon.com/developertools/2928/` and set it up with your AWS credentials within the instance. You need to just download and unzip it. No installation is required; it will come as a zip bundle.

   1. The CLI also depends on the environment variable, so we have to again set its path with the `AWS_RDS_HOME` variable:

      For Linux:

      ```
      $ export AWS_RDS_HOME=/usr/local/RDSCLI1.15.001
      $ export PATH=$PATH:$AWS_AUTO_SCALING_HOME/bin
      ```

      For Windows:

      ```
      C:\> set AWS_AUTO_SCALING_HOME=C:\CLIs\ RDSCLI1.15.001
      C:\> set PATH=%PATH%;%AWS_AUTO_SCALING_HOME%\bin
      ```

Your environment variables in the Windows machine may reset when you close the terminal window. You may want to set them permanently using the `setx` command same as `set`.

3. Authenticate your AWS account with RDS tools:

    1. After signing in, you need to create access keys and secret keys for your account. You have to provide these keys to your CLI tools. You can create your AWS secret keys and access ID from `https://console.aws.amazon.com/iam/home?#security_credential`.
    2. Create a new file called `CredentialFile` and save your access key ID and secret access key to the file.
    3. Give `600` permission to that `CredentialFile` file if you are a Linux user.

        ```
        $ chmod 600 CredentialFile
        ```

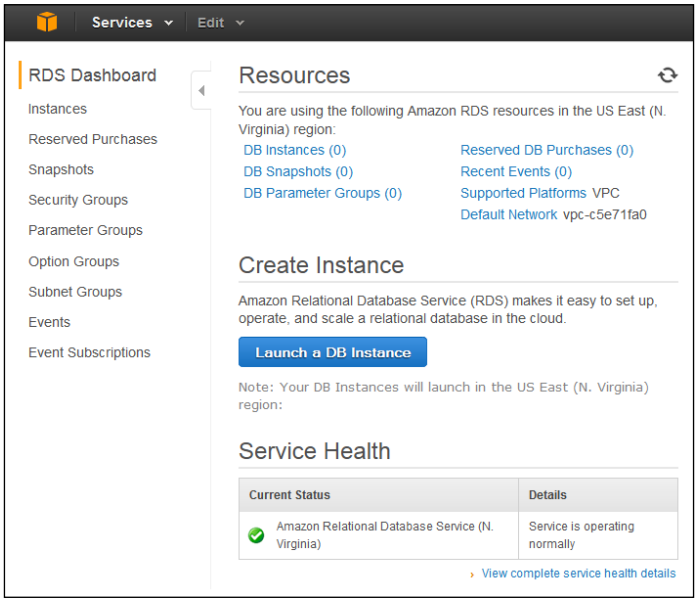
    4. Set the `AWS_CREDENTIAL_FILE` variable based on your file location.

Now, you are almost done with setting up the AWS RDS CLI tools. So you can test the tools on a Windows or Linux machine using the following command:

```
rds --help
```

You should see the usage page for all Amazon RDS commands. If it shows results, that means you have done a good job of setting up an RDS toolkit. Now it's time to create and launch an instance from CLI for RDS.

The following is the screenshot for the RDS dashboard from which you can perform the same operations and manage your database on the AWS infrastructure:



Now, to start an RDS using CLI, follow below steps:

1. To launch an RDS instance from CLI, use the following command. This will create a database instance called `uchitinstance` with 5 GB of storage and an initial database named `uchit`:

   ```
   rds-create-db-instance --engine MySQL5.3 --master-username root
   --master-user-password mypass --db-name uchit --db-instance-
   identifier uchitinstance --allocated-storage 5 --db-instance-class
   db.m1.xlarge –-header
   ```

2. You can see the below output if everything is correct:

   ```
   DBINSTANCE  DBInstanceId        Class          Engine    Storage
   Master Username

   Status    Backup Retention

   DBINSTANCE  uchitinstance  db.m1.xlarge  mysql5.3  5        root

   creating  1

   SECGROUP  Name     Status

   SECGROUP  default  active

   PARAMGRP  Group Name        Apply Status

   PARAMGRP  default.mysql5.3  in-sync
   ```

3. To list the available instances, use the following command:

   ```
   rds-describe-db-instances –-headers
   ```

Authorizing network access

You have to authorize your access to your database instance before you can connect to it. Access to database instances is controlled by database security groups. A database security group (called `default`) is created automatically the first time you create your database instance, and you can create new customized database security groups as required.

By default, a database security group has no access enabled; you must precisely authorize network ingress. There are two techniques for this:

- Authorizing a network IP range using the following command:

  ```
  rds-authorize-db-security-group-ingress default –cidr-ip
  192.0.4.0/30 –headers
  ```

You will get the subsequent output:

```
SECGROUP Name Description
SECGROUP default Default
IP-RANGE IP Range Status
IP-RANGE 192.0.4.0/30 authorized
```

Also, to check the state of the authorization, use the following command:

```
rds-describe-db-security-groups default --headers
```

- Authorize the present EC2 security group using the following command:

```
rds-authorize-db-security-group-ingress default --ec2-security-
group-name myec2group --ec2-security-group-owner-id 123456789021
```

You will get the subsequent output:

```
SECGROUP Name Description
SECGROUP default default
EC2-SECGROUP myec2group 987654321021
authorizing
IP-RANGE 192.0.2.0/30 authorized
```

Once a database instance is created, you can use any supported tools for the database engine that the instance uses to connect. For example, you can use the MySQL command-line tools to connect to the instance you just created. To connect to a database instance using the MySQL monitor use the following command:

```
mysql -h uchitinstance.abcedefhijklsm.us-east-1.rds.amazonaws.com -P 3306
-u root –p
```

You will be able to see output similar to the following:

```
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 350
Server version: 5.3-log MySQL Community Server (GPL)
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql>
```

There are several other commands from which you can maintain, reconfigure, and manage your RDS listed as follows:

```
rds-authorize-db-security-group-ingress
rds-create-db-instance
```

```
rds-create-db-parameter-group
rds-create-db-security-group
rds-create-db-snapshot
rds-delete-db-instance
rds-delete-db-parameter-group
rds-delete-db-security-group
rds-delete-db-snapshot
rds-describe-engine-default-parameters
rds-describe-db-instances
rds-describe-db-parameter-groups
rds-describe-db-parameters
rds-describe-db-security-groups
rds-describe-db-snapshots
rds-describe-events
rds-modify-db-instance
rds-modify-db-parameter-group
rds-reboot-db-instance
rds-reset-db-parameter-group
rds-restore-db-instance-from-db-snapshot
rds-restore-db-instance-to-point-in-time
rds-revoke-db-security-group-ingress
rds-version
```

You can try the preceding commands to learn their effects but do it step by step as per the Amazon guidelines.

Summary

With the services in this chapter, you can manage your databases and their life cycles on the AWS infrastructure. For start-ups and developers, you need some basic knowledge of AWS tools and services. Even on the AWS website, you can find very basic to advanced level material to start with database services.

So, in the upcoming chapters we will talk about Big Data with the AWS platform. Users will come to know how to leverage the benefits of the AWS platform for Big Data and its challenges.

4

Web Application and Batch Processing Architecture

AWS provides a vast variety of services, which enables architects, developers, and administrators to step up their ability for production environments while working with customers and focused IT or non-IT applications on Cloud.

Batch processing on AWS permits on-demand provisioning of a multipart job processing architecture, which can be utilized for sudden or late deployment of a diverse, ascendable "grid" of worker nodes that can swiftly process large batch jobs in correspondence. There are various batch-angled solicitations in place in this era that can influence this style of on-demand processing, incorporating claims processing, enormous scale conversion, media transcoding, and multi-part data processing work. There are a couple of application management services available in the AWS basket, which can really help you leverage the benefit of the AWS infrastructure and managed platform to build your desired solutions by using management services. Before jumping into batch processing, we will see the monitoring service (Amazon CloudWatch) and one of the best deployment services (AWS CloudFormation) to understand their behavior so that you can utilize them in your infrastructures/applications on AWS. In this chapter, you will learn how to manage your application by following AWS application monitoring and deployment services such as the following:

- Amazon CloudWatch
- AWS CloudFormation

Moreover, after getting a brief overview of the preceding services, you will go through how batch processing works on the AWS platform and how you can host your simple static or dynamic website easily. In this chapter, we will cover the following topics:

- Alarms with Amazon CloudWatch
- Batch processing flow
- Amazon CloudFormation
- A case study on Web application hosting

So, let's start with the first important service: AWS CloudWatch.

Alarms with Amazon CloudWatch

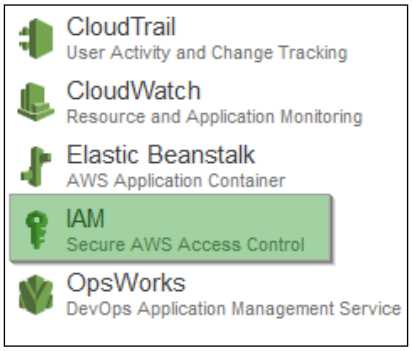
Amazon CloudWatch is the best ever monitoring service from Amazon to monitor all of your Cloud resources and applications on AWS. Amazon CloudWatch provides metrics to monitor your resources by collecting and tracking data from your resources. You can utilize Amazon CloudWatch metrics to gain insights of resource utilization, application performance, and instance operational health. So, let's go through it with some exercises, as practice makes perfect!

Amazon CloudWatch generally provides predefined metrics, such as CPU usage, Disk Read, Disk Writes, and so on. You can utilize these predefined metrics by default. However, if you want to create some custom monitoring metrics, follow the ensuing steps. So, in this section, you will monitor the custom CloudWatch metric. For this, you have to perform the following steps:

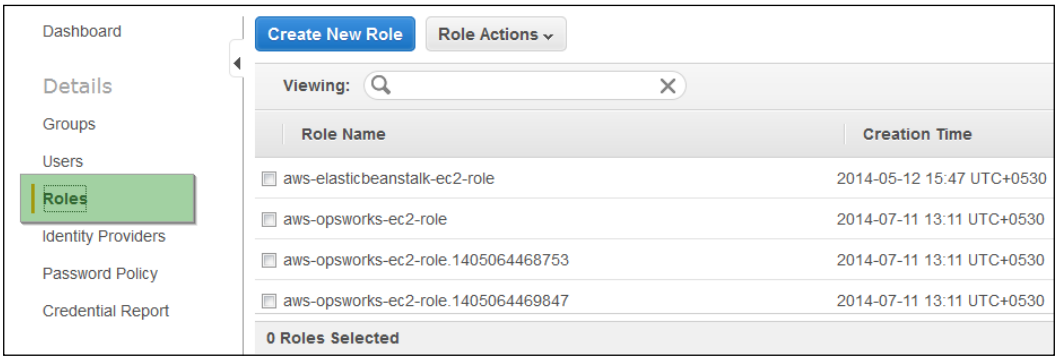
1. Create a custom IAM role for an EC2 instance so that the instance can have permission to write statistics to CloudWatch.
2. Launch a new EC2 server instance.
3. Use **Secure Shell** (**SSH**) to log in to the server and generate a custom CloudWatch metric.
4. Use CloudWatch to monitor this custom metric.
5. Create a CloudWatch alarm, which will be triggered whenever the custom metric you created drops below the level you mentioned.

Now, let's start with the first step, which is creating a custom IAM role for an EC2 instance.

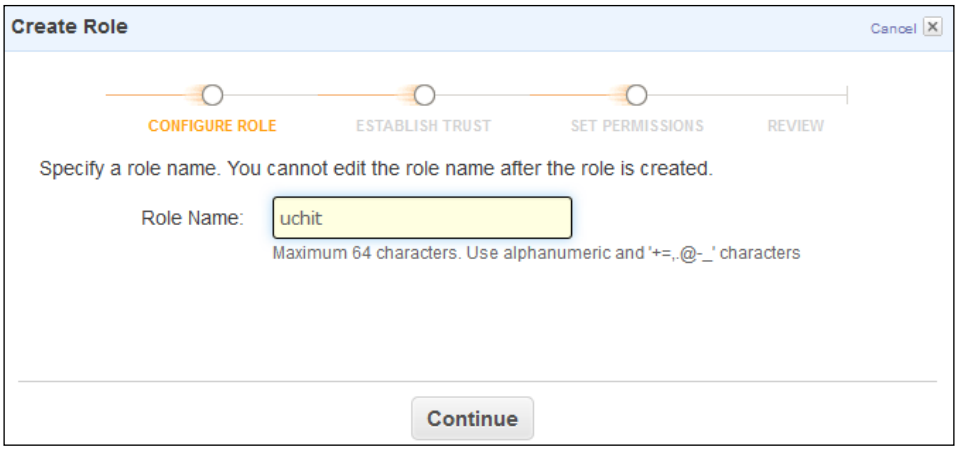
1. On the AWS Management Console, go to **Services** | **All services** | **IAM**.



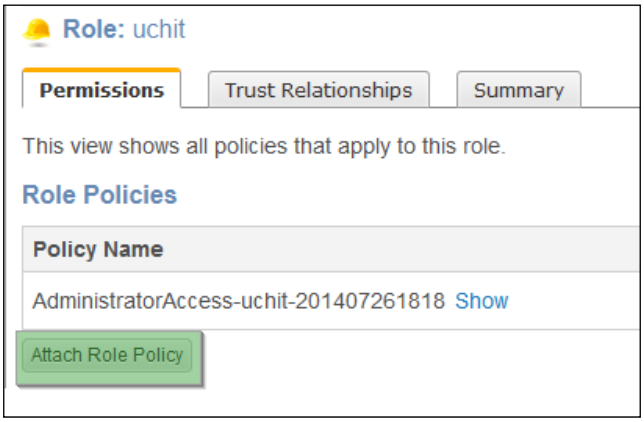
2. In the navigation pane, click on **Roles**.



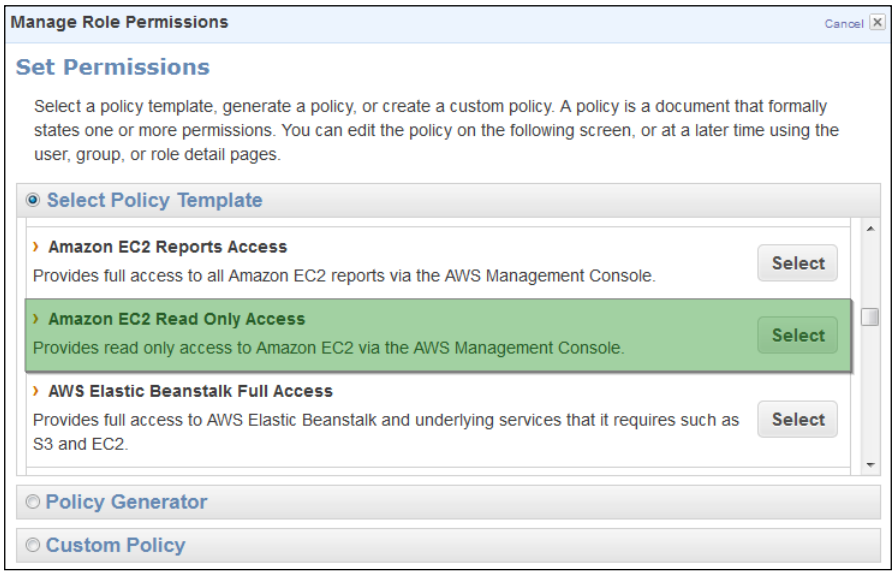
3. Click on **Create New Role** and provide parameters as needed. (For example, **Role Name**: `uchit` (you can give any name), in following screen, select AWS services role: **Amazon EC2**, and at last, select policy template **CloudWatch Full Access Template**).



4. Review the policy and click on **Continue**, and then click on **Create Role**.
5. Select the newly created role, and on the **Permissions** page, click on **Attach Role Policy**.



6. Under **Select Policy Template**, choose **Amazon EC2 Read Only Access**.



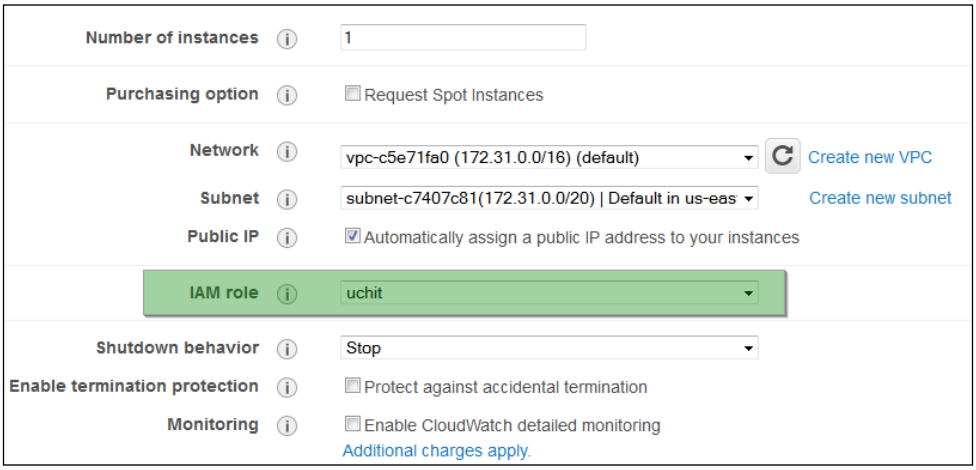
7. Follow the same steps from step 5 for accessing SQS from the policy template.

Creating an EC2 instance

Until here, you have worked from access and policy end, so from now on, you have to create an EC2 instance with a declared IAM policy. Here, you will monitor the instance performance in subsequent processes.

1. Navigate to AWS Management Console and click on **EC2**.

2. Specify the IAM role and launch the instance. Please make sure that you use the t2.micro instance type, which may come with the *Free Tier* service in your account.



Once you are done launching an instance, log in to an instance using SSH and use AWS CLI to generate a custom metric.

Make a note in the text file of your public IP address to use it further and it will be called "monitoring-client-ip" in the following steps.

To set the region value to your environment variables, please go through the following steps:

1. Copy the AZ value *minus the final letter* to a text file. This will be your region and will be referred to as `current-aws-region`.
2. Set the environment variable for region:

```
export AWS_DEFAULT_REGION=<cuurent-aws-region>
```

To check the region for our current instance, you can execute the following commands:

```
REGION='curl http://169.254.169.254/latest/dynamic/
instance-identity/document|grep region|awk -F\"
'{print $4}''
echo $REGION
```

In my case, the output is shown as follows:

```
us-east-1
```

3. Use the following command to create a CloudWatch metric:

```
aws cloudwatch put-metric-data --namespace  Uchit --metric-name
Test  --value 8 -debug
```

This command will create the sample metric. Also, by using this command, you can create any custom metric such as performance metric or usage metric. You can also provide free memory metric or disc space metric to get some fruitful practices. Now, you can see this metric on your CloudWatch dashboard. To monitor the custom metric in CloudWatch, perform the following steps:

1. On the AWS CloudWatch Management Console, navigate to **Metrics** | **Namespace** | **Custom Metrics**.
2. After selecting **uchit**, a normal graph will be displayed.
3. For the CLI console, use the following command to retrieve statistics on the CLI console, so the following command will trigger the alarm whenever the metric falls down from the designated level:

```
Aws cloudwatch get-metric-statistics --metric-name "Uchit"
--namespace="Test" --start-time=$ (date -d yesterday -I) --end-
time=$ (date -d tomorrow -I) --period=400 --statistics="Maximum"
```

To monitor a Custom CloudWatch metric, perform the following steps:

1. Under **Alarm Summary** in the CloudWatch Management Console, click on **Create Alarm**.
2. Select the custom metric and configure **Alarm threshold**; for example, provide **Name**, **Description**, and **Attention Level**.
3. Under **Actions**, provide state of the alarm and notification alerts settings.
4. Finally, click on **Create Alarm**.

To record a data point with CloudWatch that can trigger the alarm from the SSH console, use the following command:

```
aws cloudwatch put-metric-data --namespace  Uchit --metric-name Test
--value 4 -debug
```

To check the status of the alarm, you can verify it from the AWS CloudWatch monitoring console. That's it!

Batch processing flow

AWS allows multiple job processing architectures via batch processing for the instantaneous deployment of scalable nodes, which can quickly set up and perform tasks in parallel. There are plenty of batch-oriented apps in the market today that can leverage on-demand processing and large scale transformation work.

Batch processing architectures are identical with highly changeable practice patterns that have significant usage (for example, finance year end processing) followed by major periods of underutilization. There are many ways to start with a batch processing architecture. Here, you will go through basic batch processing steps by horizontally using the Amazon EC2 to compute and Amazon SQS for message queuing. To create a batch processing cluster, you will use the AWS management console. You will perform the following tasks:

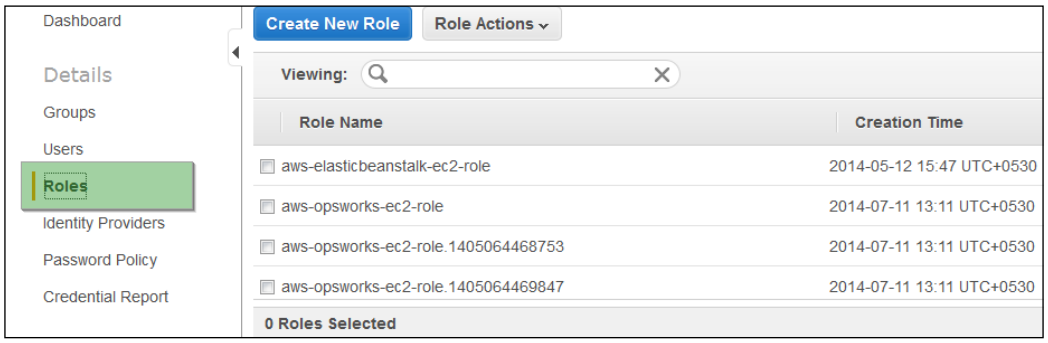
- Launch and configure an EC2 instance, which will work as a template for the worker node in your cluster
- Create the AMI from the instance
- Use SQS to task queues to pass messages to your EC2 instances
- Launch Auto Scaling Group
- Schedule work via a task queue
- Observe the output

In this batch processing cluster, the worker nodes in your cluster will convert a number of different images into a single montage image. A worker node will download images from the URLs provided by you and integrate those into a single montage image using the *ImageMagick* tool. Let's start this operation one by one. You have to start with creating an IAM role.

Creating an IAM role

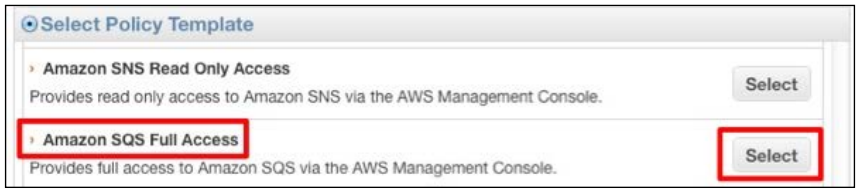
Your batch processing node will communicate with the queuing service SQS to get processing instructions and will keep output from node into S3 bucket as per given instructions:

1. Go to the AWS IAM console and in the left pane, move to **Roles** and click on **Create New Role**.



   Perform the following operations in this window:

   - Give the role a name, such as `BatchProcessing`, and click on **Continue**.
   - In **Select Role Type**, select **Amazon EC2**.
   - Locate the **Amazon SQS Full Access** policy and select it.



   You have to add additional permissions to S3.

2. Select the newly created role and click on the **Attach Role Policy** button at the bottom of the window to attach other policy.

3. Find Amazon S3 full access and apply it as another policy. So far, you are done with IAM Roles and Policy creation.

Now it's time to move towards the EC2 section to launch the master EC2 instance. Launch an EC2 instance that has Amazon Linux with a configuration script for the ImageMagick tool and the batch processing software, as shown here:

1. In the configure Instance Details panel,
    - Provide role as **BatchProcessing**
    - Expand the **Advanced Details** section (scroll down to locate it)
    - Provide the configuration script as user data, as shown in the following code:

```
#!/bin/bash
yum install -y ImageMagick
easy install argparse
mkdir /home/uchit/jobs
wget -o /home/uchit/image_processor.py http://goo.gl/8e3WdB
```

Before using the `wget` command in the bash script, it is recommended that you download `image_processor.py` first manually from the given URL and use your own S3 bucket to store and retrieve it.

2. Configure security group called batch processing and make sure that you have port 22 (SSH) opened.

3. Review the configuration and launch the instance.
4. Log in into your instance.

After you log in, you can see the `jobs` directory and the `image_processor.py` script.

For Mac or Linux OS, there is no need to convert a `.pem` file to `.ppk`, but you have to change the permission to `600` for the `.pem` file to log in using SSH into the instance.

It's time to create the AMI from the launched instance, so go to Amazon EC2 Management Console.

1. Select your respective instance and from the **Actions** menu, select the **Create Image** option.
2. Provide the necessary details such as image name, image description, and so on, and create it.

Your instance will be rebooted once and you will lose SSH connectivity at this point of image creation.

Initially, the AMI will be in a *Pending* state and eventually it will be in a *Available* state.

Creating SQS tasks

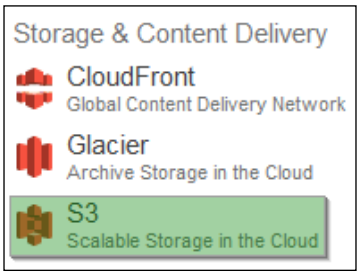
To dispatch work from the input queue and view the results via the output queue, you have to go to Amazon SQS console and perform the following steps:

1. Click on **Create New Queue** and configure it by providing the necessary inputs such as the following:
    - Queue name: input
    - Default visibility timeout: 90 seconds
2. Create another queue named the output queue.
3. Now, select the input queue and from the **Queue Actions** menu, select **Send a Message**.
4. Provide the image URLs which you want use here. You can also put your images in the S3 bucket and provide the S3 URL.
5. Click on the **Send Message** and **Close** button.

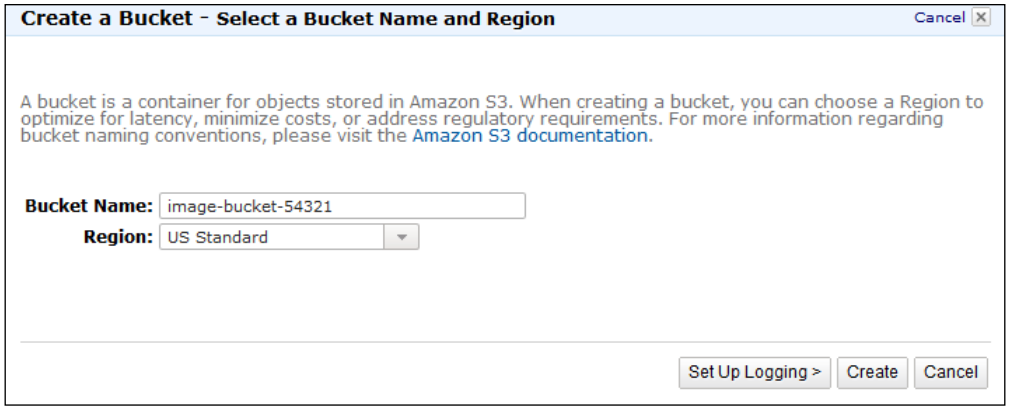
Creating S3 bucket

To hold the output of your worker nodes, you need to configure the S3 bucket for it as shown here:

1. Select **S3** from the services menu.



2. Click on **Create Bucket** and provide the necessary information.

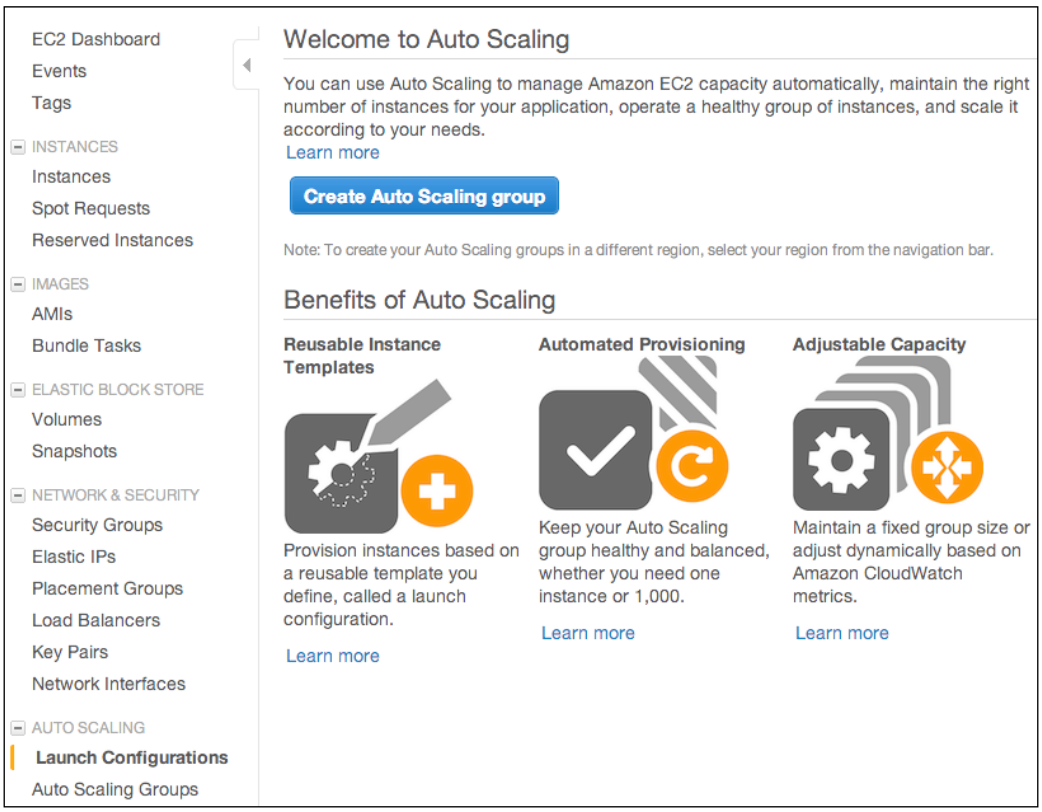


3. Click on **Create** and you are ready to go with S3 too.

By creating all this, you are ready to launch worker nodes within an Auto Scaling group.

Launching worker nodes

Using AWS Management Console, you can launch configurations and Auto Scaling using a single mouse click, and also bid for spot instances within this console when required.



To start with the configuration, please follow the given steps carefully:

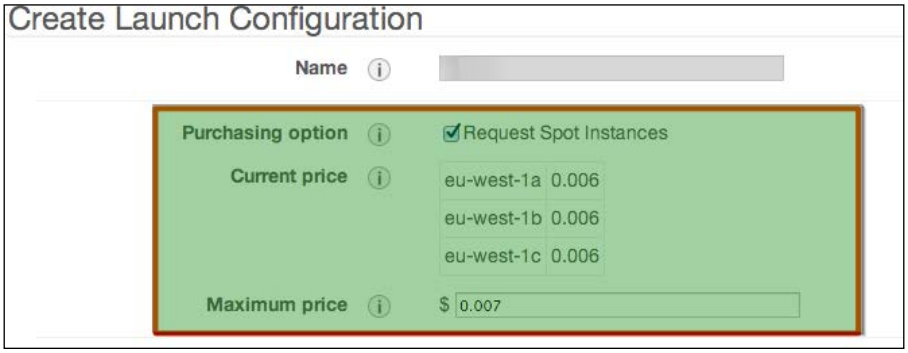
1. Creating a launch configuration.

   So, here, we have to select the AMI and its relevant architecture, which suits our application. You have already created the AMI, so select that AMI from the AMI section (from the left pane) of EC2 Management Console.

2. Provide the configuration details. You can provide information in the **Advanced Details** section, which lets us choose the AMI's Kernel and RAM disk and optionally fill in a user data block.
3. You can provide spot instance bid details, if you want.

You can select an on-demand instance also because there is nothing specific to go only with spot instances, but via spot one, you can reduce your OPEX.

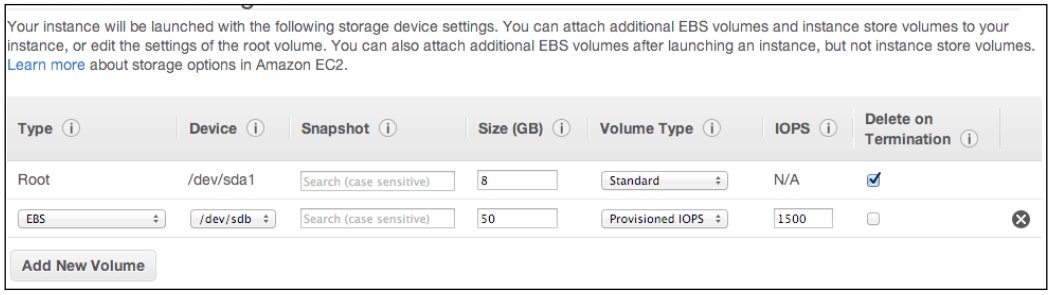
   Here, in the configure details panel, you have to provide Name as `Workers` and IAM Role as `BatchProcessing`.



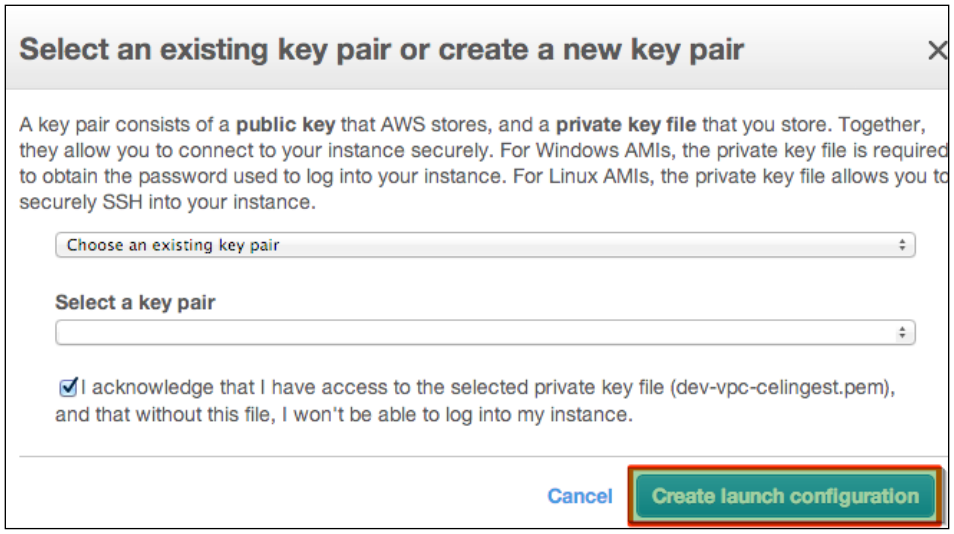
4. In the **Advanced** Section, provide your scripts in the **User Data** option, as shown here:

```
#!/bin/bash
/usr/bin/python /home/ec2-user/image_processor.py &
```

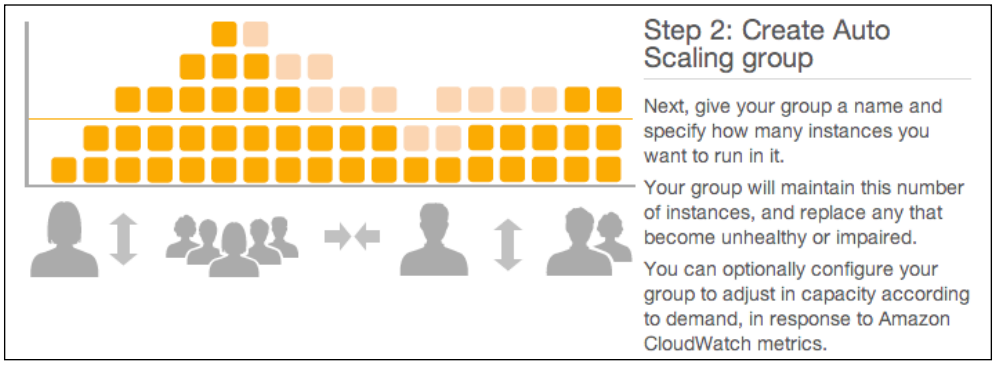
5. Provide your necessary storage options in the form of the **Root** device size, EBS volume size, and so on.



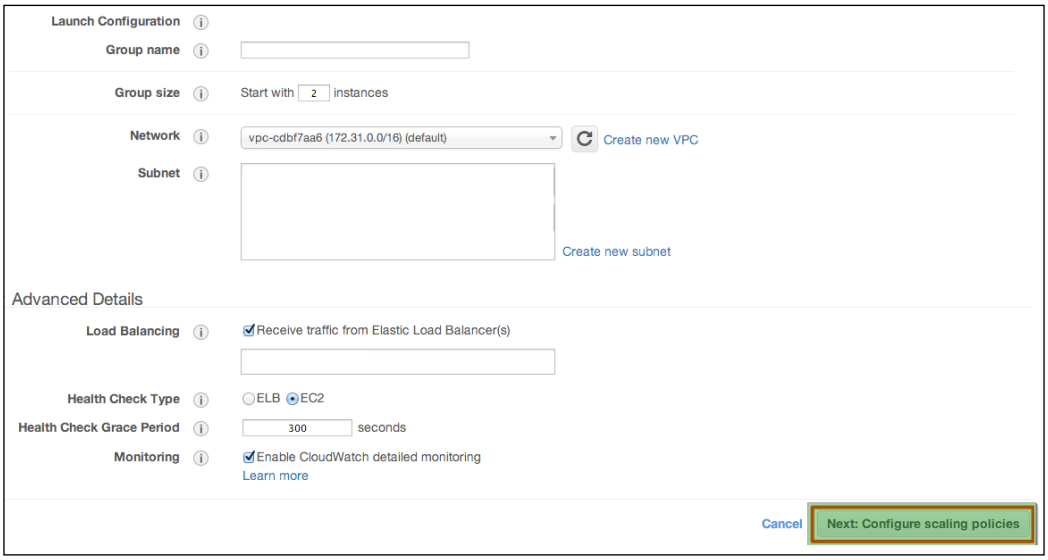
6. You can select an appropriate **Security Group** or just add a new customized one named `BatchProcessing`, where you restrict SSH access to your IP while leaving HTTP and HTTPS access open to be accessed globally.
7. Finally, just last tab **Review** to review, and eventually, change the previous choices, if required.
8. Finally, choose the key pair to access your instances and click on **Create launch configuration**.



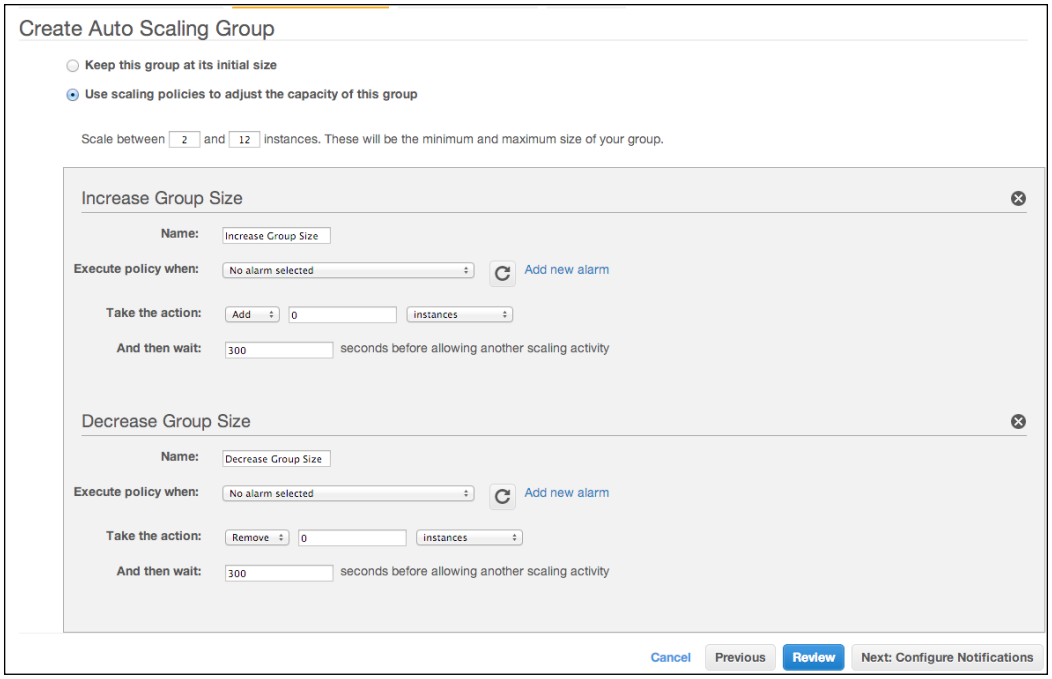
So far, you have completed the launch configuration for the AWS Auto Scaling service, but you have to configure the Auto Scaling group based on a schedule or policy. So, let's create an Auto Scaling group for your web application.



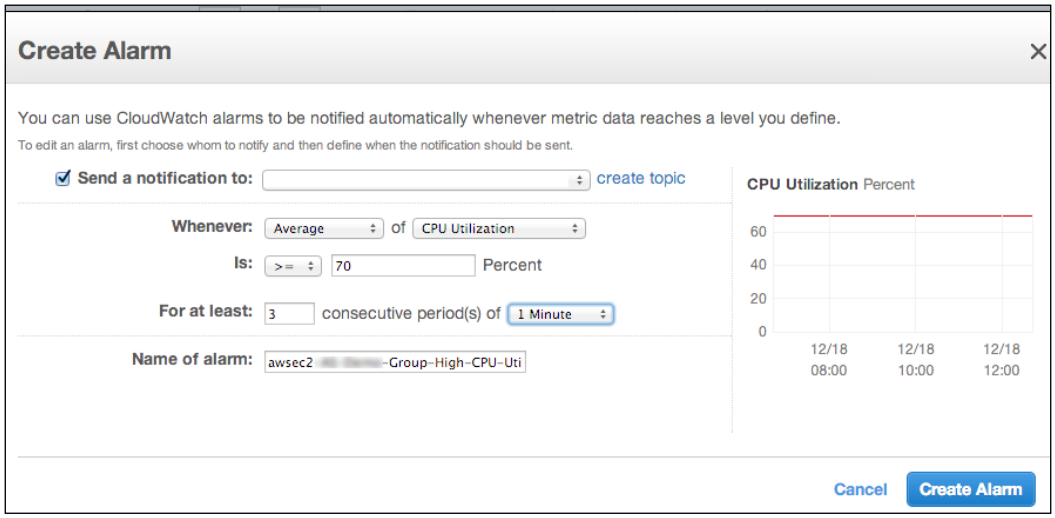
1. In the very first tab, there will be a few choices, for example, group name, size, network, subnet, and so on, with some advance details. Here, in advanced details, you can call AWS load balancer, which will route the traffic based on policies. Fill the appropriate details and click on **Next: Configure scaling policies**.



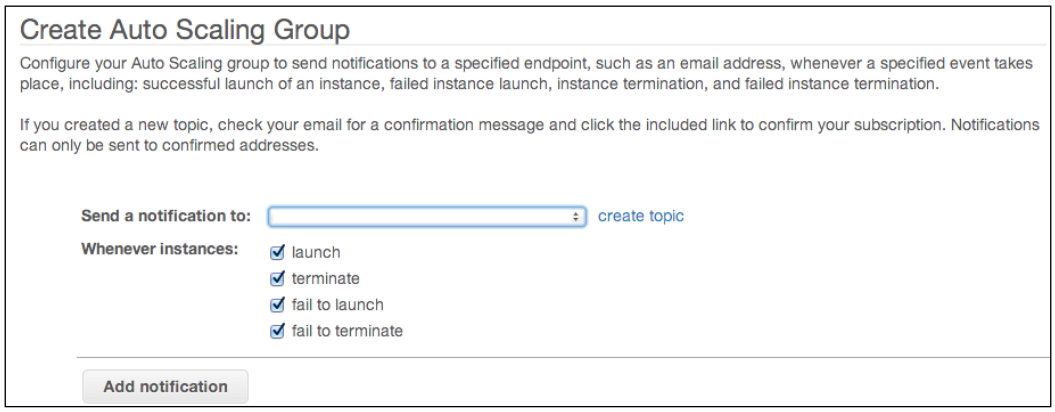
2. The next tab is very important as it allows you to select between two fundamental scaling plans. There are two options:
    1. **Keep this group at its initial size**, which will ensure that your group figures out a number of healthy instances equal to the initial size you mentioned. At any point, if an instance fails the checks, it will be replaced automatically.
    2. **Use scaling policies to adjust the capacity of this group**, in which you can select a CloudWatch alarm based on scaling policies to increase or decrease the number of instances.



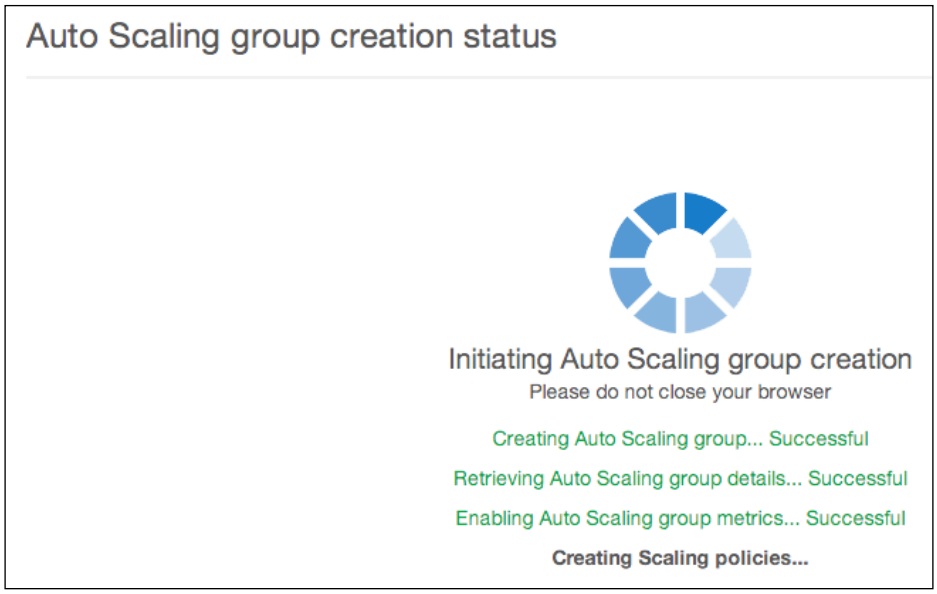
3. You can set policies for the minimum and maximum number of instances initiating your Auto Scaling group. You can add CloudWatch alarms easily later, or during deployment, but here, we'll add the one to allow your group to increase its size, as shown in following screenshot, and vice versa, you can configure to decrease the group size as well:



4. You are almost done, so now it's time to configure the notifications relative to your group to get updates:



5. Finally, again, review the configuration in the **Review** tab, and after clicking on the **Create Auto Scaling Group** button, you will be redirected to a status screen showing the creation of your resources. If something fails, you're prompted to retry the single resource initiation again with correction.



So, after initiating, you can see the new scaling activity on your EC2 dashboard. And whenever required, you can make changes in these configurations as per the requirement.

Dispatching work and viewing results

You have to use SQS Management Console to put some more messages into your *input* SQS queue. Your worker node will expect a new line delimited list of URLs of images.

Choose your *input* queue and confirm that you have one message in the queue. Even if one message does not exist, check the following things:

- Worker node
- IAM role configuration
- Queue names
- BatchProcessing role for instance

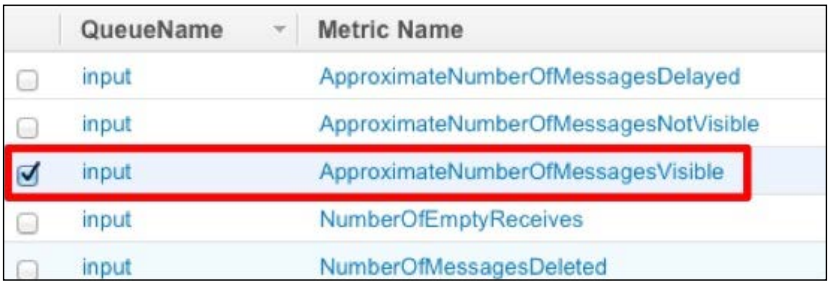
The following are the steps to view the output from *output* queue:

1. Select the output queue and from the **Queue Actions** menu, select **View/Delete Messages**.
2. Click on **Start Polling for Messages**.
3. Locate your message and click on **More Details** to view the message body.

Monitoring the cluster

You can now go to the Amazon CloudWatch to monitor your cluster. As you already defined the CloudWatch alarm at the time of creating Auto Scaling policies, you need to check metrics from the **Services** menu. To do this, perform the following steps:

1. Click on **Browse Metrics**.
2. Click on the **SQS metrics** header.
3. Choose the line for:
    - Queue Name: **input**
    - Metric Name: **ApproximateNumberOfMessagesVisible**

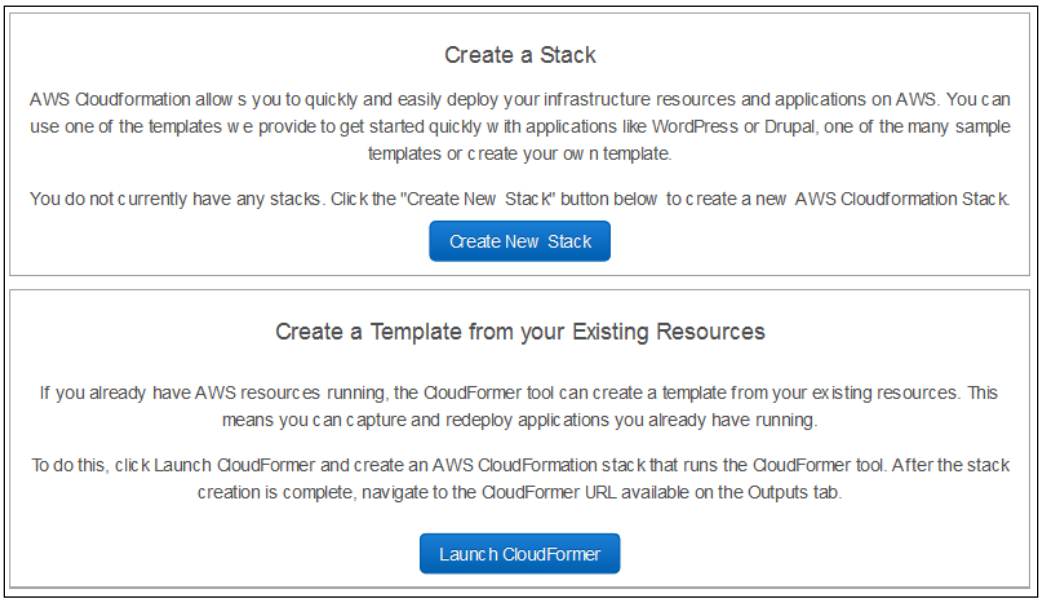


Check the configuration and you are done. Based on your Auto Scaling policy, your cluster will work efficiently for the load. On AWS, there are some specific AMIs that are available, which can help you generate a load, as well as some utility commands such as "stress", which will burst your CPU.

Amazon CloudFormation

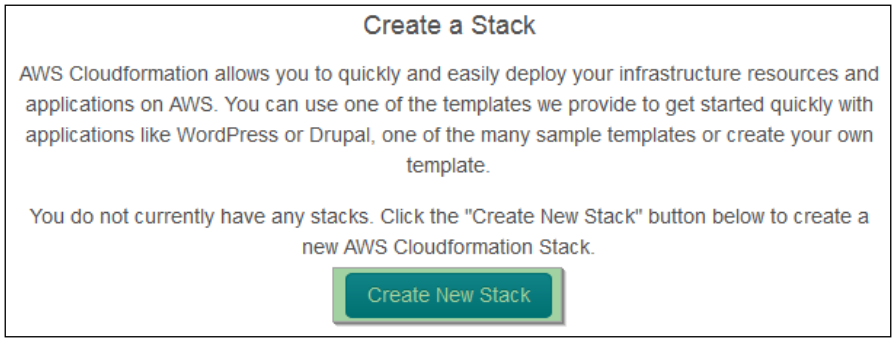
Amazon CloudFormation is a way to launch your Cloud environments without difficulty. That is, when you launch a CloudFormation environment, you will be able to launch specific AMIs with scrupulous key pairs, on predefined instance sizes, and given load balancers. Also, if any segment of your environment fails to launch, the environment rolls back to previous state, terminating all the pieces down the way.

Of course, it's not as simple as that. Every so often, assets need to know about each other (for example, what connection string should an EC2 instance use to connect to your RDS instance that CloudFormation just launched). To assemble a set of Amazon CloudFormation templates that enable a typical enterprise environment, you will have to use a tool that inspects your running environment and creates a CloudFormation template for you. You can find Amazon CloudFormation Management Console, as shown here:

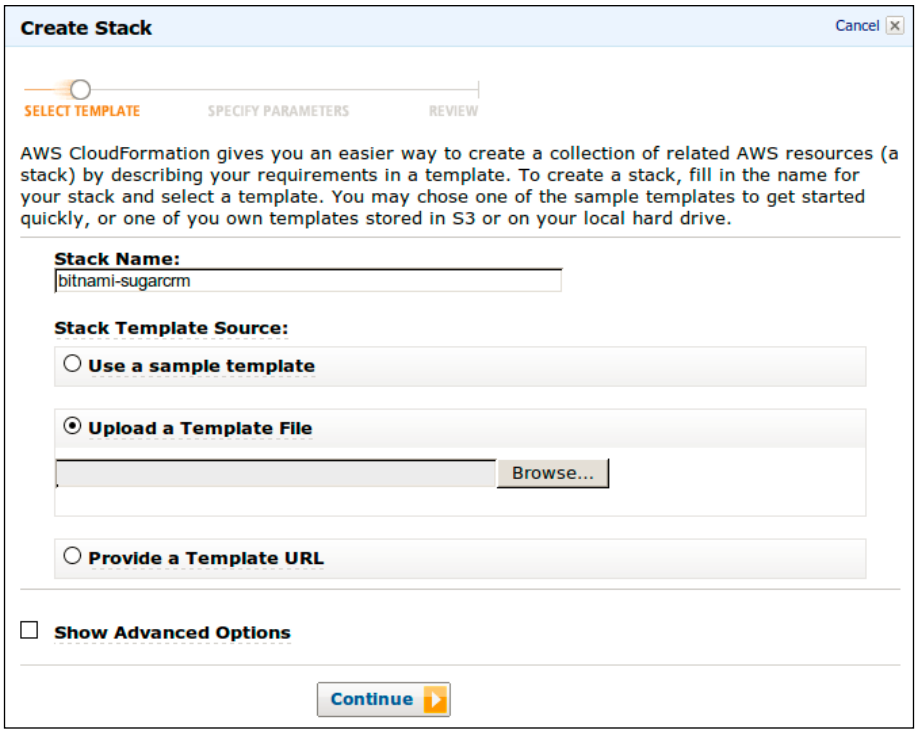


Here is an example: let's start with deploying the SugarCRM server using a template file. For this, you need to save the template from the URL `http://goo.gl/kCjGwE` as the `sugarcrm.template` file on your local system. Now, you have to just execute the following instructions to launch the template with a full stack of resources.

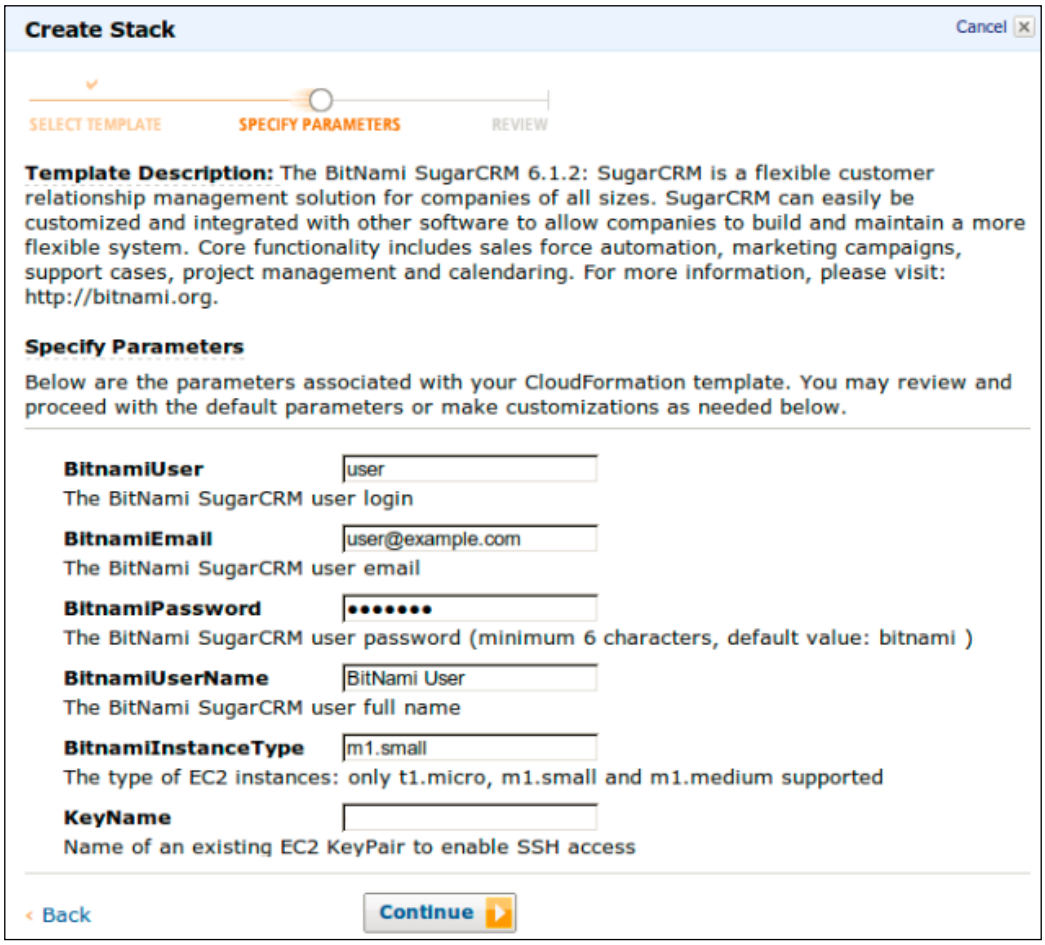
1. Click on **Create New Stack** on AWS CloudFormation console.



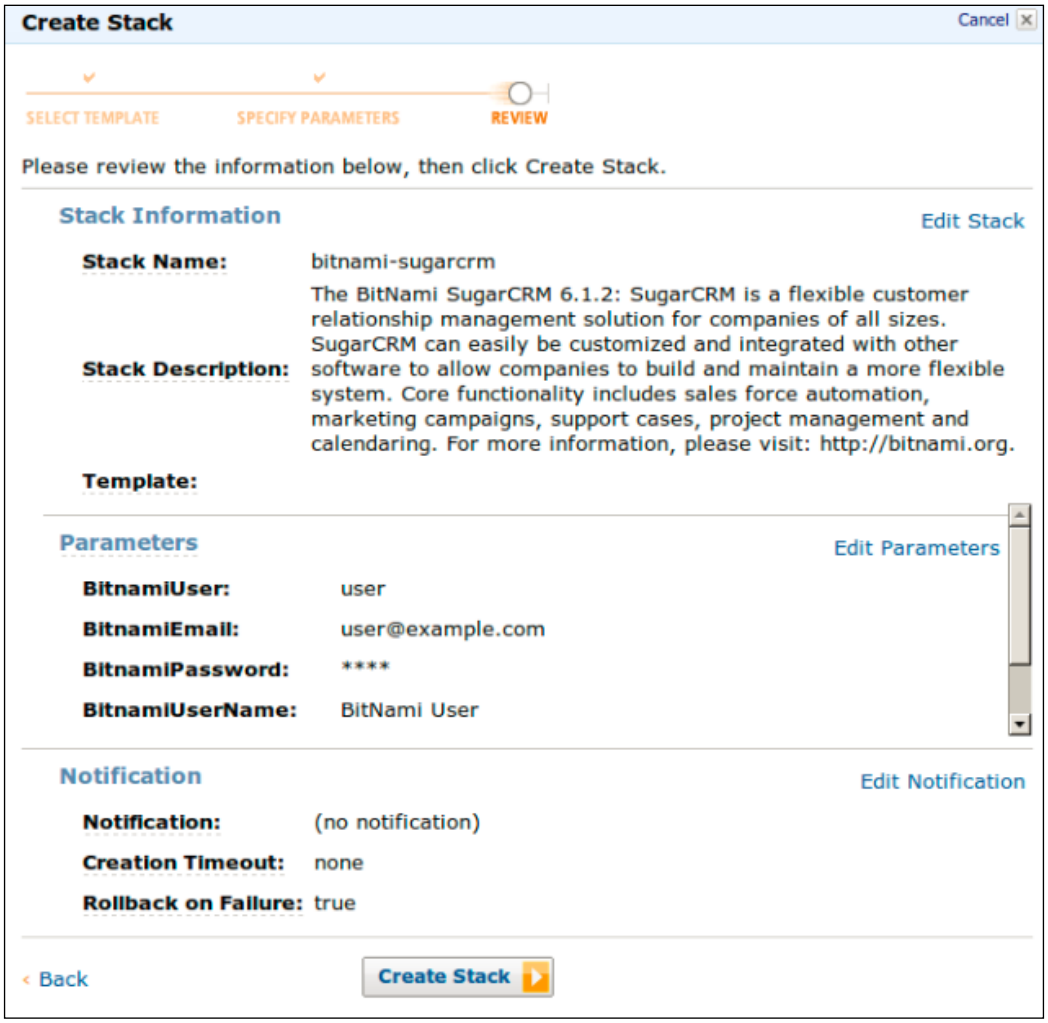
2. Choose your **Stack Name** as you want, but remember that it must not include whitespaces in name.



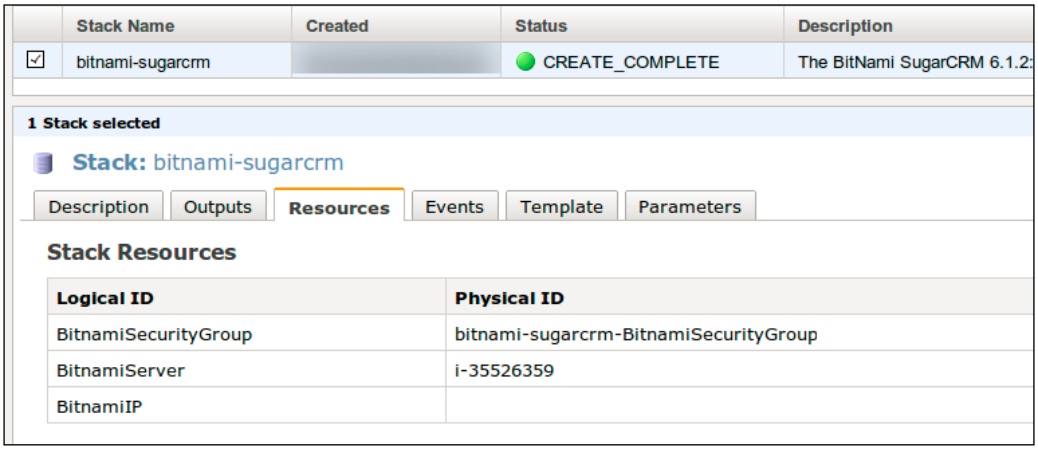
3. Provide your application credentials, such as the username and password.



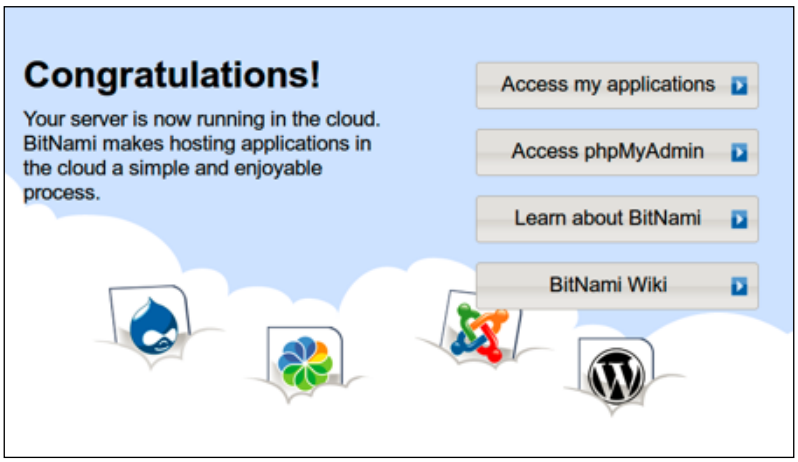
4. Review your configuration and click on **Create Stack**.



The setup process will take some time, around 5 minutes, so please have some endurance and wait. Meanwhile, you can create an elastic IP if it's not already created.



After a successful template creation, enter the Bitnami IP value in the web browser window and you will see the start page, as shown here:



You can delete the new environment whenever you want by simply deleting the stack that you have just created.

In this way, you can launch your application without touching a number of services on the Amazon stack. Simply go with Amazon CloudFormation and deploy your template; Amazon will take care of the rest.

Where should I start on AWS?

AWS provides a wide range of services. It is a common confusion for every newcomer to get confused about what and where to initially start, and here, our current section provides you with an overview of where should we start and how we move ahead in our journey of the Cloud. Here are a few very common areas of services:

- Hosting a simple web app
- Storage
- Managing resources

These basic services can be learned using the free tier package itself.

Case study

As an example, let's take a look at how to get your LAMP environment ready on AWS in the following case study.

The following is a real-life case study.

LAMP on your Amazon EC2

Let's look at the procedure of getting your LAMP (Apache, PHP, MySQL combination is often known as LAMP stack or LAMP server) stack working on your Amazon EC2 instance as a case study here.

Here, I explained how to launch LAMP on a single EC2 instance for simplicity. You can use an EC2 instance for Apache and PHP, while you can leverage RDS to launch your MySQL.

Prerequisites

Before actually getting started with the LAMP stack, let's take a look at the following prerequisites:

- A launched instance with a public DNS name reachable on the Internet (refer to the preceding sections)
- A security group that is well configured to allow SSH (Port 22), HTTP (Port 80), and HTTPS (Port 443) connections respectively

Installing and starting the LAMP server

Let's try to dive into our instance and install the required packages.

- Connect to your instance, if you have any, or launch an instance, and then connect to it
- Install MySQL and its dependencies using the `yum group install` command (here, we will use Amazon Linux):

```
sudo yum groupinstall -y "Web Server" "MySQL Database" "PHP Support"
```

- Install the PHP – MySQL package using following command:

```
sudo yum install -y php-mysql
```

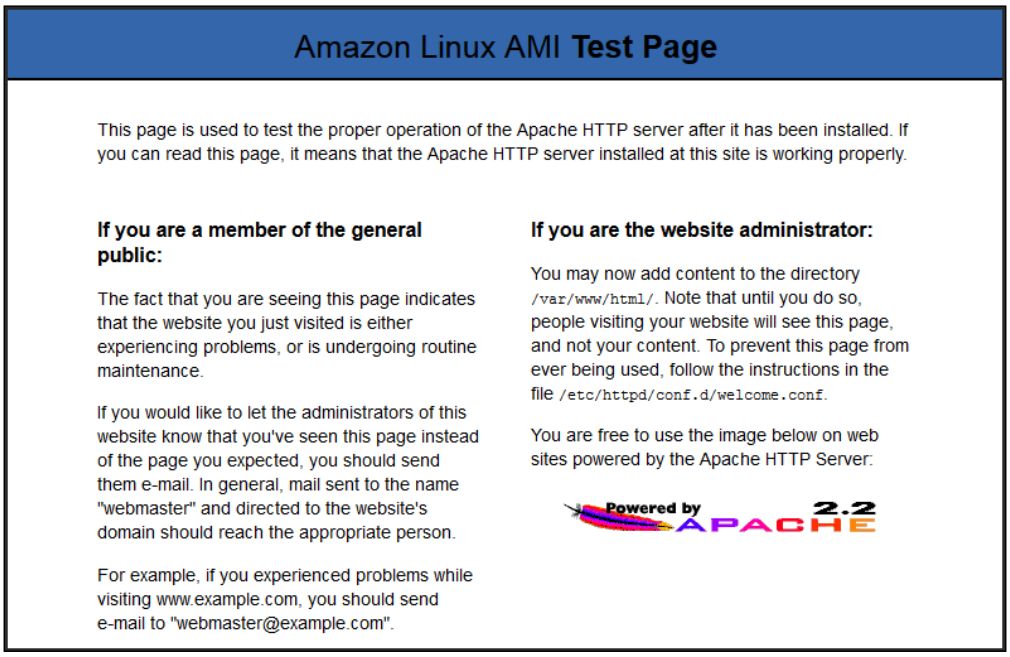
- Start the Apache web server using the following command:

```
sudo service httpd start
```

- Configure the Apache web server to start each system boot using the following command:

```
sudo chkconfig httpd on
```

- Test the web server by using a public DNS address in your browser (you should see the Apache test page as shown here):



File permissions

You have to modify some files permissions to make it accessible through the Web.:

- Add the `www` group to your instance using the following command:

  ```
  sudo groupadd www
  ```

- Add your username to the `WWW` group:

  ```
  sudo usermod -a -G www your-user-name
  ```

- Log out and log in again to check the membership with the `WWW` group
- Update the group ownership of `/var/www` (which is the Linux default) and its contents to the `WWW` group:

  ```
  sudo chown -R root:www /var/www
  ```

- Update the directory permissions of `/var/www` and its subdirectories to add group write permissions and to set the group ID on future subdirectories.
- Recursively, change the file permissions of `/var/www` and its subdirectories to add group write permissions.

Testing the LAMP web server

We can test the LAMP web server using the following:

- Create a simple PHP file in the Apache document root using the following command:

  ```
  echo "<?php phpinfo(); ?>" > /var/www/html/phpinfo.php
  ```

- Access this file from the browser using the public DNS address: `http://YOUR_DNS_NAME/phpinfo.php`. You should see the default PHP Info page.

Summary

In this chapter, we got an overview of Amazon CloudWatch and created a custom metric for it. Later, we moved to CloudFormation and have seen example of how to use the CloudFormation service. Finally, we saw how to create a batch processing architecture with an example.

Later, we saw how to host the LAMP stack on a single EC2 instance with minimal configuration. In this way, you can host your normal web application using different services of AWS as per your experience and convenience.

In the next chapter, you will learn about high availability, disaster recovery, and AWS Private Cloud (Amazon VPC). Users will come to know how to leverage the benefits of the AWS platform to build a private Cloud in it.

5
High Availability, Disaster Recovery, and Amazon VPC

The AWS Cloud offers a low-cost margin for maintaining a convoy of disaster recovery servers and data storage resources. With the AWS Cloud, you can take benefit, of geo-distribution and imitate the environment in other locations within minutes. Imagine that your hardware fails because of some unknown reason. Suppose that outages occur because of failure, or some catastrophe strikes your web application. What if you will be slammed with more than the expected number of requests per second for one day or one hour? What if, with time, your application software stops working? By being a cynic, you end up thinking about revival strategies in the blueprint stage, which helps in creating a superior system.

Virtual Private Cloud (**VPC**), unlike EC2, presents you with a virtual private network with power over routing tables, DHCP selection sets, and more. This has greater benefits than EC2. While VPC is a unique product that is specific to AWS and appears to *lock you into the AWS infrastructure/platform*, the model that VPC takes is similar to running your own dedicated hardware on AWS or in your data center.

In this chapter, we will cover the following topics:

- Disaster recovery circumstances with AWS
- Data replication
- Architecting with Amazon VPC

Disaster recovery circumstances with AWS

Disaster recovery (**DR**) is about preparing for and recuperating from a disaster. Any occurrence that has a negative influence on your production line's continuity / down time over Cloud or finances could be labeled a disaster. This could be hardware or application failure, a network problem, a power problem, physical disasters such as fire or flooding, human blunders, or other inconsequential disasters.

To reduce the impact of a disaster on the trade, companies devote time and resources to map, organize, practice, manuscript, instruct, and modernize processes to deal with such problems. The amount of speculation for the disaster recovery scheduling of a meticulous system can vary noticeably, depending on the cost of a prospective outage.

Recovery time objective and recovery point objective

There are two collective industry standings for cataclysm design:

- **Recovery time objective** (**RTO**): This is the duration of time and the service level to which a production method has to be restored after a debacle to evade intolerable penalty allied with a rupture in business continuity. For instance, if a disaster occurs at 01:00 P.M. and the RTO is 9 hours, the DR progression would guarantee recovery to the adequate service level by 10:00 PM.
- **Recovery point objective** (**RPO**): This depicts the satisfactory sum of data loss considered in time. For example, if the RPO was 2 hours, after the system was up again, it would surround all data up to 11:00 A.M. because the failure occurred at noon.

There are four DR scenarios that compare AWS Cloud infrastructure and services with traditional DR methods:

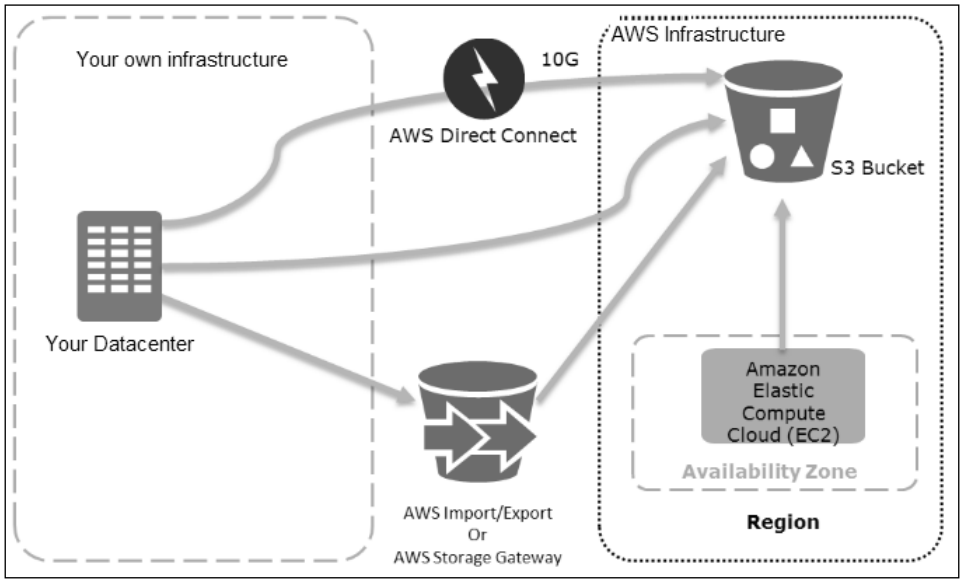
- Backup and restore
- Pilot light for modest recovery into AWS
- Warm standby solution
- Multisite solution

These are just examples of achievable approaches, and variations and combinations of these solutions are definitely possible.

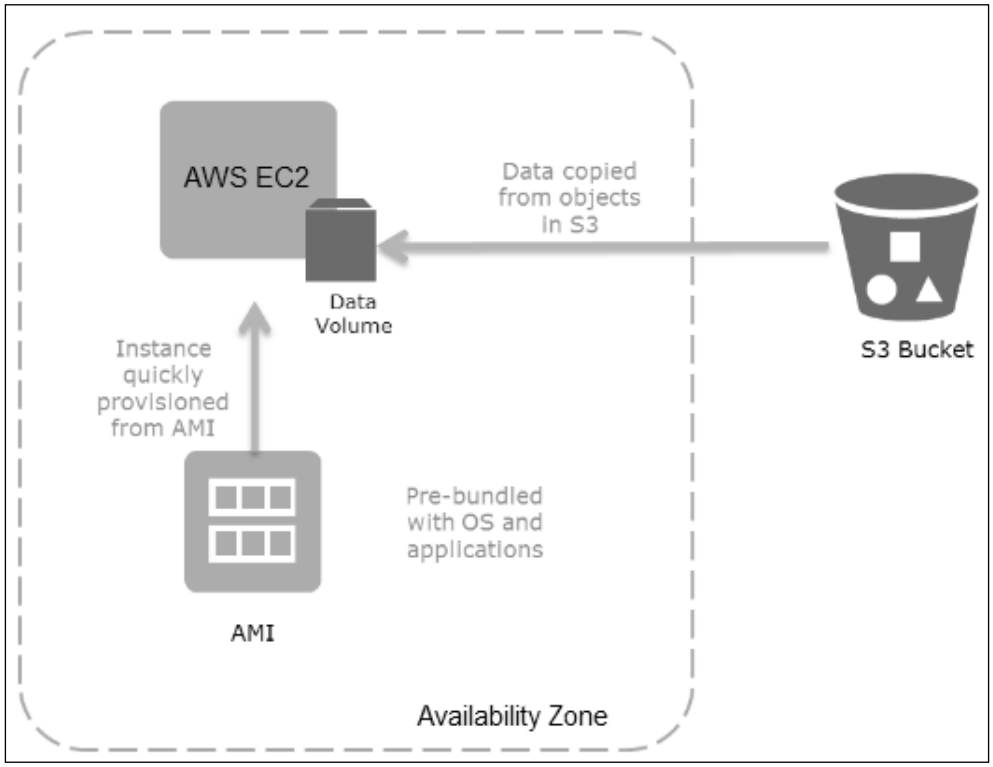
Let's talk about these DR scenarios one by one in detail.

Backup and restore

Data is often backed up to tape and sent off-site. In such cases, recovery time will be longer. Amazon S3 will be an ideal solution to back up your data, as it is designed to deliver 99.999999999 percent durability of objects over a prearranged year time. Moreover, the AWS Import/Export service empowers you to move large quantity of data by shipping storage devices straight to the AWS data center. The AWS Storage Gateway service can apprehend snapshots of your on-premise data and copy it to Amazon S3 as backup. You can use even EBS volumes from snapshots.



From the preceding figure, one can understand how to transfer data from your in-house data center to the AWS Global Infrastructure. Recovery of your data in a disaster scenario needs to be foolproof, swift, and consistent. The client should guarantee that systems are configured appropriately to handle custody of data and security of data, and have tested their data recovery processes before failure takes place.



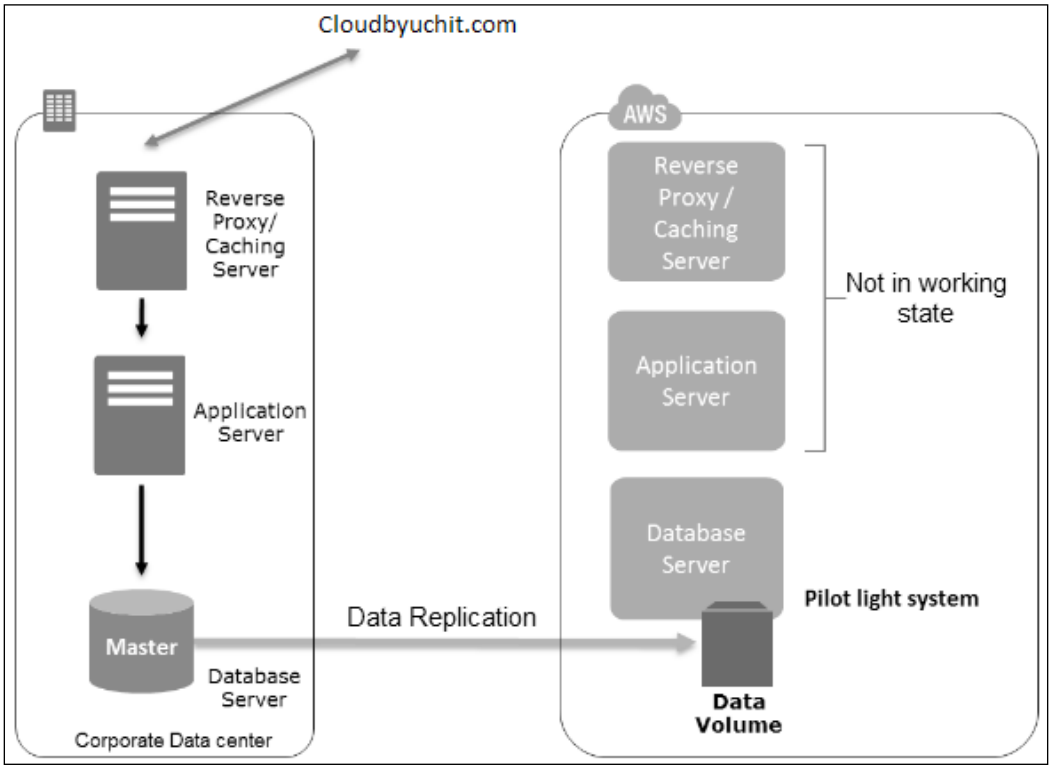
Pilot light recovery in AWS

The word "pilot light" came from the gas heater. Like in a gas heater, a small flame will be always on and can swiftly ignite the entire heating system as needed. This scenario is similar to a backup and restore scenario, but the critical core elements of your system are already configured and running in AWS all the time. When the requirement comes for recovery, you can swiftly provision a production environment for the critical core.

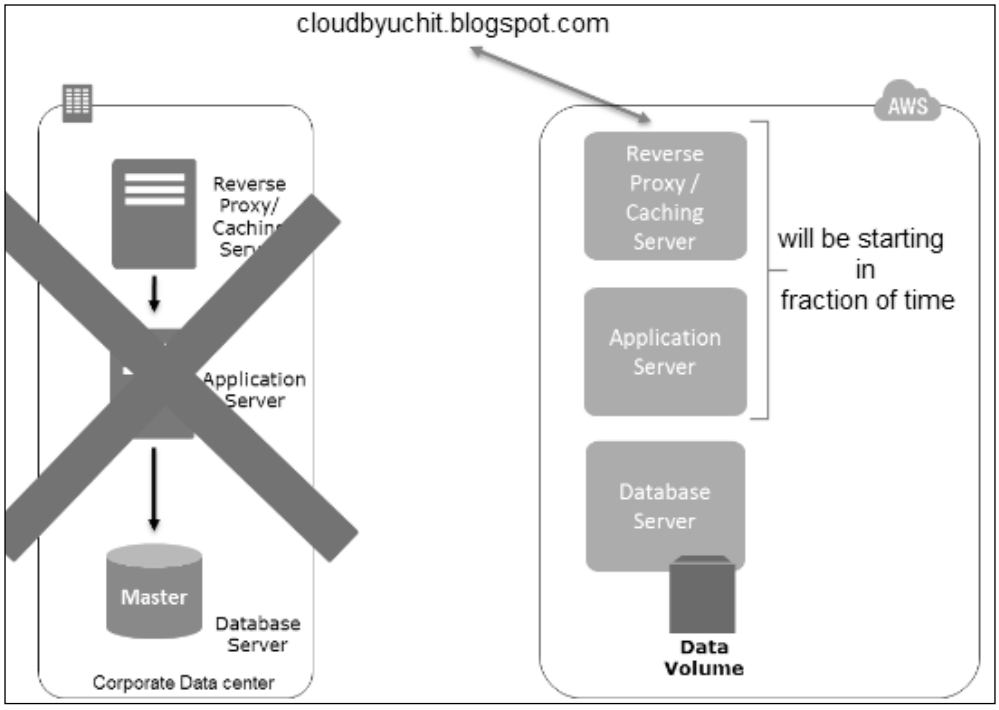
The pilot light method will give you a quicker recovery time than the "backup and restore" scenario, as the core elements of the system will be already running and will continue to be up to date.

There are basically two phases:

- **Preparation phase**: In this phase, you will be replicating/mirroring your data continuously in a period of time and testing your components.



- **Recovery phase**: In this phase, you will simply switch from the primary environment to the pilot environment until your primary environment is up and running.



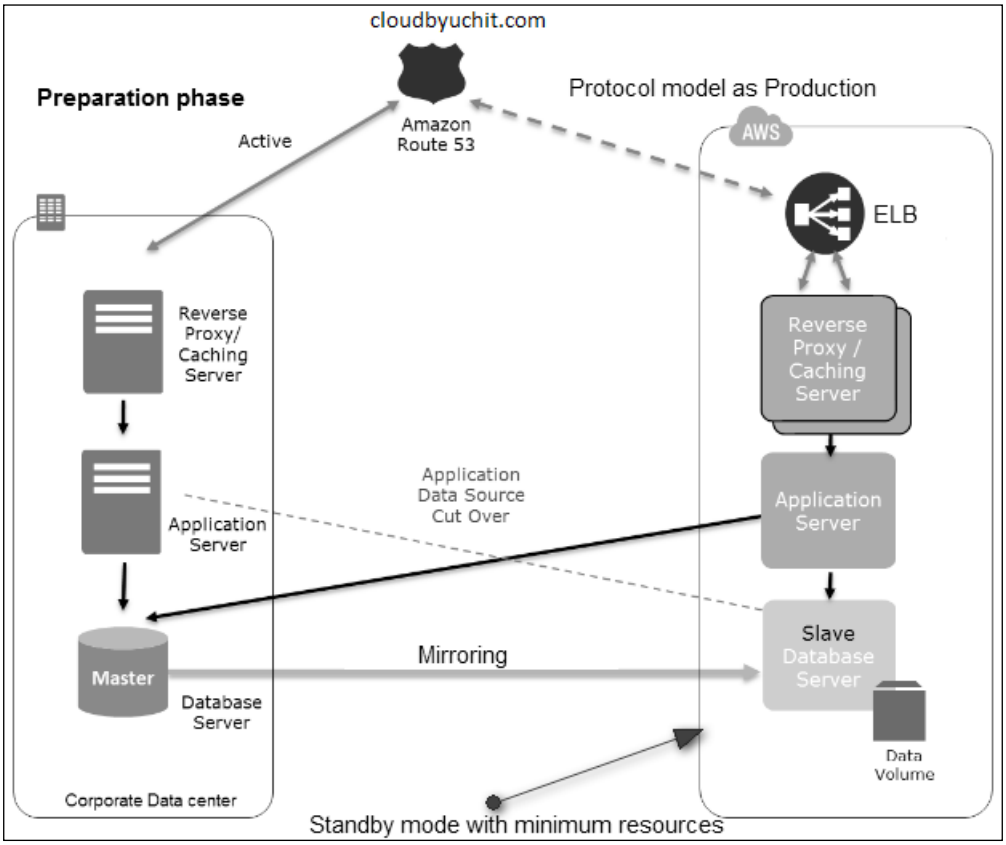
Warm standby solution

A warm standby solution broadens the pilot light method's fundamentals and preparation. It cuts the recovery time in advance because some services will always be running, unlike pilot light. By identifying your most critical components, you would fully duplicate those components on the AWS infrastructure and always have them up and running.

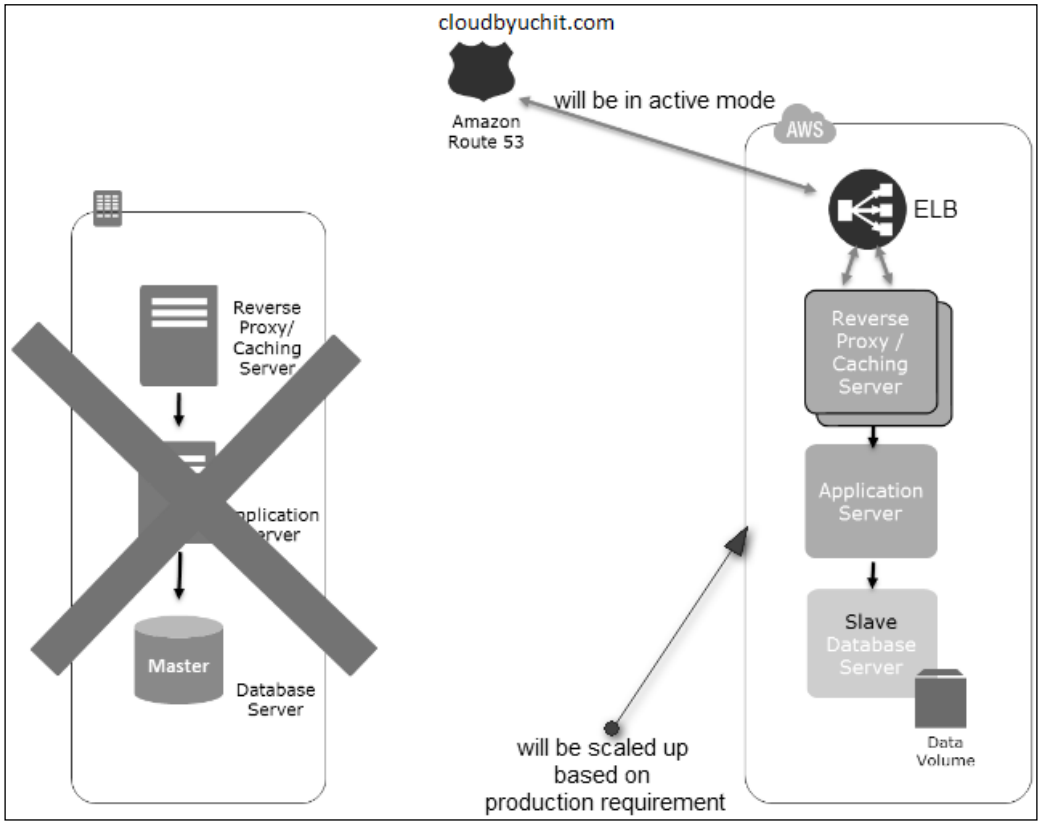
These instances can be running on a basic required size of AWS EC2 instances all the time. The warm standby model will not be scaled as production components who can load more traffic, but it will be like fully functional model. The instance may be used for non-production work, such as testing, quality assurance, internal exercise, and so on. At the time of disaster, this protocol model can be achieved by adding more instances to the load balancer or by resizing the small-capacity servers to run on larger EC2 instance types on AWS. However, resizing is not recommended as you may have to face downtime.

There are two phases:

- **Preparation phase**: In this phase, the AWS standby solution will run continuously with production components.



- **Recovery phase**: In case of failure, the standby environment will be scaled up automatically to serve production load and DNS entries will get changed to route traffic. So, you should create your environment in such a way that if a component goes from your infrastructure, the secondary component should take its place to come up with a full working infrastructure from the standby mode within moments.

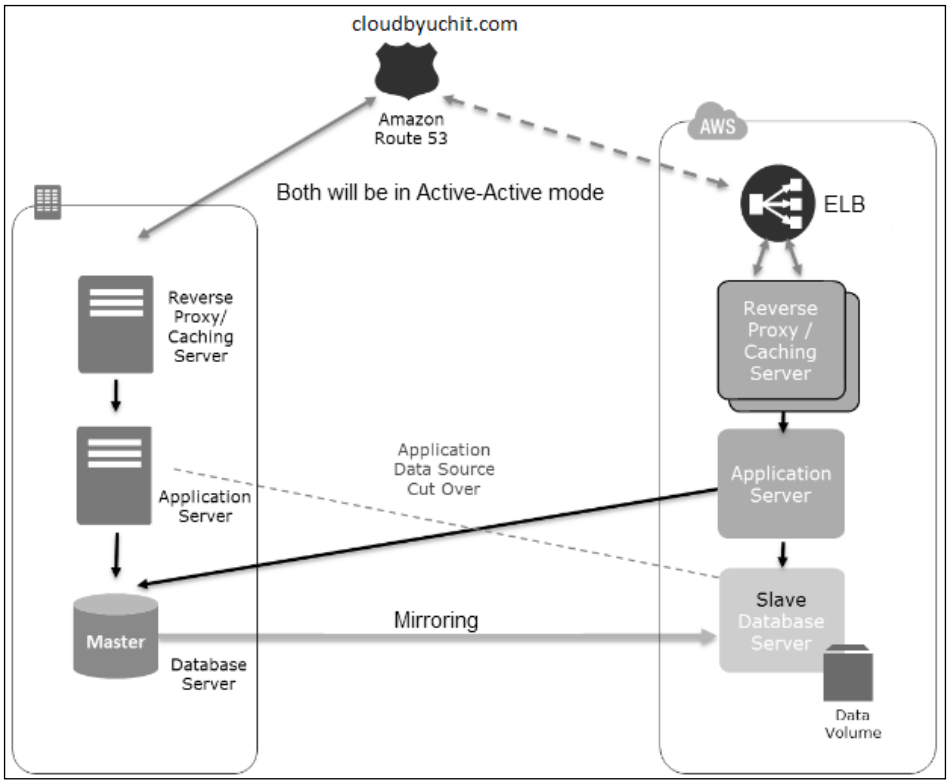


Multisite solution

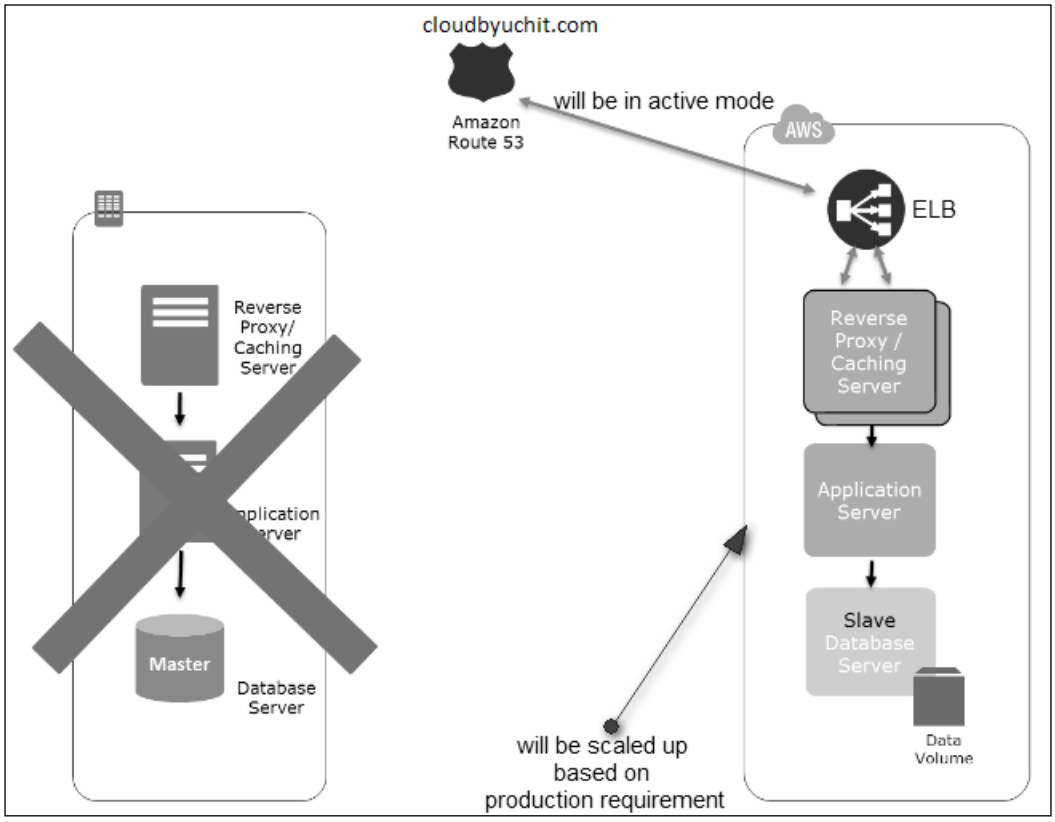
You can say that a Multisite kind of solution is an "Active-Active" approach configuration, as it goes in AWS, as well as on present in-house infrastructure; that is, both the infrastructures will be up and running and both will share the load. The data replication method will be determined by the retrieval point you pick. There are dissimilar replications methods used by AWS.

A weighted DNS service, such as Amazon Route 53, can be expected to route your traffic to diverse sites. By doing this, your ratio of traffic will go to AWS and the residue will go to your on-site infrastructure, which can be primary or subordinate, because Route 53 also provides latency and failover routing policies. In a disaster situation, you can adjust the DNS weighting and send all traffic to the AWS servers or vice versa.

- **Preparation phase**: In this phase, the AWS DNS service named Route 53 will route the some portion of the traffic to the AWS site, and other traffic will be on your on-premise infrastructure.

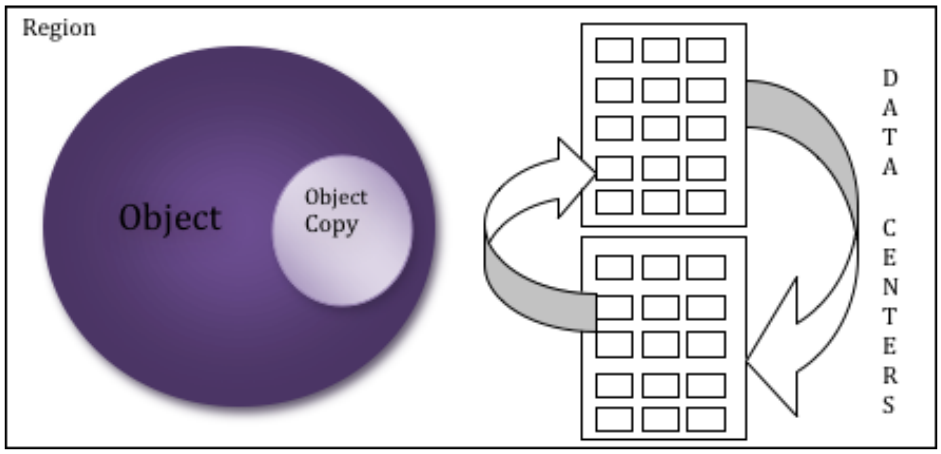


- **Recovery phase**: In case of breakdown, the DNS service will be slashed to route the traffic to catastrophe zone and route all traffic to the available infrastructure.



Replication of data

When you are replicating data to an isolated locality or let's say in diverse region, there are a few key concerns:



- **Distance between the application sites**: Large distances are characteristically subject to more latency or jitter.
- **Bandwidth availability**: How broad and variable are the interconnections links?
- **Data rate required by the web application**: In real time consequences, data rate should be always lower than the available bandwidth of network.
- **Replication**: The replication function should be working in parallel with each component.

There are two elementary approaches when you are replicating the data. They are as follows:

- **Synchronous replication**: Data is atomically simplified in multiple positions (multiple data centers which will be different by geographical locations), as shown in preceding diagram. Synchronous replication impacts will be huge, like a craving on network performance and availability.
- **Asynchronous replication**: Data will not be updated in multiple locations automatically. It will be relocated as your Cloud provider's network performance, available bandwidth permits it, and your web application will continue updating or putting data that may not be fully replicated until that time.

Several database systems endure asynchronous data replication where particular database systems do not. The database replica can be discovered tenuously, and the replica does not have to be utterly in sync with the primary database instance. This can be good enough in many real-time circumstances, for example, as a backup of resource data or reporting/read-only use cases. In AWS distributed infrastructure, AZs inside an region are well linked with each other, but physically apart. For example, when you are deploying your database on AWS infrastructure using the Multi-AZ mode, the Amazon **Relational Database Service** (**RDS**) will practice synchronous replication to duplicate data in a second accessible that is any available AZ. This will undertake that data will not be lost if the prime AZ becomes inaccessible due to a disaster.

Furthermore, AWS regions are absolutely independent of each other, and there is no technical or functional discrepancy in the way you approach them and use them. This will enable customers to form disaster recovery practices that extend overseas, minus the clashes or costs that this would usually incur. Clients can back up their data and systems (virtual machines) to two or more AWS regions, allowing service restitution even in the face of large-scale disasters. Clients can use all AWS regions to serve their customers over the globe with reasonably low complexity in their operational progressions.

Architecting with Amazon VPC

Amazon VPC lets you envision a logically isolated sector of the AWS Cloud, where you can spin AWS resources in a virtual network that you express. Nowadays, users adopt default VPC in which all the instances can be launched without the creation of new VPC. Essentially, Amazon VPC is an arrangement of the following components:

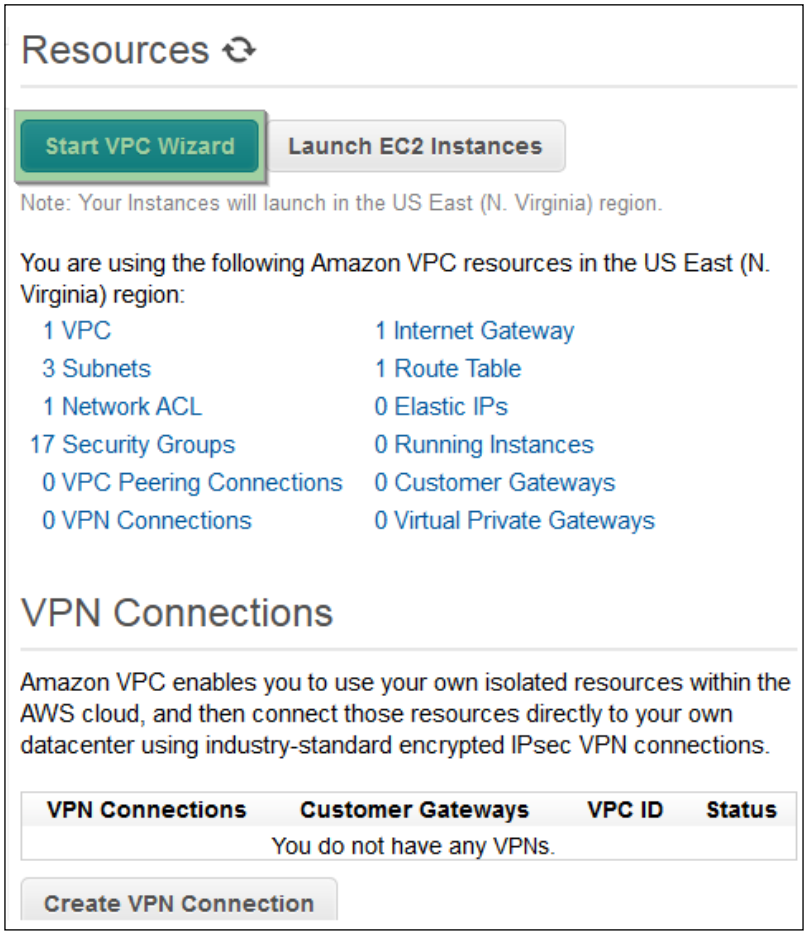
- A private, isolated segment on the AWS Cloud where you can launch AWS resources on a virtual network for which you gain full control is *Dedicated Instances*; here, no other person's data can be hosted
- A virtual network topology that resembles the established network in your data center
- Complete control, incorporating your own IP address range, subnets, route tables, **Access Control List** (**ACL**), and network gateways

The overall VPC is envisioned to include several basic features. It extends up to two AZs, so that you can dispense your applications across these AZs to architect your application's durability and high availability based on the traffic and number of users. The recommended approach is to use at least three AZs for a proper data replication in a cluster. With two AZs, if a single AZ is down, it can take the majority of a cluster down as well. Three AZs can survive a failure of a single AZ, which is not the case with two AZs.

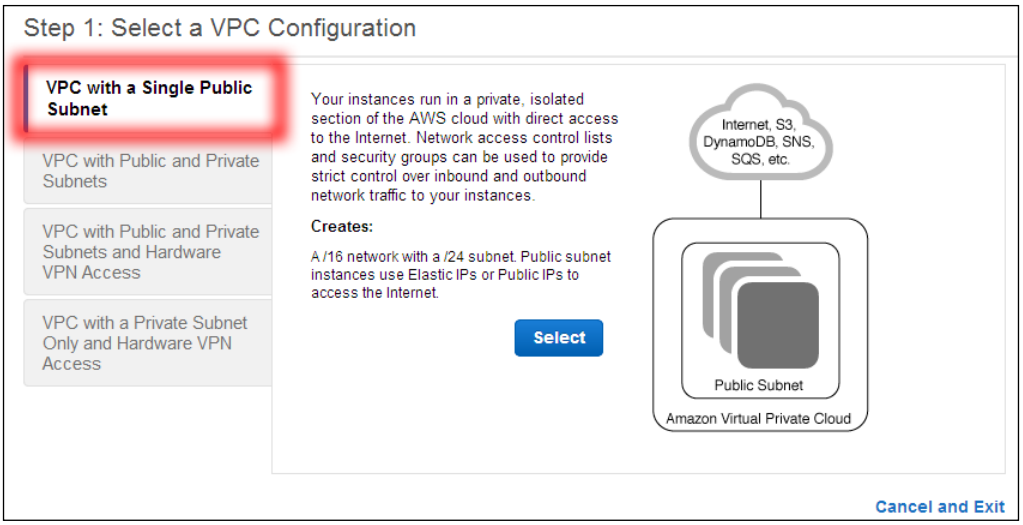
Within every AZ, there are two subnets. The public subnets can route unswervingly to the Internet. The private subnets will be able to communicate with any other subnet surrounded by the VPC or **Network Address Translation** (**NAT**) instances, but there will be no direct contact between private subnets and the public internet. AWS recommends using a NAT instance to allow the instances in the private subnets to connect to the Internet.

Furthermore, Amazon VPC provisions both hardware and software VPN tunnels at the data center level, and also client-based tunnels via software correspondences such as OpenVPN. VPC permits control over many tiers of security, and at the same time, it's easy to construct. To start with VPC, perform the following steps from the AWS Management Console:

1. Choose the **VPC** service and click on **Start VPC Wizard**.

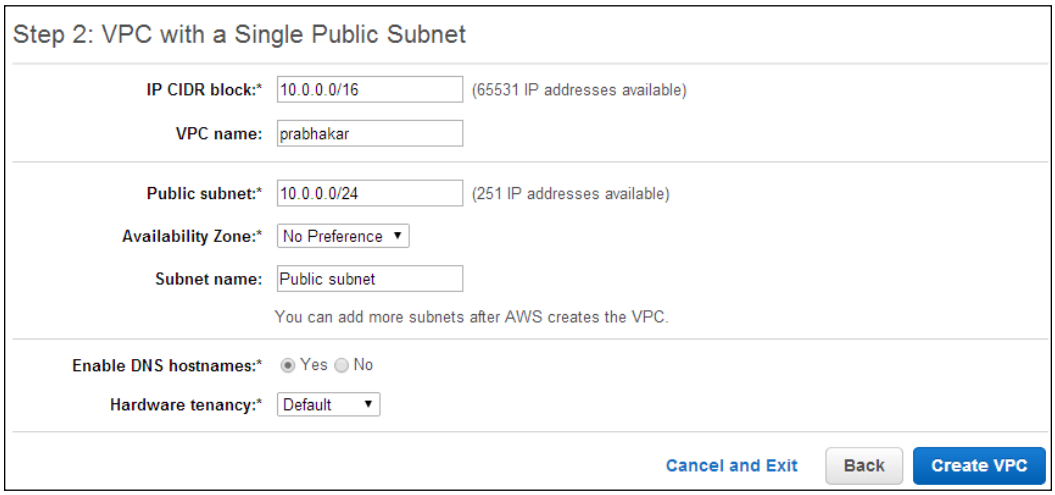


2. For this exercise, you'll produce a VPC with a single, public-facing subnet and then manually insert specific rules that automatically set up the second wizard. Choose the initial wizard substitute as **VPC with a Single Public Subnet**, and then click on **Select**.

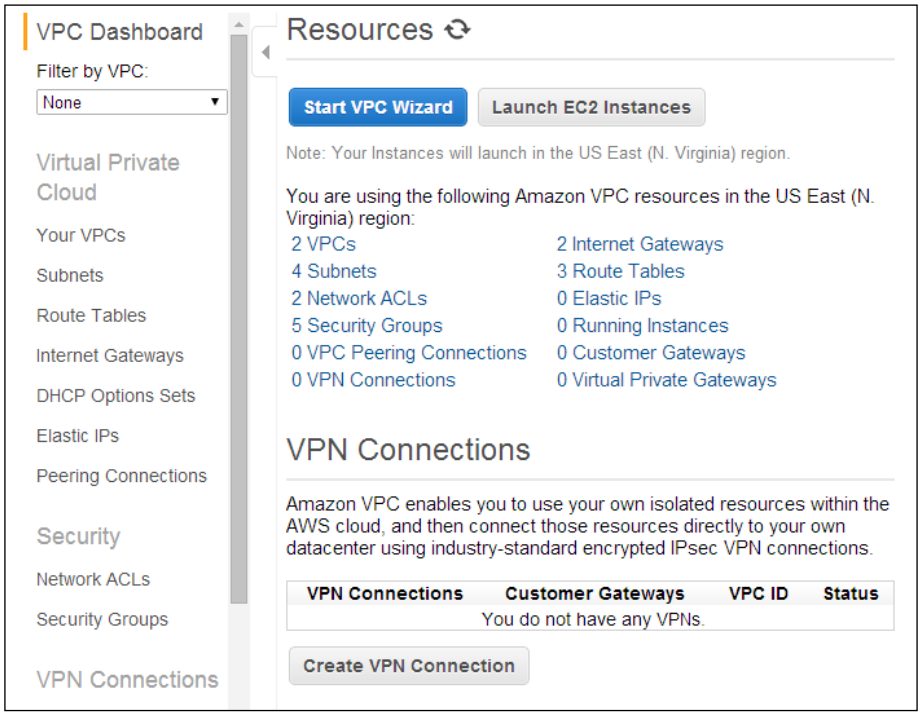


3. In the **VPC name** textbox, provide the name of your choice and enter `10.0.0.0/16` as the IP range in the **IP CIDR block** box. This value practices the **Classless Inter-Domain Routing** (**CIDR**) standards so that your VPC will have a private IP address space ranging from 10.0.0.0 to 10.0.255.255. In the **Hardware tenancy** drop-down list, confirm that **Default** is selected. This will run your VPC on common hardware. Finally, select **Yes** from **Enable DNS hostnames** and click on the **Create** button.
4. Review your configurations before finishing. The public subnet will have a default IP address range of 10.0.0.0/16, which is not routable on the public Internet, and the wizard will set up an Internet gateway with a public IP address. Nevertheless, the private subnet won't communicate with the Internet until you overtly permit it to do so. If you check that alternative, then AWS assures that no other account will be located on the same hardware and you will be the only one who will own that resource. But remember, this alternative will cost a flat USD 10/hour per region, so don't choose it unless you actually want it!

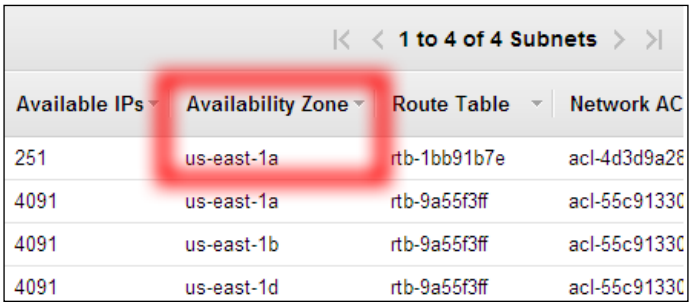
5. Finally, click on **Create VPC**.



6. It may take some time for the wizard to finish the job. Once the wizard has done its job, you will see the following screen:



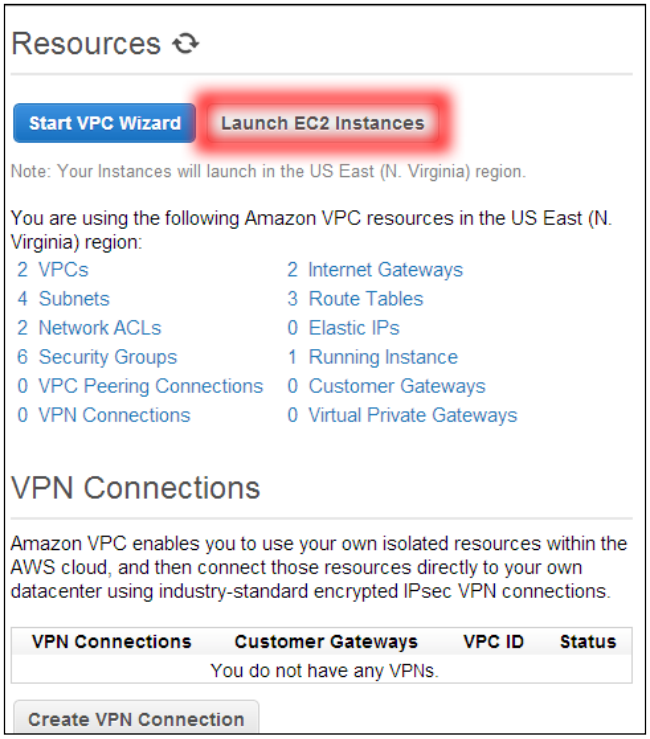
7. Click on the **Subnets** link and note the **Availability Zone** option chosen by the wizard. Note it as you will need it in subsequent stages.



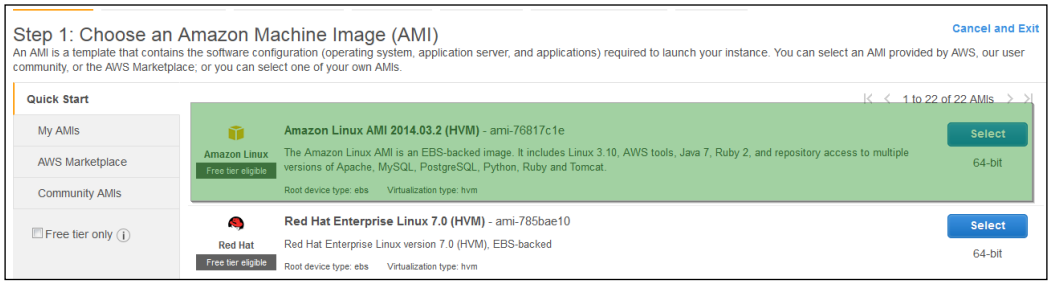
Launching an instance in the VPC

Now, let's see how we can launch an instance in VPC:

1. Navigate to the VPC dashboard and click on the **Launch EC2 Instances** button. This will move you to the **AWS EC2** tab in the console. Once there, just click on **Launch EC2 Instances**.

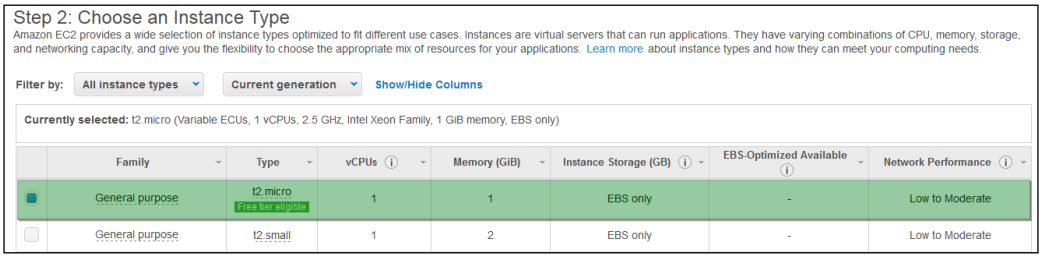


2. Pick the initial preference **Basic 64-bit Amazon Linux AMI**. The genuine AMI ID in the wizard may be different from the one in the screenshot, as the AMI is modernized regularly by AWS.



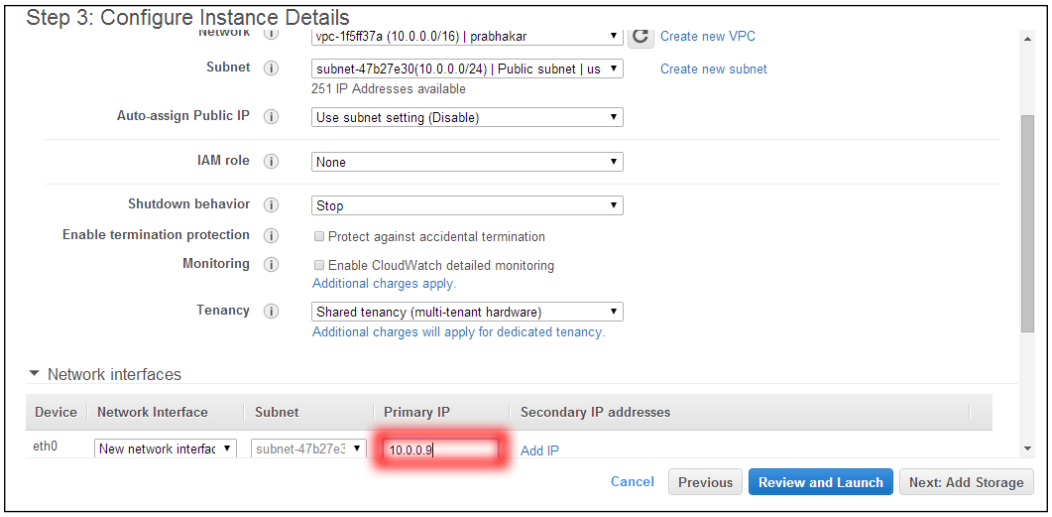
AMI provided by AWS

3. You are going to launch an instance into the VPC from which you can learn about the working of VPC. Alter the instance type to **t2.small** or **t2.micro**, or choose from the several additional instance types provided in the Amazon VPC.



Instance types

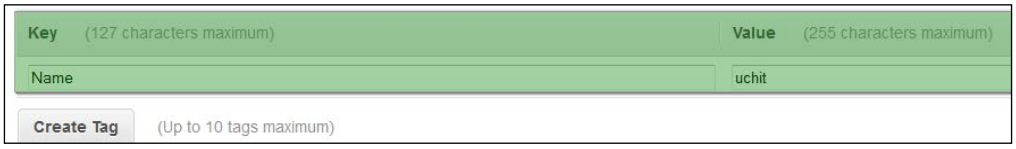
4. In the **Configure Instance Details** page, remember that the AZ will be auto picked by the subnet, and there is only one special in the drop-down list because you've only formed a single public-facing subnet. Provide `10.0.0.9` as the IP address.



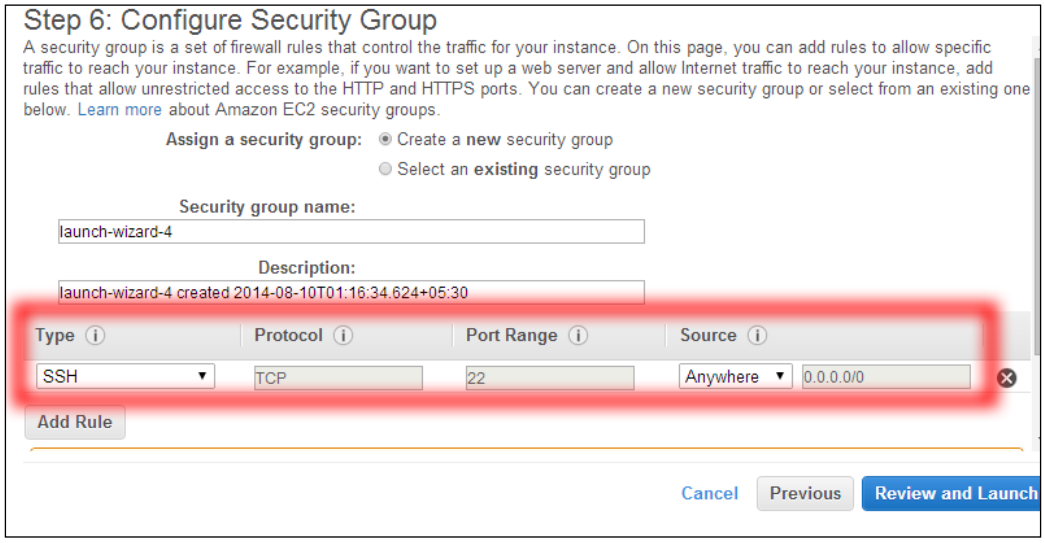
The preceding screen presents two essential features that are exclusive to VPC, at least when compared to public EC2 instances. The first one is the static IP address that you just inserted. In the public-facing instance, you can appoint Elastic IP addresses; nevertheless, those addresses are either public-facing only (but here, this address is a private, non-routable address from Internet), or Elastic IPs are enclosed after the instance takes off (the preliminary address on a public instance will be appointed via DHCP).

In addition, this static address factors the way that you presume a static address to work as typical EC2 instances.

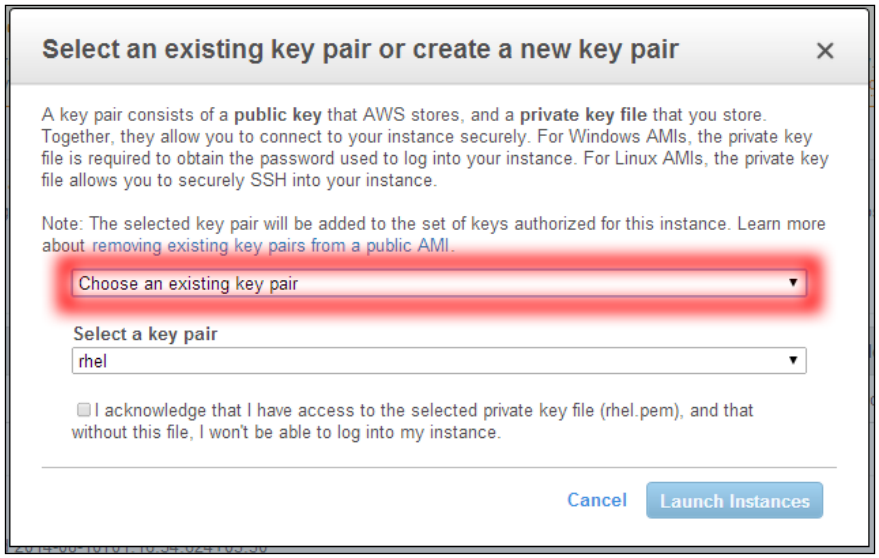
1. Note, that we are allotting the address "from the outside" of the VPC network. If you were to login to the server and assign the network address at the OS level, AWS can appoint the address as in the illustration to `10.0.0.9`. At last, click on **Next** to navigate to the next phase.



2. VPC security groups differ from "public" security groups. Among other things, these security groups control both inbound and outbound traffic as standard EC2 security groups. "Public" groups barely limit the inbound traffic. So, let's fashion a new security group that will be the default optimal on given screen. Add a rule for SSH if not present and also for All ICMP protocols. Don't neglect to click on **Add Rule** to employ your alterations and open whichever port you wish to unlock. Then, click on **Review and Launch**.



3. If you already have a key pair, practice it. If you don't have it, you have already studied how to generate and practice it.



4. Evaluate the surroundings and spin the instance by clicking on the **Launch** button.
5. Next, ensure one instance is running in the VPC environment that has an IP address `10.0.0.9`. In order to approach that instance over the public Internet, you have to give that instance a public IP address because the specified IP address is private only. To allocate a public IP address in VPC, you must to assign and subordinate an Elastic IP Address for VPC from Elastic IP wizard.

Navigate to the Elastic IP wizard on the left corner in EC2 Management Console and click on the **Allocate New Address** button. Then, select **VPC**, click on **Yes**, and click on **Allocate**. Subordinate the IP address to the VPC instance that you just created by clicking on the **Assign Address** button and picking the instance you started (it will present you the all instances which will be accessible in your EC2). You must select either the instance or the network interface. Finish the job by selecting **Yes** and clicking on the **Associate** button. If you don't see the Elastic IP that you just assigned from Elastic IP wizard, crosscheck the **Viewing** filter and change it to **ALL** or **VPC Addresses**.

Creating a private subnet

You have successfully created the public subnet. So, for the next stage you have to create a private subnet that will be communicated to public subnets via a NAT instance. To start with a private subnet creation, perform the following steps:

1. Let's create a private subnet, let's say a "10.0.1.0" subnet that will not straightforwardly connected to the public Internet. Check the first digit in the third octet of the IP address, which will be other than 0 for the public subnet. Going back in the **VPC** tab, click on **Subnets** and select **Create Subnet**.
2. Provide `10.0.1.0/24` as the IP address. Technically speaking, the AZ could be different from formerly created one; nevertheless, using multiple AZs doesn't mark as an irrelevant deal when you are modestly dividing "public" from "private". In a fully-developed production environment, you will be tending to a total of four subnets (two sets of public and private pairs) dispensed throughout manifold AZ.
3. Now, it's time to launch instances into public or private subnets, as preferred. Insert public facing instances, for example web servers, into the public-facing subnet `10.0.0.0/24`. Insert backend instances that should not be accessible from the Internet directly, for example, DB servers, into the private subnet `10.0.1.0/24`.

Spinning a database instance in the private subnet

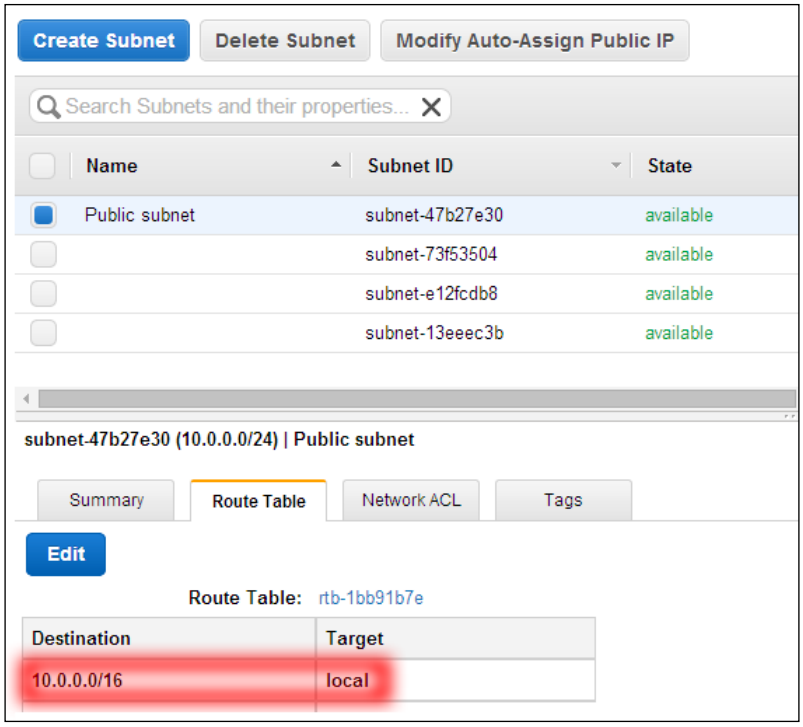
Liftoff added Amazon Linux server. However, this time spin an instance with the private subnet (10.0.1.0/24) and provide it an IP address of `10.0.1.10`. Define the server like the backend DB Server and allocate the same key pair to it as the publicly created instance. You can fashion a new **Security Group**, but open up port 22 for SSH and all ICMP ports identical you did previously in the public subnet.

The instance you just launched in the private subnet only has a private IP address. So, the question is, how can you connect to the server via SSH from the Internet in order to tenuously administer it? (I'm absolutely ignoring the statistic that in a tangible production use case, you shouldn't link straightforwardly from the outer world Internet). Three components can enable RDP connections in a public subnet:

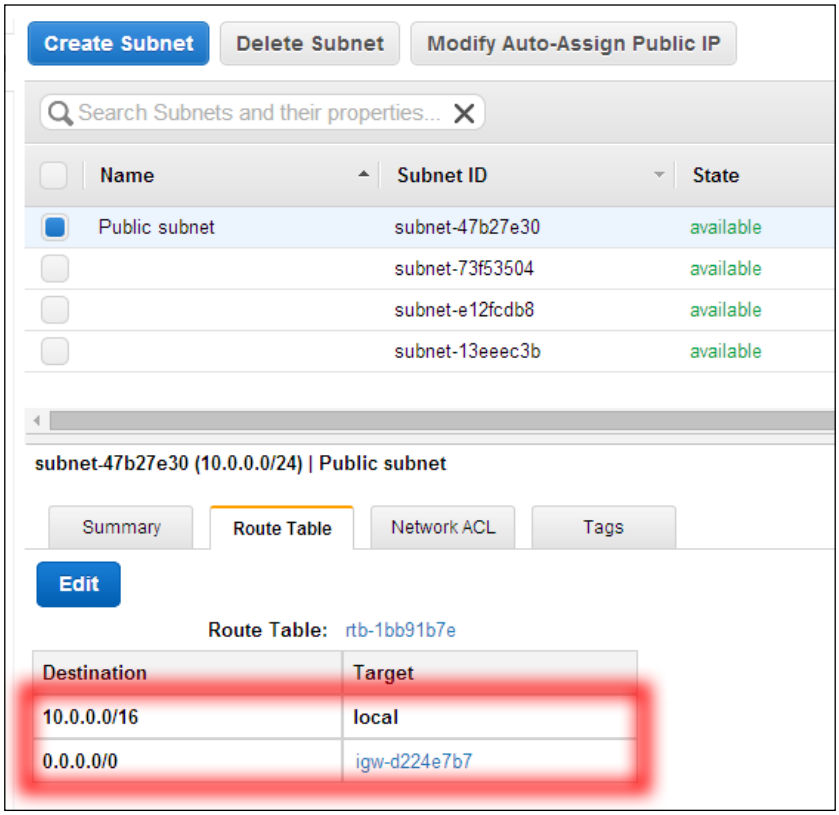
Port forwarding from an instance:

- An SSH gateway instance
- A VPN tunnel

A naive swift question, can't you just allocate an Elastic IP address to the backend DB server and access it from the Internet? Even though it seems to be a logical solution, it won't work. You can try to allot an address to verify this point yourself. The purpose is that the route table connected with this subnet does not permit traffic to or from the Internet.



You can fill details in the preceding screen to get the succeeding one, which is for the public subnet. Look at that additional line that indicates the Internet gateway as a considerable route:



You could switch route tables, at which point the Elastic IP address would start to acknowledge Internet traffic. Of course, we don't need to do that, specifically since this subnet is private one. This subnet is so quarantined that servers on it can't even extend to the Internet. This server would certainly not be proficient to download an update from the Internet, as it will incapable to correspond with any Internet-based download servers. For the most part, that's closely what you desire.

Consequently, you can determine a NAT instance in the public subnet and route outbound traffic over it via a special route table. Indeed, that's what the wizard does if you select the second option (both a public and private subnet).

Creating a Remote Access Software VPN to your VPC

You can practice with any VPN software that appears recognizable to you, such as Windows Server 2003/2008 incorporated clarifications, OpenVPN, or OpenSwan. As per my understanding for connection, you can practice OpenVPN, which is permitted for two parallel VPN connections.

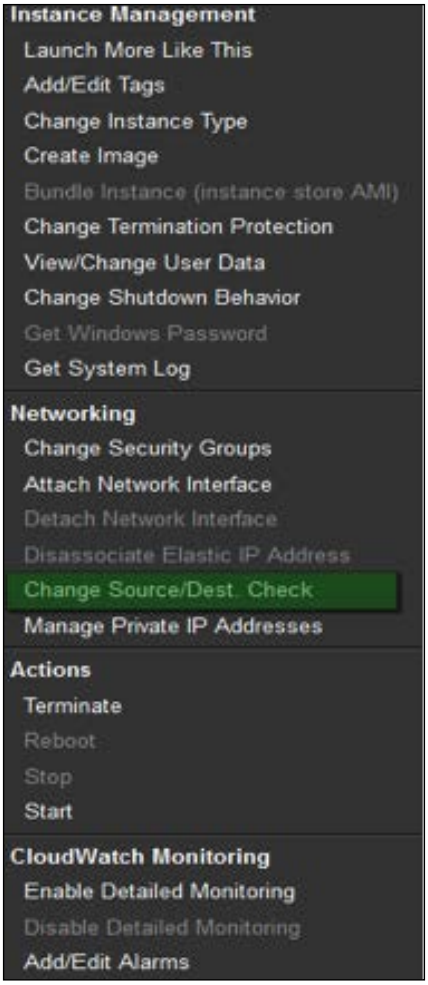
Launching an OpenVPN instance

You're working to construct an OpenVPN employment from an AMI image for now, which can be done as follows:

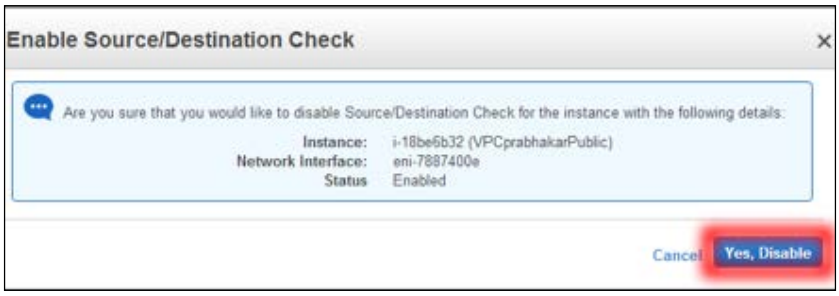
1. Start with the cheapest **N. Virginia** region, click on the **AMIs** tab in the **EC2** tab. Choose **Amazon AMIs** and then search for **openvpn**. In this screenshot, you will see an image for version 2014.03.2 with `ami-id ami-76817c1e`. Nevertheless, this might not currently be the newest version, so pick accordingly by name. Be assured that you are working to lift off this AMI into the VPC by the public subnet.
2. Use `10.0.0.99` as the static IP address and label it like the OpenVPN server. Confirm that you have configured a new **Security Group** with the following rules:

ICMP		
Port (Service)	**Source**	**Action**
ALL	0.0.0.0/0	Delete
TCP		
Port (Service)	**Source**	**Action**
22 (SSH)	0.0.0.0/0	Delete
443 (HTTPS)	0.0.0.0/0	Delete
943	0.0.0.0/0	Delete
946	0.0.0.0/0	Delete
UDP		
Port (Service)	**Source**	**Action**
1194	0.0.0.0/0	Delete

3. Acknowledge the settings and spin the instance. Voila! You are approximately done by 70 percent. Disable source/destination checking on the VPN instance. Each EC2 instance performs source/destination checks by default. This means that the instance must be the source or destination of any traffic it sends or receives. However, a NAT instance must be able to send and receive traffic when the source or destination is not itself. Therefore, you must disable source/destination checks on the NAT instance. To do this, first right-click the OpenVPN instance and choose the **Change Source/Dest. Check** selection, as shown here:



4. Then, click on the **Yes, Disable** button to disable the check for the instance:



The Source/Destination Check popup

5. Now, assign a new Elastic IP address to the OpenVPN server. Verify that you are able to ping the OpenVPN instance and SSH into the OpenVPN server using SSH client. For this instance, you should log in as the `root` user. OpenVPN will query you tons of subjects for authorization and all. Agree to the terms and then select the defaults for the rest. Once it's finished, you want to design a password for the `openvpn` user. To do so, type the following command on the CLI:

```
PROMPT> passwd openvpn
```

6. Then, enter a password; it won't display the characters as it's a default Linux behavior. It will ask one more time to confirm that you are typing the accurate one, as shown in the following screenshot. Retype and memorize the password for further use.

Password confirmation

Downloading the OpenVPN client

You can download the OpenVPN client by performing the following steps:

1. Provide the Elastic IP address for the VPN server in the browser (acknowledge security exclusions if you get any). You should now see the login window. Log in as the `openvpn` user and enter the password.

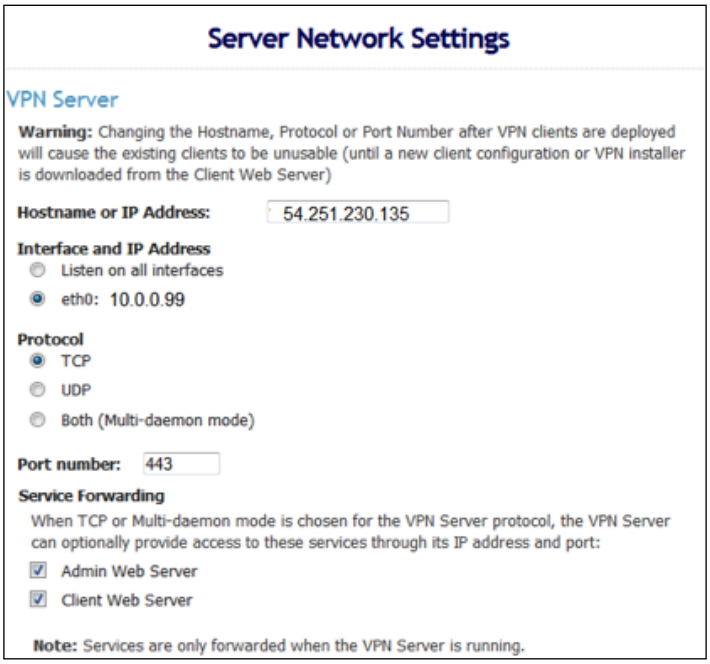
2. Now, download and ordain the client relevant to your platform.

Configuring the OpenVPN server

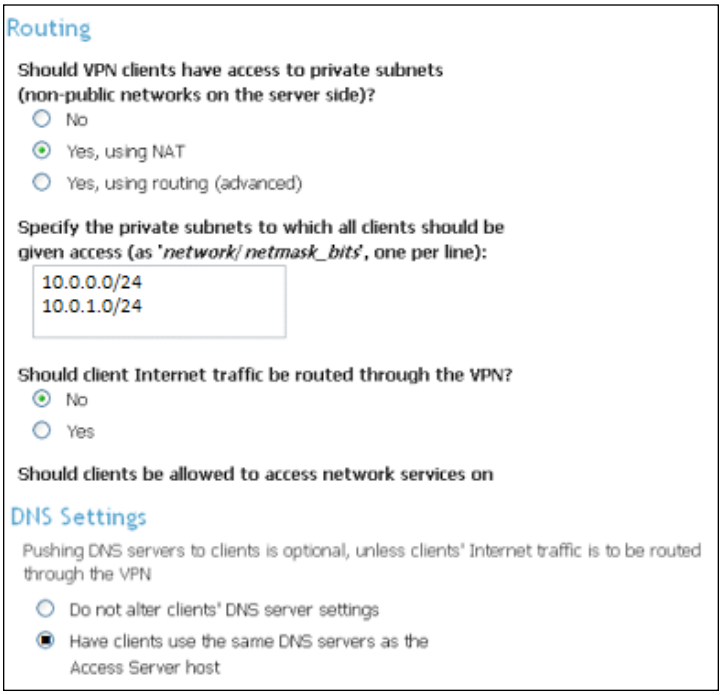
To configure and connect OpenVPN server, follow these steps:

1. Connect to the OpenVPN server administrator boundary using a browser. The address will be `http://<Elastic IP>/admin`.

2. Acknowledge the terms and conditions and click on **Agree**. Click on **Server Network Settings** and provide your server's public IP address, as shown in the following screenshot:



3. Permit access to the private subnet in the VPC. Connect on **VPN Settings** to insert the private subnet under the **Routing** division, which will be `10.0.1.0/24` in your illustration:



You can associate to it by SSH if you require. If the page is visible on a browser, it means you have done this successfully! You have effectively created a VPC environment.

Summary

In this chapter, you started with the vital steps and logic that should be implemented as a disaster recovery plan. Then, we discussed data replication and proposed AWS disaster recovery configuration. Finally, we saw how to create and configure a VPC within a AWS public Cloud infrastructure.

In the next chapter, you will come to know how to setup SDKs and IDE toolkits that can be used during programming with different AWS services.

6
Tools for AWS and Setup Guidelines

There are several interfaces available to make user interaction with AWS easy and effective. To start with tools for AWS, you need to understand what these tools are and how they work with AWS. Java is widely used with AWS and the most important factor is that it is free to download and install. There are SDKs and libraries for Ruby, Python, and PHP that you can find from following URLs:

- `http://aws.amazon.com/sdk-for-ruby/`
- `http://aws.amazon.com/sdk-for-python/`
- `http://aws.amazon.com/sdk-for-php/`

There are four interfaces that are commonly used to interact with AWS:

- AWS Management Console
- Eclipse plugin
- Command-line interface
- API

We will cover the following topics in this chapter:

- AWS SDK
- AWS Toolkit for Eclipse
- Configuring the AWS Toolkit

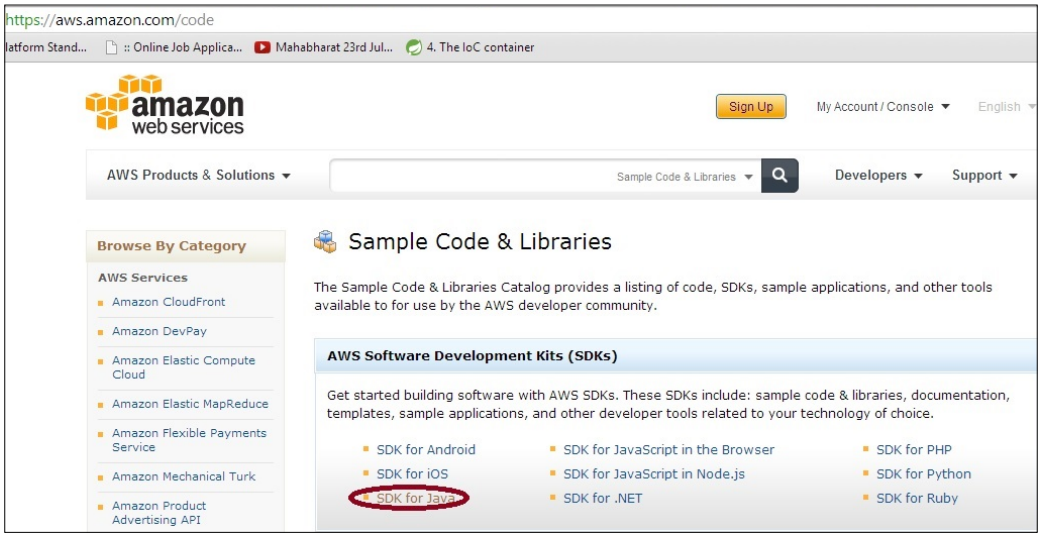
Working with AWS SDKs and IDE toolkits

A **Software Development Kit** (**SDK**) or "devkit" is typically a set of software development tools that permit the development of applications for a convinced software package, software context, hardware stage, computer classification, video game console, OS, or parallel development platform.

We can use APIs to bridge the gap between hardware and applications. Communal tools incorporate debugging services and other utilities that are repeatedly used in an **Integrated Development Environment** (**IDE**). Also, SDKs commonly involve mockup code and associate methodical notes or supplementary documentation to refine points made by the main orientation material.

SDKs may have enclosed licenses that make them unsuitable to develop software projected to be advanced under a mismatched license. For design, a proprietary SDK will probably be inappropriate with free software development, whereas a GPL-licensed SDK can be incompatible with exclusive software development. LGPL SDKs are characteristically user-friendly and comprehensive for privately-owned development.

As shown in the following screenshot, you will be able to realize the AWS SDK for Java on the AWS website. If you click on it, it'll be downloaded on your system.



Let's understand each folder and the internal files:

- **Documentation**: This folder contains all the content needed for reference. It has the syntax, package, structure, and description of each class/method. Using this documentation, we can understand the underlying class/method that we can use in our own library.
- **Lib folder**: This folder contains the `.jar` files that are necessary to start the development of AWS using your Java code. It contains other `.jar` files like the following:
    - `aws-java-sdk-<version>.jar`: This contains all the classes required for AWS development, for example, class for AWS authentication and so on. Generally, people use dependency management tools, such as Gradle, Maven, or Ivy, to fetch the SDK jars. It is commonly used to execute all command-level operations from the command prompt. It can only be executed once they are configured in environment variables.
    - `aws-java-sdk-<version>-sources.jar`: While creating code, if you want to attach source files for reference, you can only do that using this JAR.

  There is one `javadoc.jar` file that stores the documentation for all AWS classes. Since it's optional, it's upon developers whether they need to keep them for reference.

- **Samples**: This folder contains the samples programs for quick understanding of code and its nature. In general, it's not easy to adapt new classes without understanding their basic flow. These codes will give you a hands-on exercise. The basic examples are as follows:
    - `Amazon-DynamoDB`
    - `Amazon-EC2SpotInstances-GettingStarted`
    - `Amazon-EC2SpotInstances-Advance`
    - `Amazon-Kinesis`
    - `Amazon-Kinesis-Application`
    - `Amazon-s3`

    - `AmazonS3TransferProgress`
    - `AmazonSimpleEmailService`
    - `AmazonSimpleQueueService`
    - `AwsCloudFormation`
    - `AwsConsoleApp`
    - `AwsFlowFramework`

- **Third-party**: This folder contains the third-party APIs that can be utilized in code structuring. Some third-party APIs are as follows:
    - aspectj-1.6
    - commons-codec-1.3
    - commons-logging-1.1.1
    - freemarker-2.3.18
    - httpcomponents-client-4.2.3
    - jackson-annotations-2.1
    - jackson-core-2.1
    - jackson-databind-2.1
    - java-mail-1.4.3
    - joda-time-2.2
    - spring-3.0
    - stax-api-1.0.1
    - stax-ri-1.2.0

stax-api-1.0.1 and stax-ri-1.2.0 are not included in SDK v1.8.10.1

You can also add your own component, such as jQuery or others, for useful purposes and strong adaptability. This was all about the AWS SDK structure. While executing examples, you will see its usage and get an in-depth understanding of flow and its components.

- **The AWS Toolkit**: The AWS Toolkit for Eclipse is an open source plugin for the Eclipse Java IDE that makes it easier for developers and code integrators to develop, debug, integrate, migrate, and deploy Java-based applications that use the AWS resources platform. There are some extraordinary functions/features that make the Amazon platform the best option for developers:
    - **AWS Explorer**: This empowers you to correlate with numerous AWS services from the Eclipse IDE. The AWS Explorer provisions managed data services such as **Amazon Simple Storage Service**, **Amazon SimpleDB**, **Amazon Simple Notification Service**, and **Amazon Simple Queue Service**. AWS Explorer is further proposing to introduce **Amazon Elastic Compute Cloud** management and deployment functionality to the AWS Elastic Beanstalk via SDK or API. AWS Explorer provides various AWS accounts; you can undoubtedly transform the capitals presented in AWS Explorer from one account to alternative. AWS Explorer also empowers you with additional functionality such as the capability to design and accomplish key pairs and security groups.

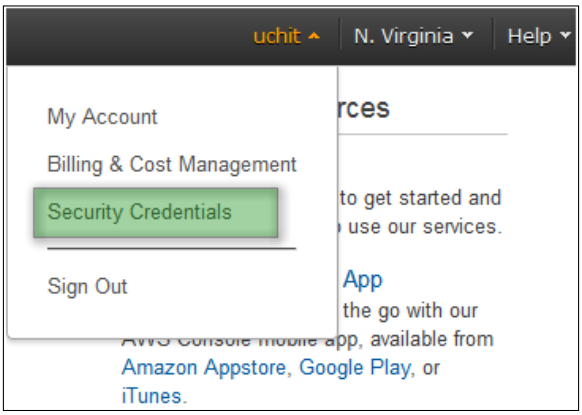
The AWS Toolkit for Eclipse will install the newest version of the AWS SDK for the platform you have designated. From Eclipse, you can directly manage, modify, construct, and deploy any of the illustrations encompassed in the SDK packages. You can download and install it using the following steps:

1. Go to URL `aws-amazon.com/sdk-for-java`.
2. As shown in the following screenshot, click on **AWS Toolkit for Eclipse**:

3. Another way to configure this is to download Eclipse Juno/Luna from the link `https://www.eclipse.org/downloads/`.
4. Start Eclipse, go to **Help**, and then click on **Install New Software**.
5. In the **Work with** box, type `http://aws.amazon.com/eclipse`, and then press the *Enter* key.
6. In the list that appears, expand the **AWS Toolkit for Eclipse** option.
7. Select **AWS Toolkit for Eclipse** to download it.
8. Click on **Next** and the Eclipse wizard will take you through the other installation procedures by default.

To access AWS through the Eclipse Toolkit, you have to configure the Eclipse Toolkit with your access key ID and secret access key that should be available in your AWS account. In addition to allowing the Eclipse Toolkit to admit your account, your access keys are used to allow web services-based requirements to AWS. Allowing web services requests ensures that only approved programs can make such requests. Moreover, by associating access keys with each web services request, AWS will be able to track service usage for billing and monitoring.

The keys will have a combination of an access key ID and secret access key, which will be used to sign programmatic logical request that you will compose from application source code to AWS for accessing resources. If you don't have access keys, you can create the keys from the **AWS Management Console**. To do so, go to **Security Credentials** and select the **Access Key ID** option, as shown in the following screenshot:

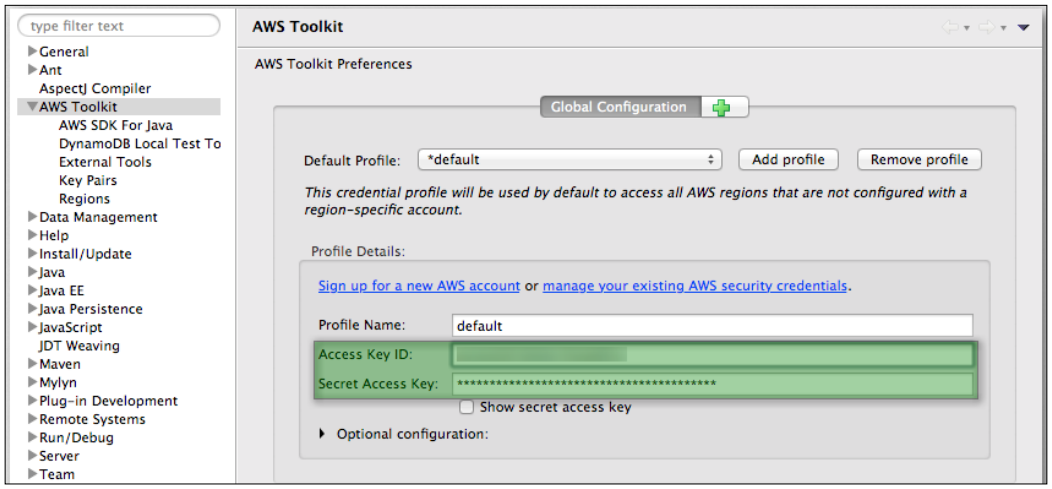


A better way would be creating an IAM user or group with limited privileges (only those needed by an app developed) and use their access keys instead. This will also allow us to later mark the access keys as "inactive" or delete them when they are not needed any more. Using an account's root credentials for app development provides it with access to all AWS services and these credentials can't be easily revoked later. Keep the credentials confidential in order to guard your account, and never e-mail it. Do not share it with a third person from your organization, even if investigations come from AWS or from any other channel.

To add your access keys in the Eclipse Toolkit, follow the procedure given here:

1. Open the Eclipse **AWS Toolkit Preferences** dialog box and click on **AWS Toolkit** located in the sidebar.
2. Type your access key ID in the **Access Key ID** box.
3. Type your secret access key in the **Secret Access Key** box.
4. Click on **Apply** or **OK** to store your access key information.

Here is an example of a configured **AWS Toolkit Preferences** screen with the default account:



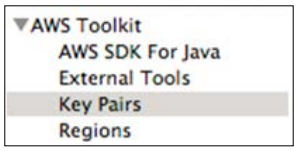
The preceding dialog box enables you to add access information for more than one AWS accounts by selecting the designated profile. Multiple accounts enable developers and administrators to split resources that will be used for the development stage from resources that are used in the production stage.

To add another set of access keys, perform the following steps:

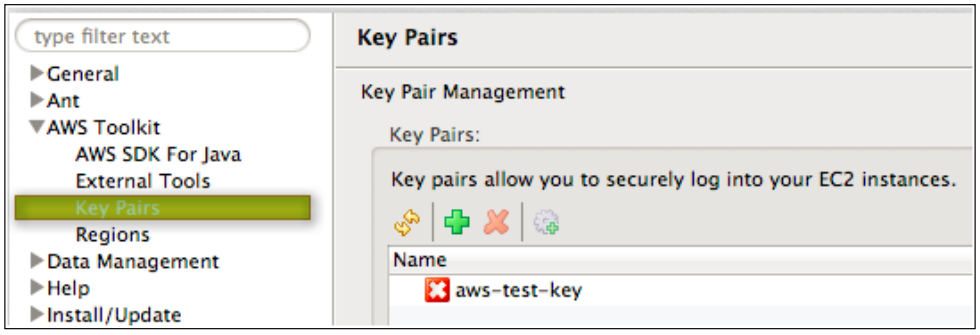
1. On the **AWS Toolkit Preferences** screen, go to the **Profile Details**, and click on **Add Account** button.
2. Add your account particulars to the **Account Details** section.
3. Pick a suitable name for **Account Name** and enter your access key information in the **Access Key ID** and **Secret Access Key** boxes.
4. Click on **Apply** or **OK**.
5. You can repeat the preceding steps as practice for as many sets of AWS account intelligence that you require.

The Eclipse Toolkit can also achieve Amazon EC2 key pairs from an AWS account. Nevertheless, you will have to associate private keys to use in the Eclipse Toolkit manually. To observe Amazon EC2 key pairs in the Eclipse Toolkit, follow the next steps:

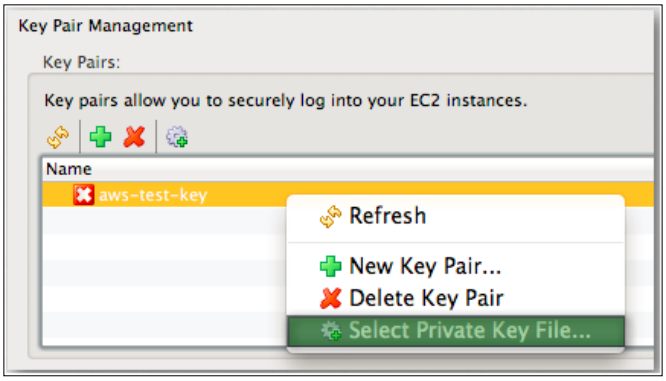
1. Open the Eclipse **Preferences** dialog box and click on **AWS Toolkit** in the sidebar to view the supplementary classifications of the Eclipse Toolkit surroundings and configure it.



2. Go to **Key Pairs**. Eclipse will display a list of available key pairs in that window. If a key pair has a red "X" sign next to it, you must create an association of a private key with the key pair to practice it with the present use case.



3. Right-click on the key pair and select **Select Private Key File** from the context menu. Go to the private key file and select it to subordinate it with your key pair, as shown here:

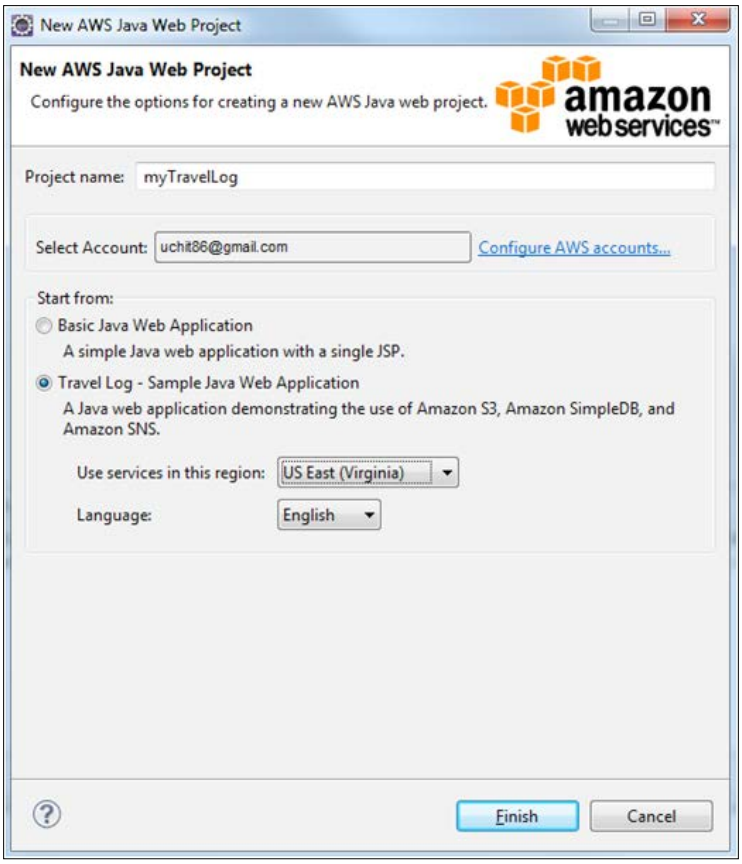


To employ the web application, you have to perform the following steps:

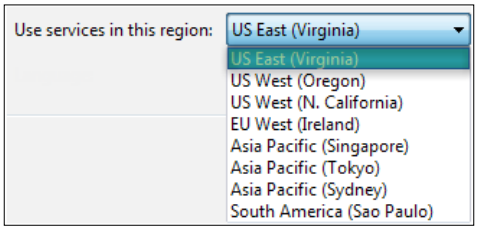
1. On the Eclipse toolbar, click on the **AWS** icon, and then select **New AWS Java Web Project**.
2. In the new **AWS Java Web Project** dialog box, in the **Start from:** option of the dialog box, set **Travel Log** as **Sample Java Web Application** and give a name `myTravelLog` in the **Project name** box.
3. Click on the **Finish** button. The toolkit will create the project and the project will be presented in **Project Explorer**.

If you aren't able to see **Project Explorer** in Eclipse, go to the **Window** menu in Eclipse, click on **Show View**, and choose **Project Explorer**.

Now you are ready to start.

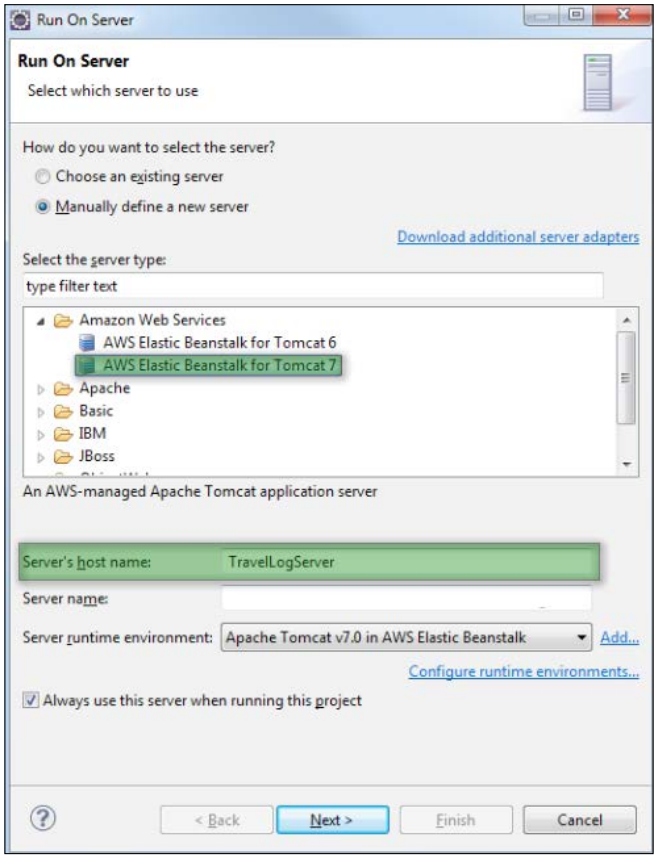


The **AWS Java Web Project** dialog box will allow you to choose the region in which your web application will be deployed to and run from. Here, I am going to select **US East (Virginia)**, as it is the cheapest one in my case.

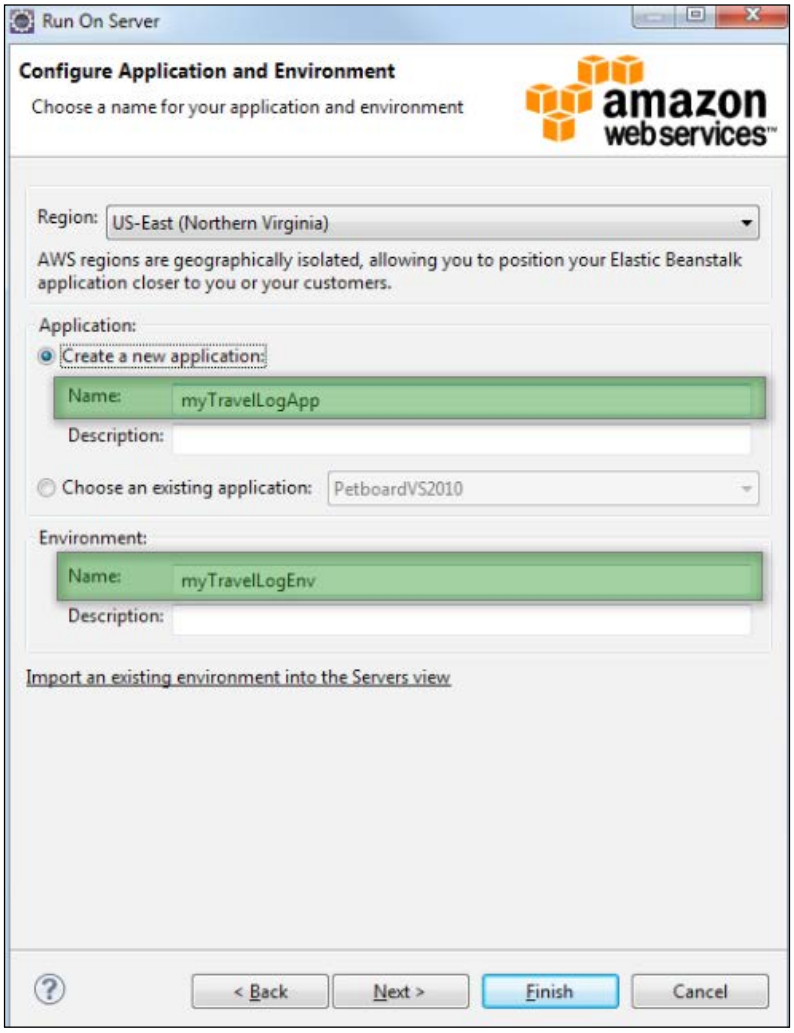


After selecting the proper Region, you have to proceed with the selection of container in the following manner:

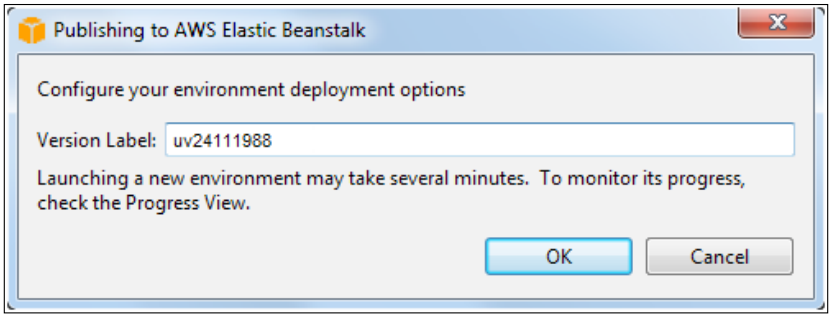
1. From **Project Explorer**, navigate to **myTravelLog | Run As | Run on Server**.
2. In the **Run on Server** dialog box, select **Manually define a new server** and click on **AWS Elastic Beanstalk for Tomcat 7** from the catalog of specified server choices.
3. Enter an appropriate name, such as `TravelLogServer`, into the server's hostname box or provide a specific name of your choice.
4. Finally, choose **Always use this server when running this project** and select **Next**.



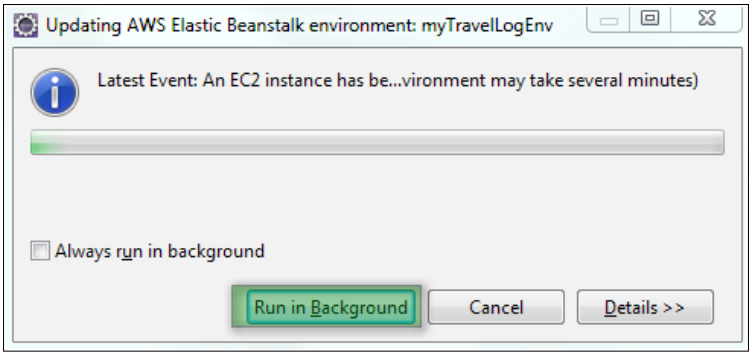
5. In the **Run On Server** box, supply an application name, such as `myTravelLogApp`, and an environment name, such as `myTravelLogEnv`. Then, click on **Next**. This is shown in the following screenshot:



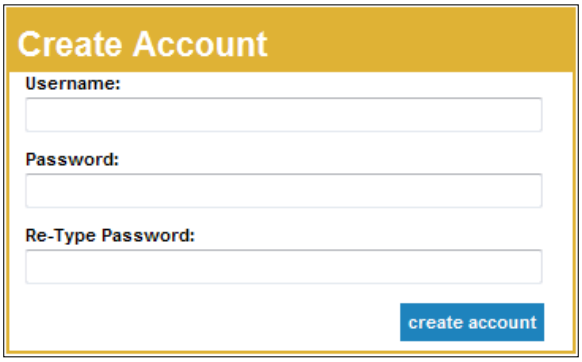
6. Go to **Run On Server | Advanced configuration**. The **Advanced configuration** dialog box allows you to state further parameters for your web application reference, such as IAM role, CNAME, Announcement email address, and so on.
7. Before deploying your application to AWS Beanstalk container, the toolkit will display a dialog box where you can fix a **Version Label** where you can specify the version number. The AWS Toolkit will produce a distinctive version label which is time stamped but still editable.
8. Click on **OK**.



9. While your application is deploying, the AWS Toolkit will display a progress status, as shown in the following screenshot:

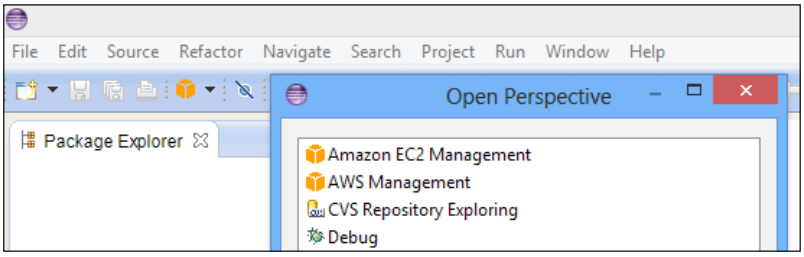


10. When the deployment is complete, you will be able to see the following screen. This is the user interface for the Travel Log application that will be running on your Amazon EC2 instance.

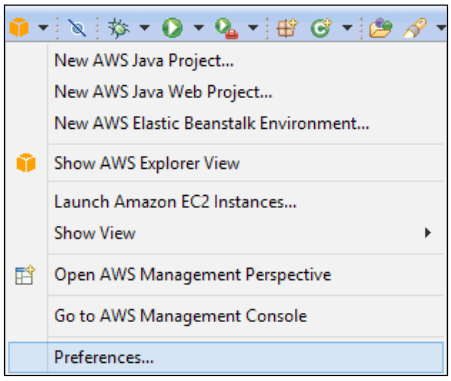


So, the preceding example was about Elastic Beanstalk deployment using AWS SDK Toolkit. Let's see one more example with the NoSQL database called DynamoDB from AWS to get more hands-on experience.

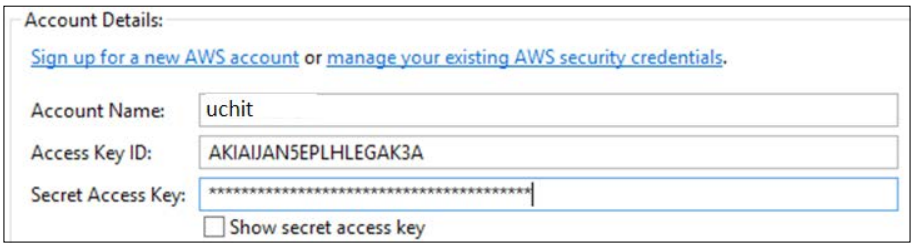
In Eclipse, you can see a new icon . This is the AWS Toolkit for Eclipse icon. The default Eclipse perspective will be Java; to open the perspective to work with DynamoDB, go to Window and select the **Open Perspective** option. It must show the first two perspectives, as shown here:



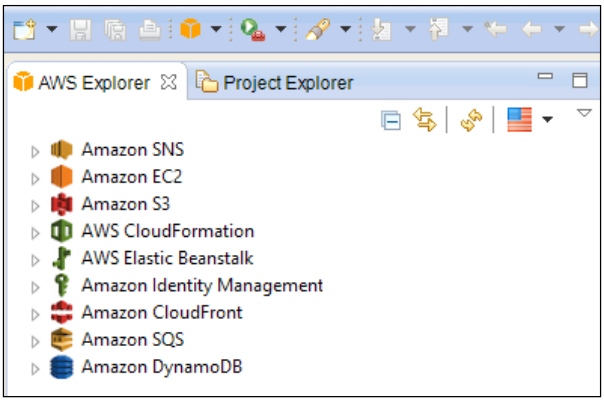
11. Double-click on **AWS Management** to start working. While working in the AWS Management Console, we must log in with our AWS username and password. But for the Eclipse plugin, we need to specify a few more attributes for authentication. We can specify our account details by clicking on the AWS Eclipse Toolkit icon and selecting the **Preferences** option.



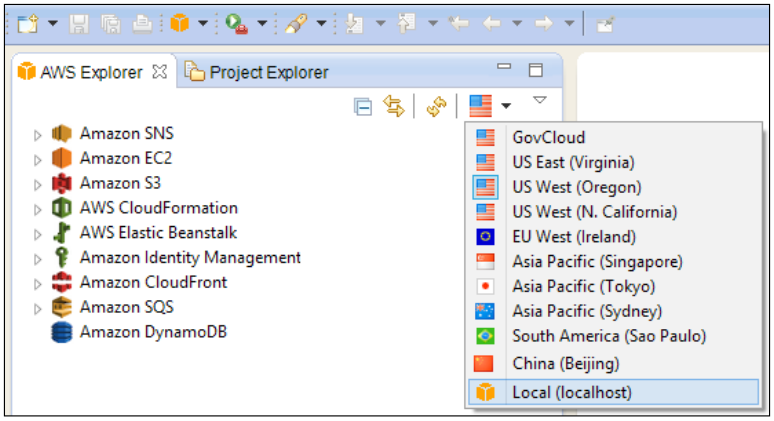
12. The pop-up window that appears will ask for the following details, which we can fetch from security page in the AWS Management Console.



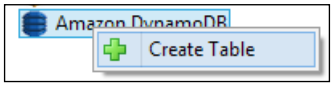
13. Once the configuration is successful, all the AWS components will be loaded. You can right-click on corresponding service and select refresh to get the latest data. The icon in the top-right (with US flag) is the region that you have currently selected. If the DynamoDB table is created in the US East (Virginia) region using AWS management console, and if you have selected the US West (Oregon) region here in Eclipse, then you will not be able to see that table in your perspective.



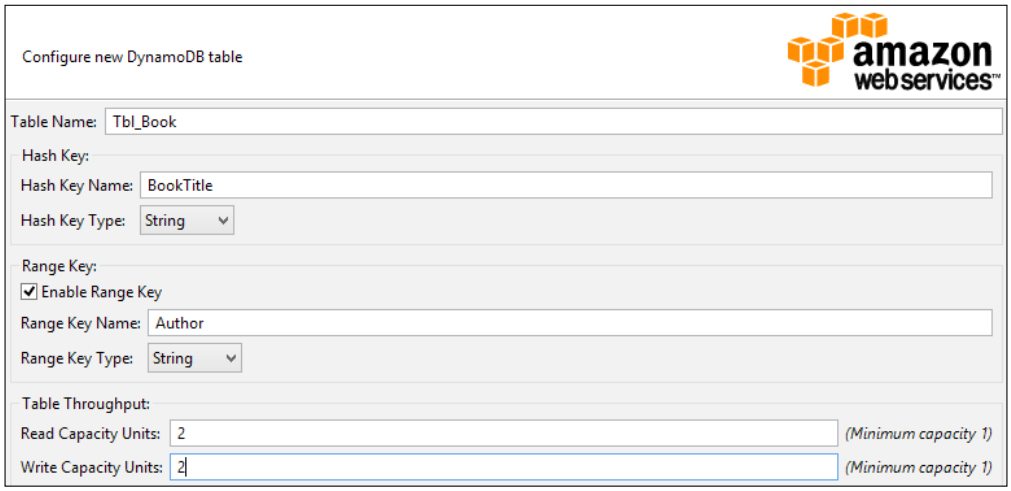
14. You can change the region with a single click on the drop-down with the flag on top-right.



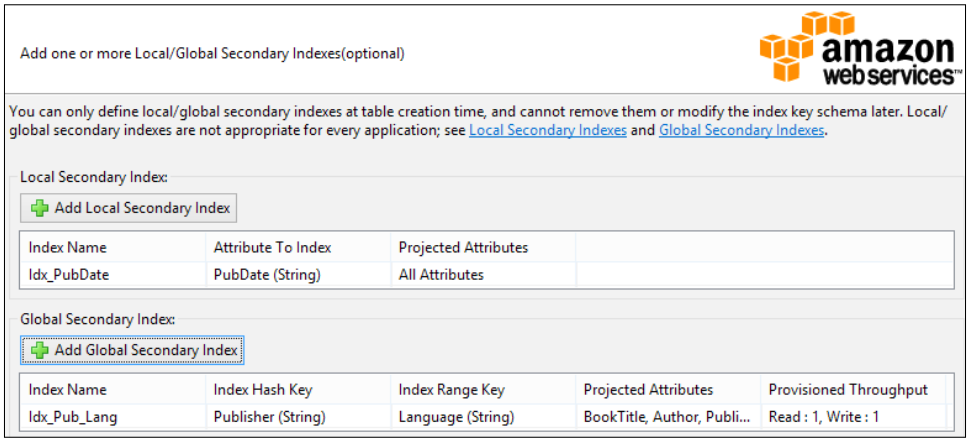
15. Once the desired region is selected, right-click on **Amazon DynamoDB** and choose **Create Table** option.



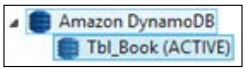
16. The **Create Table** option will open the following window and ask for the mandatory parameters to be passed while creating a table.



17. Since the index has to be created along with the table, you need to specify (if any) the secondary indexes here itself.

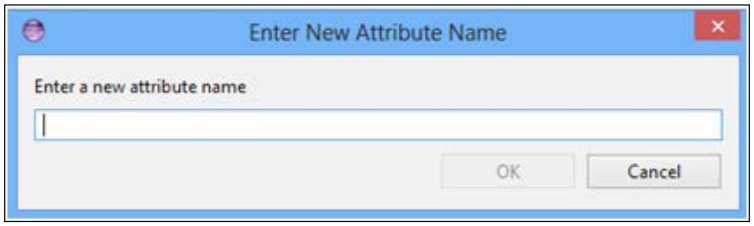


18. After that, the table will be created and it becomes active.



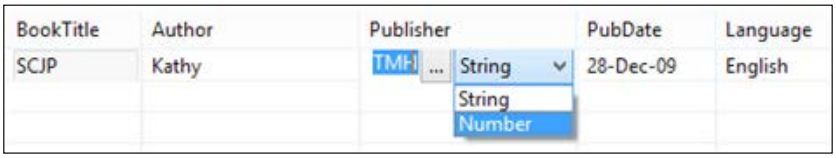
19. Double clicking on the table name will open the following tab. There will be five icons (just after the table name) in the tab. We will look at the functionalities of each icon later. Now, you need to focus only on the last icon, which is used to add attributes to the table.

20. Clicking on the "Add attribute" column icon will open the following window that asks us to enter the name of the attribute to be created (it will not ask for the type):

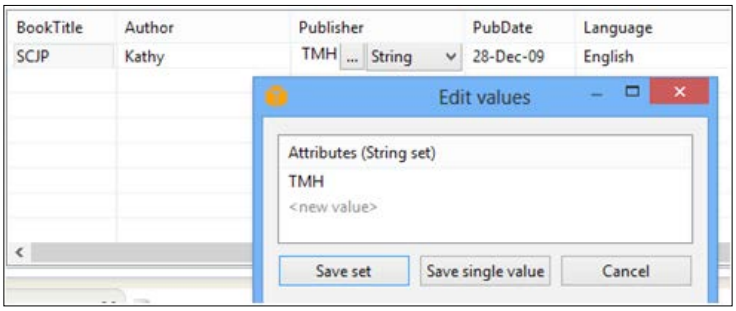


21. After adding the necessary attributes, click on the grid to insert a value. By default, all attributes will be treated as string type; to change its number type, just click on the button **a** (as shown in the **Publisher** column) in the following screenshot:

22. Clicking on the button **a** will open a drop-down from which we can choose whether we want the field to be a string or number. You might ask, "How can I insert a set data type?" The answer lies in the next screenshot:



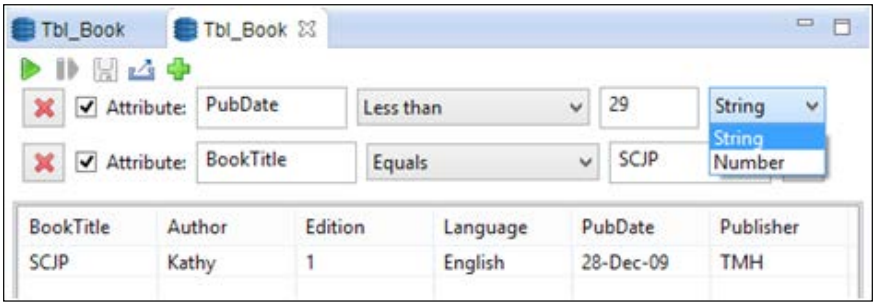
23. To change an attribute type from normal type to a set, just click on the icon with ellipses (**...**). It will open an **Edit values** window that asks whether we need to make this attribute a string or a string set. Click on **<new value>** to add new value to this set.



24. After inserting all the necessary attributes for an item, click on the grid below the item. The item with attribute values entered will appear in red color, which means the item is not saved into the table.

25. Let's discuss the DynamoDB utility icons. The first icon is used to run a scan on the DynamoDB table, the second icon is used to pause the scan, the third saves the items into the table, and the fourth icon is used to export table data into a CSV file. The last icon is used to add an attribute to the DynamoDB table.

26. Now, let's see how to scan a table using the Eclipse plugin. Clicking on the Add scan condition icon (in the previous screenshot) will open the following options. In the textbox next to the **Attribute** check box, enter the attribute name to be queried and select the comparison operation to be performed. Then, enter the selection criteria and click on the scan icon (first one). It will retrieve the result as follows:



Working with tools and code libraries

You have installed and configured the Eclipse plugin and AWS SDK tools in previous sections of this chapter. Let's continue with the DynamoDB example here.

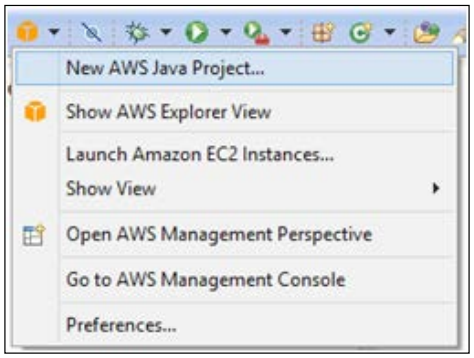
In this section, you are going to do some hands-on DynamoDB by interacting through Eclipse plugin and AWS SDK tool. Along with AWS Management Console, DynamoDB supports lots of tools and libraries, which is also applicable for other AWS services. Another important DynamoDB tool is DynamoDB Local. You can easily create tables, indexes, attributes, and items. After doing all of these offline, you can commit or save to AWS DynamoDB, thanks to DynamoDB Local.

If you are going to deploy a web application (let's say, a JSF application) and decide to use DynamoDB as the database, one of the biggest challenge would be integrating DynamoDB with Java. This is where the SDK comes into picture. By importing and including certain DynamoDB libraries, you can play with DynamoDB using simple Java code.

Creating an SDK project

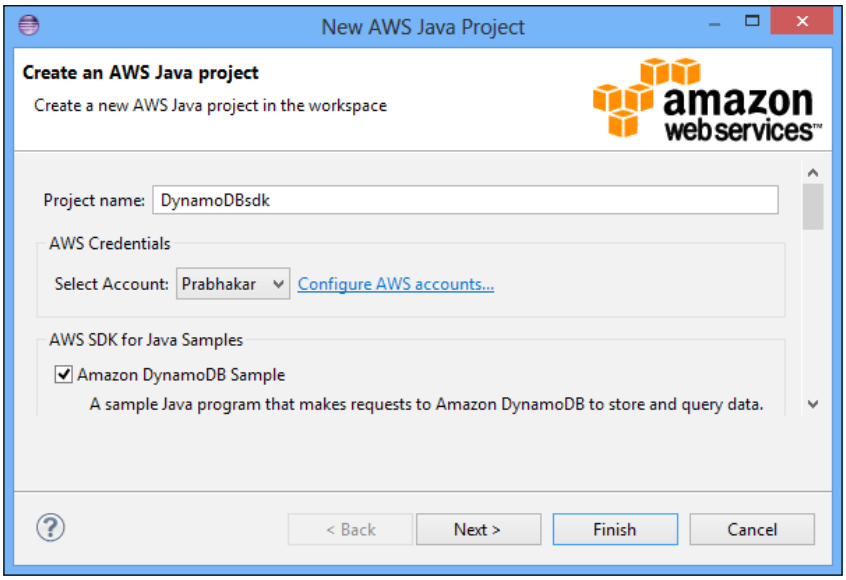
If you have already installed the Eclipse plugin, you will be able to see the credentials file created and you will be ready to work on the SDK plane.

1. Clicking on the AWS Eclipse Toolkit icon will provide us with the option to create new AWS project, as shown here:



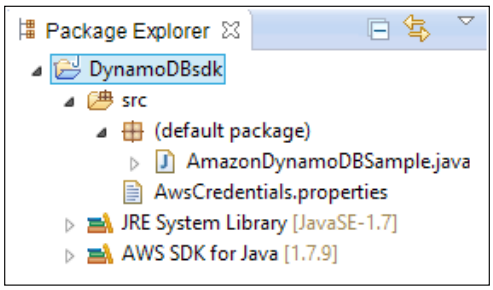
2. Now, we need to select **New AWS Java Project**. Clicking on this option will open the window shown in the next screenshot. Clicking on this option for the first time will present a few sample codes from AWS and will ask whether we want these sample codes to be part of the project.
3. It is recommended that you check the **Amazon DynamoDB Sample** check box for the first time to understand the syntax of DynamoDB table operations.
4. Once done, select the AWS account that is already configured or configure a new AWS account.

5. Click on the **Next** button to proceed.

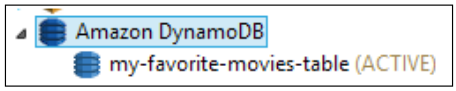


Clicking on the **Next** button will create a new project with the name specified in the previous window. The project structure is shown in the next screenshot.

6. In the `src` folder of the project, the credentials file will be made available by default. The sample DynamoDB code will also be available in the default package of this `src` folder in a file named `AmazonDynamoDBSample.java`.

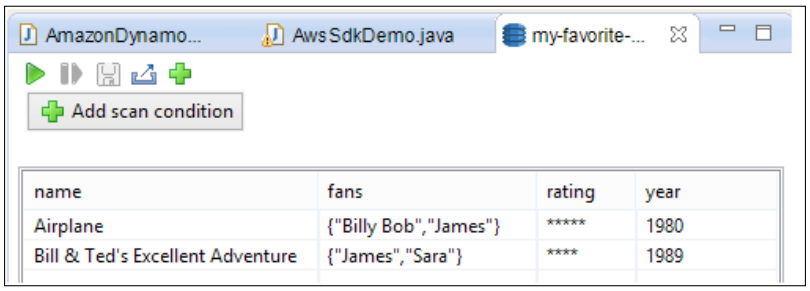


7. Since the sample code is provided by AWS, I don't want to get into trouble by providing the code here. So, we will see what that sample code does. First and foremost, it creates a table named **my-favorite-movies-table** in the **US_WEST_2** region. Once we have run this code, we need to open AWS Explorer and refresh **Amazon DynamoDB**, as shown here:



Make sure that you're selecting the correct region (**US_WEST_2**) in the AWS Explorer; otherwise, you cannot see the table being created.

8. Double-click on the table name to open the following window that shows the content of the table:



I hope that there is nothing I need to explain about the table's attribute names and types. In this table, **name** is the only key attribute.

The sample code provided will not have code for creating any indexes. First, you will create a new class named `AwsSdkDemo` in the same project.

In this DynamoDB class (`AwsSdkDemo`), you are going to perform the following DynamoDB operations:

- Initialize our AWS credentials
- Define the table attributes

- Define the key schema (of table and indexes)
- Define the provisioned throughput
- Create a table with the preceding parameters
- Describe the table
- Add (insert) items into the table

Java SDK operations

There are five user-defined private functions that are being invoked in the following code (you will learn each and every function in detail later):

```
public class AwsSdkDemo {
static AmazonDynamoDBClient client;
initializeCredentials();
String tableName = "dynamodb_table";
if (Tables.doesTableExist(client, tableName)) {
  System.out.println("Table " + tableName + " already EXISTS");
}
else {
  ArrayList<AttributeDefinition> attributeDefinitions =
getTableAttributes();
  ArrayList<KeySchemaElement> keySchemaElements =
getTableKeySchema();
  ProvisionedThroughput provisionedThroughput =
getProvisionedThroughput();

  CreateTableRequest request = new CreateTableRequest()
    .withTableName(tableName)
    .withAttributeDefinitions(attributeDefinitions)
    .withKeySchema(keySchemaElements)
    .withProvisionedThroughput(provisionedThroughput);
  CreateTableResult result = client.createTable(request);

  Tables.waitForTableToBecomeActive(client, tableName);
  TableDescription tableDescription = client.describeTable(
new DescribeTableRequest()
    .withTableName(tableName))
    .getTable();

  System.out.println("Created Table: " + tableDescription);
  putItems(tableName);
}}
```

The first method `initializeCredentials` is for loading our AWS credential and to authenticate ourselves to AWS in order to the program for performing the DynamoDB operation.

If we wish to perform DynamoDB operations, it must be done through the following client:

```
static AmazonDynamoDBClient client;
```

The following block will initialize the table name to the local variable. Then, the `if` condition will check whether there an existing table with the same name (in the client configured region) and return the Boolean value. If a table already exists, then the `syso` message will be printed. The code is as follows:

```
String tableName = "Tbl_Book";
if (Tables.doesTableExist(client, tableName)) {
  System.out.println("Table " + tableName + " already EXISTS");
}
```

The following block will create `CreateTableRequest` with attributes such as `tablename`, attribute definitions, key schema, provisioned throughput, and indexes:

```
CreateTableRequest request = new CreateTableRequest()
  .withTableName(tableName)
  .withAttributeDefinitions(attributeDefinitions)
  .withKeySchema(keySchemaElements)
  .withProvisionedThroughput(provisionedThroughput);
```

The following line will submit the table creation request through the DynamoDB client:

```
client.createTable(request);
```

The following line of code will pause further execution of the code until the table becomes active (most probably used before putting items in the table):

```
Tables.waitForTableToBecomeActive(client, tableName);
```

The following code will request you to describe the table name passed as a parameter to the client:

```
client.describeTable(new DescribeTableRequest()
  .withTableName(tableName))
  .getTable();
```

The following code will update a table with the passed `UpdateTableRequest` instance:

```
client.updateTable(updateTableRequest);
```

The default location of the credential file is `$USER_HOME/.aws/credentials`. But we have to keep your credentials file at `$USER_HOME/.aws/config` so that the SDK will easily identify it. Take a look at the following code:

```
private static void initializeCredentials() throws Exception {
AWSCredentials credentials = null;
try {
  credentials = new
ProfileCredentialsProvider().getCredentials();
  client = new AmazonDynamoDBClient(credentials);
  Region usWest2 = Region.getRegion(Regions.US_WEST_2);
  client.setRegion(usWest2);
} catch (Exception e) {
  throw new AmazonClientException(
    "Invalid location or format of credentials file.",e);
}
}
```

Now, let's take a look at what was done in the code snippet:

- The first line of `try` block will load the default AWS credentials
- The next line will configure the DynamoDB client with the loaded credential
- The third line will initialize the region to `US_WEST_2`, which is Oregon
- The last line of `try` block will set the region for DynamoDB client to `US-WEST-2`

If there is an improper location of the credential file, the following exception will be thrown:

```
private static void initializeCredentials() throws Exception {
AWSCredentials credentials = null;
try {
  credentials = new
ProfileCredentialsProvider().getCredentials();
  client = new AmazonDynamoDBClient(credentials);
  Region usWest2 = Region.getRegion(Regions.US_WEST_2);
```

```
  client.setRegion(usWest2);
} catch (Exception e) {
  throw new AmazonClientException(
    "Invalid location or format of credentials file.",e);
}
}
```

The following function will prepare `ArrayList` that adds all `AttributeDefinition` to it. Each `AttributeDefinition` will take two parameters: the attribute name and the attribute type, as shown in the code. In the following code, you are defining the two attributes:

```
private static ArrayList<AttributeDefinition> getTableAttributes() {
  ArrayList<AttributeDefinition> attributeDefinitions = new
ArrayList<AttributeDefinition>();
  attributeDefinitions.add(new AttributeDefinition()
    .withAttributeName("hashKey")
    .withAttributeType("S"));
  attributeDefinitions.add(new AttributeDefinition()
    .withAttributeName("rangeKey")
    .withAttributeType("N"));
  return attributeDefinitions;
}
```

The following method will return `ArrayList` of the type `KeySchemaElement`. Inside this function, you are instantiating an `ArrayList` of the type `KeySchemaElement`. To this `ArrayList`, we are adding two `KeySchemaElement`. The first element is for setting the attribute `hashKey` as key type `HASH` and the second element is for setting the attribute `rangeKey` as key type `RANGE`. Finally, you are returning this `ArrayList`. The code is as follows:

```
private static ArrayList<KeySchemaElement> getTableKeySchema() {
  ArrayList<KeySchemaElement> ks = new
ArrayList<KeySchemaElement>();
  ks.add(new KeySchemaElement()
    .withAttributeName("hashKey")
    .withKeyType(KeyType.HASH));
  ks.add(new KeySchemaElement()
    .withAttributeName("rangeKey")
    .withKeyType(KeyType.RANGE));
  return ks;
}
```

In the following function, you will try to put items (`item1` of type `Map<String, AttributeValue>`) into the table (whose name is taken as input parameter). As discussed in previous example (the `getTableKeySchema` method), every item must have primary key attributes (the `hashKey` and `rangeKey` attributes) so that both the items have them.

In the first item (`item1`), including primary key attributes, you are adding two more attributes—namely `numberSet` (the `NumberSet` type) and `stringSet` (the `StringSet` type). In order to add the attributes, you must call the correct method of the `AttributeValue` class, depending on the type of attribute you need to put. The code is as follows:

```
private static void putItems(String tableName) {
  Map<String, AttributeValue> item1 = new HashMap<String,
AttributeValue>();
  item1.put("hashKey", new AttributeValue().withS("hash1"));
  item1.put("rangeKey", new AttributeValue().withN("1"));
  item1.put("stringSet", new AttributeValue()
    .withSS(Arrays.asList("string1", "string2")));
  item1.put("numberSet", new AttributeValue()
    .withNS(Arrays.asList("3","2","1")));
  PutItemRequest putItemRequest = new PutItemRequest()
    .withTableName(tableName)
    .withItem(item1);
  client.putItem(putItemRequest);
}
```

The following method will return the `ProvisionedThroughput` instance with the populated read and write throughput capacities for your table. The long number (`2L`) here means the maximum read or write data size per second. This is usually measured in KBps. So, here, you are restricting the read-write speed to 2 KBps. The code is as follows:

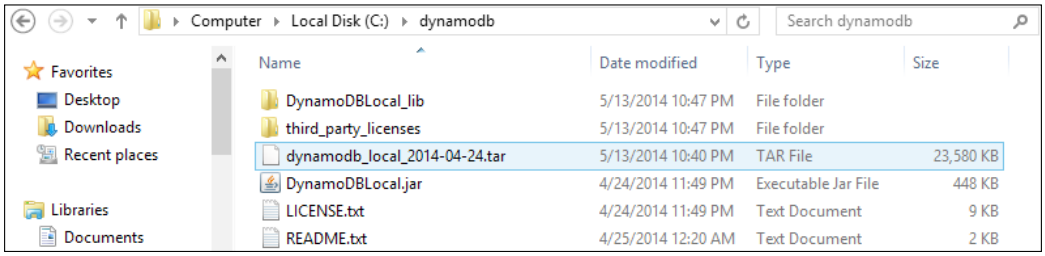
```
private static ProvisionedThroughput getProvisionedThroughput() {
  ProvisionedThroughput provisionedThroughput = new
ProvisionedThroughput()
      .withReadCapacityUnits(2L)
      .withWriteCapacityUnits(2L);
  return provisionedThroughput;
}
```

DynamoDB Local

DynamoDB Local is a local client-side database that emulates DynamoDB database in our local system. This is very helpful when developing an application that uses DynamoDB as the backend. After writing a module, in order to test whether the code works fine, you need to connect to Amazon and run it. This will consume a lot of bandwidth and a few dollars. To avoid this, you can make use of DynamoDB Local and test the code locally. Once the testing is done, you can make your application use the AWS DynamoDB service. This requires only three steps:

1. Download DynamoDB Local from `http://dynamodb-local.s3-website-us-west-2.amazonaws.com/dynamodb_local_latest`.
2. Start the DynamoDB Local service (should have JRE6 or later).
3. Point the code to use DynamoDB Local port.

There is no need to discuss more on how to download a file from the Internet. So, let's go to next point. The downloaded file might be a zipped one (`tar.gz`, `.zip`, or `.rar`). You need to extract it to a location. I have extracted it to `C:\dynamodb`, as shown here:

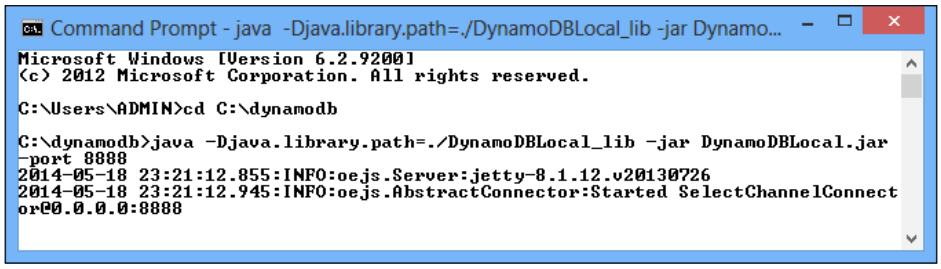


Starting DynamoDB Local is very easy. First, you need to change the working directory using the `cd` command, and then you can start DynamoDB Local on port 8888 using the following command:

```
java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar -port
8888
```

Even the command `java -D java.library.path = ./DynamoDBLocal_lib -jar DynamoDBLocal.jar` is enough to start DynamoDB Local, but it starts it on port 8000, which is occupied by my PC. That's why I use port 8888.

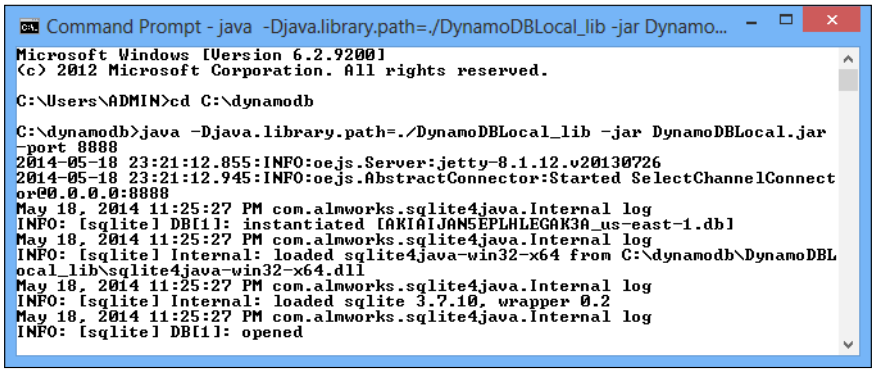
The following screenshot shows the output you'll get:



Once DynamoDB Local has started, it's easier to configure the client. You need to make changes in three lines of your `initializeCredentials` method. You need to insert a new line pointing to the DynamoDB localhost and port using the `client.setEndpoint()` method, as shown in following code snippet. Then, you need to remove other client-related setters such as `setRegion`, and so on. The code is as follows:

```
private static void initializeCredentials() throws Exception {
  AWSCredentials credentials = null;
  try {
    credentials = new  ProfileCredentialsProvider()
      .getCredentials();
  } catch (Exception e) {
    throw new AmazonClientException(
    "Invalid location or format of credentials file.",e);
  }
  client = new AmazonDynamoDBClient(credentials);
  client.setEndpoint("http://localhost:8888");
  //Region usWest2 = Region.getRegion(Regions.US_WEST_2);
  //client.setRegion(usWest2);
}
```

If you run the `AwsSdkDemo` class now, it will give the following output in the console (where DynamoDB Local started):

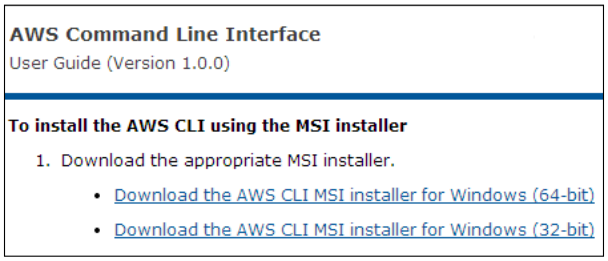


DynamoDB local stores all these data in local SQLite database.

Command-line interface

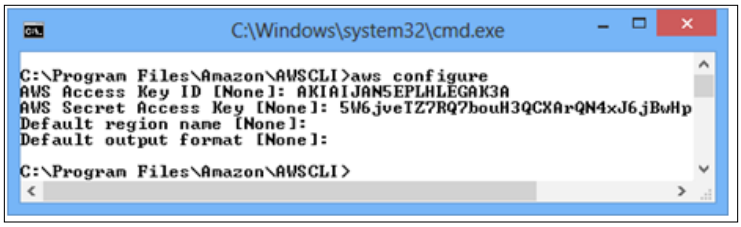
You can't beat the classics. Can you? So, let's look into the basics of DynamoDB. As the **Command-line interface (CLI)** provides good flexibility, it'll make life simpler for advanced programmers by reducing the number of clicks. This saves the time spent in writing commands and programs for redundant outputs.

To get the AWS CLI, go to the link mentioned in the following screenshot:

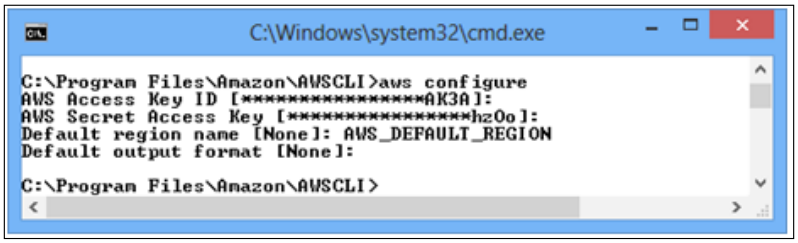


Go to `https://s3.amazonaws.com/aws-cli/AWSCLI64.msi` to download the AWS CLI setup. If you have Linux or OS X, steps will be slightly different. You can check this from the AWS documentation directly. Once installation is complete, go to the following path (in my case, it is `C:\Program Files\Amazon\AWSCLI`) in the command prompt—the path might differ based on the platform. Run the `aws configure` command to configure CLI with our AWS credentials.

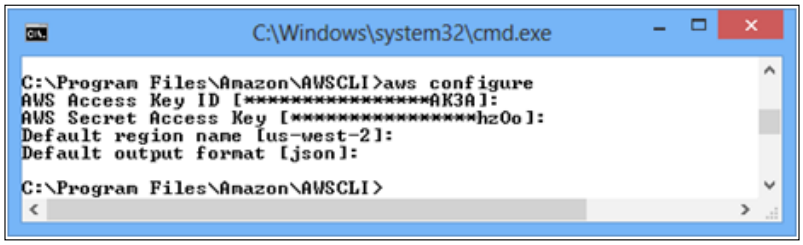
Hit the *Enter* key and you will be asked for the following four options:



If you feel that any of the parameters have changed at all, type in `aws configure` again and provide the necessary details. It is also possible to leave certain parameters empty.



As shown here, running the `aws configure` command will replace the old parameters with the newer parameters:



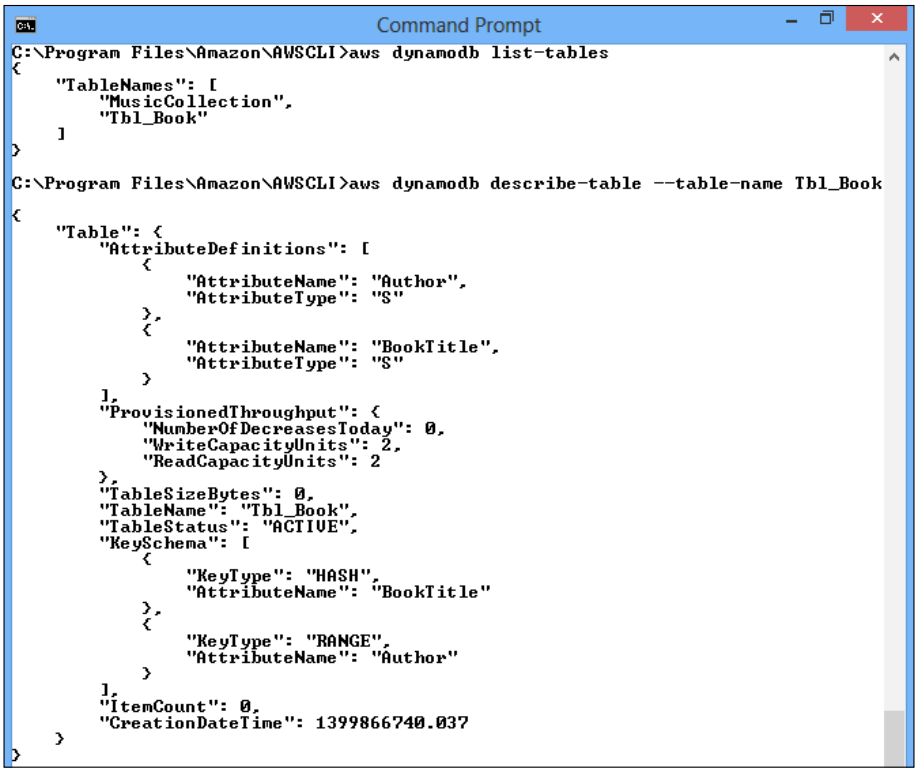
Now, you will see one of the simplest DynamoDB commands: table creation. You are going to create the same table (but without secondary indexes) you created using AWS Management Console with the help of the following command:

```
aws dynamodb create-table --table-name Tbl_Book --attribute-definitions
AttributeName=BookTitle,AttributeType=S AttributeName=Author,AttributeTyp
e=S --key-schema AttributeName=BookTitle,KeyType=HASH AttributeName=Autho
r,KeyType=RANGE --provisioned-throughput ReadCapacityUnits=2,WriteCapacit
yUnits=2
```

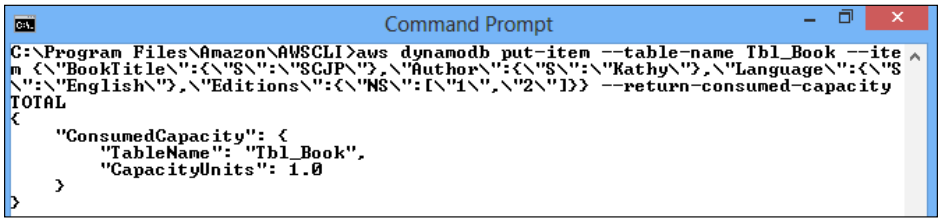
This will give the following output in your console:

The returned JSON will describe the table created.

In the following screenshot, we can see two operations. First is retrieving all the table names in the configured region using the `aws dynamodb list-tables` command. The second command is for describing a table using the `aws dynamodb describe-table --table-name Tbl_Book` command.



The following screenshot shows how an item can be inserted into DynamoDB table:



This command purely depends on the OS in which AWS CLI is installed.
For Windows 8, the command is as follows:

```
aws dynamodb put-item --table-name Tbl_Book --item {\"BookTitle\":{\"S\":
\"SCJP\"},\"Author\":{\"S\":\"Kathy\"}, \"Language\":{\"S\":\"English\"},
\"Editions\":{\"NS\":[\"1\",\"2\"]}}    --return-consumed-capacity TOTAL
```

For other platforms, if the command is throwing an error, you just need to replace `\"` with `"`. Then, it will work fine. Otherwise, type `aws dynamodb put-item help` to retrieve the format in which the request has to be made.

Summary

In this chapter, you learned about the features and uses of AWS code library and SDK. You have set up SDK and Eclipse Toolkit for your AWS account in Eclipse. Then, you performed Beanstalk deployment and understood the idea behind the Beanstalk service. After that, you looked into an example for DynamoDB from CLI by taking the example of a table. Then, you performed item operations on DynamoDB. After that, you had a look at the various tools and libraries for database services such as DynamoDB Local.

In the next chapter, you will learn about the AWS API and how to leverage the benefits of AWS APIs. You will see REST-based API examples and the best use cases of API at application level for developers.

7
Interacting with AWS Using API

There are several interfaces available to make user interaction easy and effective. There are four interfaces that are commonly used to interact with AWS:

- **AWS Management Console**: This is one of the most commonly used interfaces in AWS services because of its simplicity. End users prefer management console because this doesn't require any software to start with, just an Internet connection and a browser are sufficient.
- **Eclipse plugin**: As a commonly used open source IDE, this provides a plugin to work with AWS. Nowadays, plugins are available for people who are using Visual Studio.
- **Command-line Interface**: This provides good flexibility, which makes life simple for the advanced programmers or system administrators by reducing the number of clicks. They can use the extra time to write some commands (automation scripts in Bash or PowerShell) and program to do certain redundant jobs.
- **API tools**: These serve as the client interface to the Amazon EC2 web service. For simplicity, we can call them SDKs of AWS services.

AWS consists of real web services; they all are restricted over HTTP with different abstraction levels, like the different APIs. Some AWS services can be controlled using the **REST API**, some using the **Query API**, some by the **SOAP API**. Most often, a mixture is allowed on AWS for developers who can use any method from it as per knowledge and business/architecture requirements.

We will discuss the following topics in this chapter:

- REST-based APIs
- Authenticating requests using REST APIs
- AWS EC2 service API
- AWS DynamoDB service API

As a part of our discussion in this chapter, we will learn about all of these topics. Before that, we should have our security credentials in hand. We can access them from the security page in the AWS Management Console.

REST-based APIs

The REST interfaces presented by AWS use only and only the standard components of HTTP request messages to characterize the API action that is being performed. The HTTP standard components can be:

- **HTTP method**: This specifies the action that request will perform
- **Universal Resource identifier** (**URI**): This can be the path and query elements that signify the resource on which the action will be performed
- **Request headers**: This will be a part of a set of metadata that supplies more information about the request itself or about the requester
- **Request body**: This is the data on which the service will be performing some action

Web services that employ these mechanisms to explain operations are frequently termed *RESTful* services, a classification for services that use the HTTP protocol as it was originally intended.

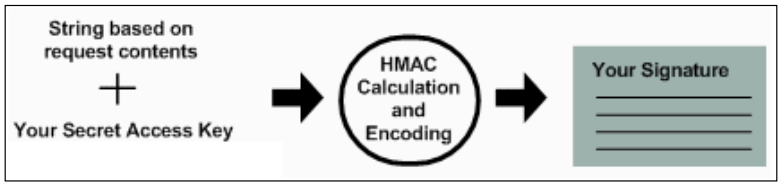
Authenticating requests using REST APIs

When you are going to access Amazon S3 or other AWS services using REST, you must provide the following items in your request so the request can be authenticated:

- **AWS access key ID**: Every request must hold the access key ID of the uniqueness you are using to send your request
- **Signature**: Each request must contain a valid request signature
- **Time stamp and date**: Each request must contain the date and time the request was created

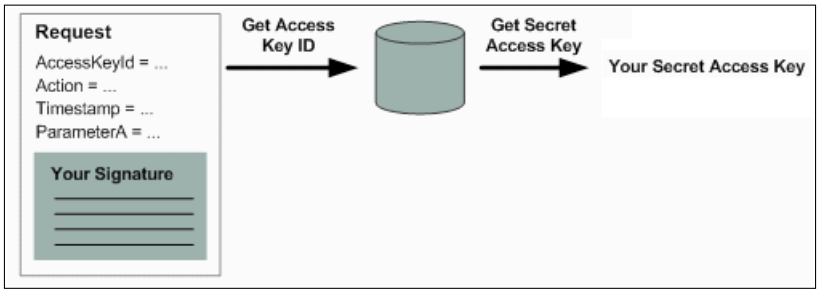
The general steps for authenticating a request are as follows:

1. Create a request that contains the following components:
    - Access key ID
    - Action
    - Timestamp
    - Parameters
2. Check the signature using your secret access key, as shown here:

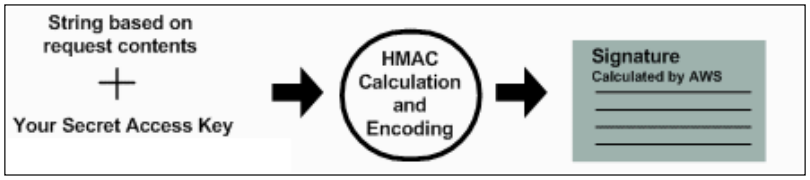


3. Send the request to AWS service by including your access key ID and the signature in your request.

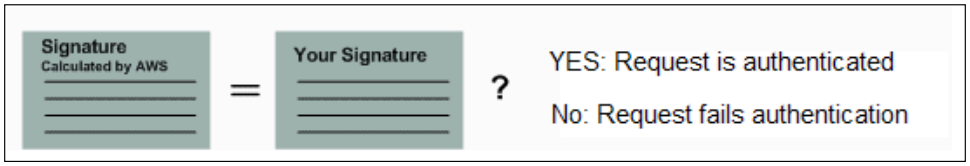
4. AWS will retrieve your Access key ID to check your secret access key.



5. AWS will compute a signature from the request data and the secret access key using the similar algorithm that you used to analyze the signature you sent in the request.



6. If the signature generated by AWS matches the one you have sent with the request, the request is considered as an authentic request.



This way, AWS is working for the authentication of a user and requests raised by user. After the authentication, you can proceed with API tools installation and configuration.

Getting started with API tools

There are multiple AWS services that have API available for use. In this chapter, you will learn about the basics of EC2 API and DynamoDB API in depth. To start with EC2 API, you will need the **X.509** certificate (you can find it under the AWS **Access Identifiers** page, in the **X.509 Certificates** section). Once this is done, you will need to download that certificate and the private key file. If you are a new to AWS, you should use IAM roles. We will go here with X.509 certificate.

To start with EC2 APIs, you should have basic knowledge of the following:

- XML
- Web services
- HTTP requests
- One or more programming languages, such as Java, PHP, Perl, Python, Ruby, C#, or C++

There are basic terms that will be frequently used in the API:

- **Endpoints**: This is simply a URL that will serve as an entry point for a web service.
- **Available libraries**: These libraries offer basic functions that without human intervention take care of tasks such as cryptographically signing your requests, retrying requests, and handling error responses so that it will be easier for you to get started.
- **Eventual consistency**: The Amazon EC2 API follows an eventual consistency model, due to the distributed personality of the system supporting the API. This means that the result of an API command you run that will affect your Amazon EC2 resources might not be immediately visible to the subsequent commands you run.

Installing API tools

To install API tools, follow the given steps. I am doing this on an Ubuntu machine, but for RedHat or OS X users, the command will vary:

1. Run the following command:

   ```
   sudo apt-get install ec2-api-tools
   ```

   If you do not have the latest Ubuntu release, the packages may be a bit old. So, add repository details by following commands:

   ```
   sudo apt-add-repository ppa:awstools-dev/awstools
   sudo apt-get update
   sudo apt-get install ec2-api-tools
   ```

2. Set the environment variables your shell profile by adding the following lines to your `~/.bashrc` file if you use Bash as your shell:

   ```
   export EC2_KEYPAIR=<your keypair name> # name only, not the file name
   export EC2_URL=https://ec2.<your ec2 region>.amazonaws.com
   export EC2_PRIVATE_KEY=$HOME/<where your private key is>/pk-XXXXXXXXXX.pem
   export EC2_CERT=$HOME/<where your certificate is>/cert-XXXXXXXXXXXX.pem
   export JAVA_HOME=/usr/lib/jvm/java-6-openjdk/
   ```

3. You are set up the API tools configuration. Now, you have to create one "keypair" that will be used to connect the instances using SSH. You can use the `ec2-add-keypair` utility to create the key and register your key with Amazon:

   ```
   ec2-add-keypair uchit-keypair
   ```

4. This will print out the private key that you will have to save in a file:

   ```
   cat > ~/.ec2/id_rsa-uchit-keypair
   ```

Running your first instance

To start an EC2 instance using API, you can use the following to create new instance using AMI. You should know the required AMI ID to launch you instance from that ID. To search AMI, you can use the following command:

```
ec2-describe-images -a
```

It will give you the whole list of AWS AMIs. To launch any instance with a specific AMI ID, you can use following command for reference:

```
ec2-run-instances ami-e348af8a -k uchit-keypair
```

Example of EC2 API

Here is a sample code to create EC2 instances with Amazon AWS SDK using APIs. This code will create an instance on AWS with configuration specified by you:

```
// for connecting to ec2
InputStream credentialsAsStream =
Thread.currentThread().getContextClassLoader().getResourceAsStream("Aw
sCredentials.properties");
Preconditions.checkNotNull(credentialsAsStream, "File
'AwsCredentials.properties' NOT found in the classpath");
AWSCredentials credentials = new
PropertiesCredentials(credentialsAsStream);

AmazonEC2 ec2 = new AmazonEC2Client(credentials);
ec2.setEndpoint("ec2.eu-west-1.amazonaws.com");

// to create ec2 instances
RunInstancesRequest runInstancesRequest = new
RunInstancesRequest()
```

```
        .withInstanceType("t1.micro")
        .withImageId("ami-62201116")
        .withMinCount(2)
        .withMaxCount(2)
        .withSecurityGroupIds("tomcat")
        .withKeyName("uchit")
```

To create an EC2 instance, you need to provide configuration parameters as described in the preceding code.

Use the following code for creating an instance:

```
.withUserData(Base64.encodeBase64String(myUserData.getBytes()))
;

RunInstancesResult runInstances =
ec2.runInstances(runInstancesRequest);

// to tag ec2 instances
List<Instance> instances =
runInstances.getReservation().getInstances();
int idx = 1;
for (Instance instance : instances) {
  CreateTagsRequest createTagsRequest = new CreateTagsRequest();
  createTagsRequest.withResources(instance.getInstanceId()) //
      .withTags(new Tag("Name", "ec2-api-test" + idx));
  ec2.createTags(createTagsRequest);

  idx++;
}
```

Now, let's discuss DynamoDB APIs with some more details. To start with DynamoDB API, you need to learn the data format that will be used in it.

Data format for DynamoDB

DynamoDB uses the JSON format to send the request and receive the response from the DynamoDB endpoint. One important rule is that the DynamoDB endpoint gets this JSON request and parses it into native format (which is not JSON). For example, JSON supports the date data type but DynamoDB does not support date. So, the request JSON should not have incompatible data type.

In order to avoid this frustration, DynamoDB has already listed the allowed data types and its representations as follows:

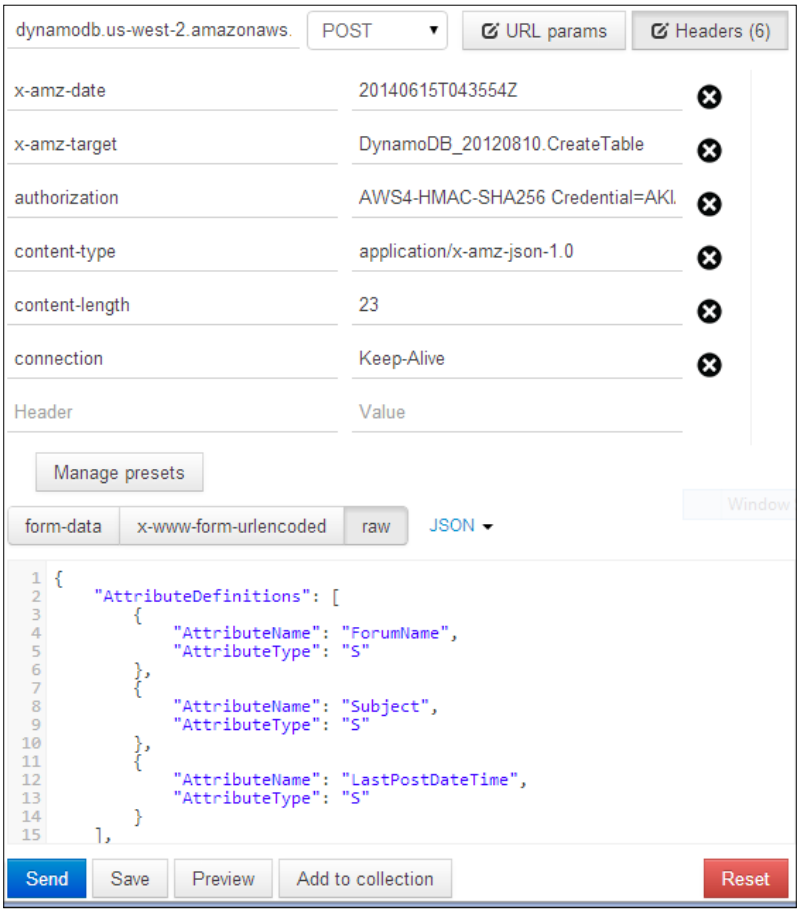
- `S` to denote **string** data type
- `N` to denote **number** data type
- `B` to denote **binary** data type
- `SS` to denote **string set** data type
- `NS` to denote **number set** data type
- `BS` to denote **binary set** data type

There are also Boolean, null, lists, and maps for document storage. In the following example requests, all the attribute names and the values will be placed within double quotes. This clearly means that all request parameters are sent as strings. While sending binary data, we need to first encode it with Base64 encoding and pass the encoded value as string. DynamoDB endpoint parses it by looking whether it belongs to any of the preceding data types. If it is proven negative, then DynamoDB will not process this request and sends response to the client that the JSON or request is invalid. I recommend that you explore the JSON syntax and its advantages, and then proceed through the rest of the chapter.

HTTP requests

As discussed, only the request and response are in JSON format. In both the client-side and the server-side, these JSON data (request and response) are parsed by SDK or browser (on the client side) and DynamoDB (on the server side). The SDKs and the CLI take care of secure request authentication and users are encouraged to leverage them. We can perform almost all kinds of DynamoDB operations through HTTP requests. All the possible operations are listed and we will discuss details in the last section of this chapter. Now, let's see the HTTP request structure.

I have used **Postman** (an extension of Google Chrome) to perform REST operations. Other than that, other software like cURL are also available for the same.



In the preceding screenshot, we need to understand the use of the three sections (on the right-hand side). The first section has a textbox, drop-down (with **POST** selected), and two buttons (**URL params** and **Headers**). This section is used to specify the information about endpoint URL and request method.

Clicking on the **Headers** button will open the second section, which is used to specify the header information. Here we are adding six header elements (as shown in the preceding screen).

The third (last) section is used to provide request body for the request. We can provide this as XML, text, or JSON. Whichever data format we choose, the body must be of the correct type. If we choose the wrong data format (or incorrect body content), the request will not reach the endpoint and it will result in the error, which we will discuss in the next section.

Request header

The DynamoDB "POST" request must have the following headers:

- `Host`: The host specifies the URL where the DynamoDB REST server (or endpoint) is located. All requests will be redirected to this server. Our database is located at "us-west-2" region, so we should specify the host as `dynamodb.us-west-2.amazonaws.com`. Instead, we can directly specify it in the address bar.
- `x-amz-date`: This header element is used to specify the timestamp of the request in ISO date time format. I have run this request on June 15, 2014 at 4:35 A.M. (GMT), so I have filled it with `20140615T043554Z`.
- `x-amz-target`: This header element is used to specify what kind of DynamoDB operation has to be performed. To put it in another way, this element give a hint to the endpoint about what is written in the request body. It will usually be in the format of "DynamoDB_<API-Version>.<Table-operation>", for example, `DynamoDB_20120810.CreateTable`.
- `Authorization`: This is a complex header element. It has lot of parameters, so we will observe what is written in the preceding screenshot without missing anything. `AWS4-HMAC-SHA256 Credential=AKIAIJAN5EPLHLEGAK3A/20140615/us-west-2/dynamodb/aws4_request,SignedHeaders=host;x-amz-date;x-amz-target,Signature=66534aa47c45417eaac116e94abce8185cfcdea5d206981ee76a09967620ca76`. We will discuss this in detail later.
- `Content-Type`: This header element is used to specify the JSON version of the request.
- `Content-Length`: This header element is used to specify the characters in the request body.
- `Connection`: This header element is used to specify whether the request has to be kept alive even after execution or terminate after some time.

The first parameter specified is the Amazon algorithm `AWS4-HMAC-SHA256`, which is used to hash the security parameters.

The second parameter is the `Credential` parameter in the format access key ID, current date in YYYY-MM-DD format, region where the table is available, service name, and the termination string `aws4_request`. Each of these parameters are separated by the / symbol. For example, `Credential=AKIAIJAN5EPLHLEGAK 3A/20140615/us-west-2/dynamodb/aws4_request`.

The third parameter is the header elements that have to be added to the request, and each element is separated by the `;` symbol. For example, `SignedHeaders=host;x-amz-date;x-amz-target`.

The last parameter is the most important parameter: the `Signature` parameter. `Signature=66534aa47c45417eaac116e94abce8185cfcdea5d206981ee76a09967620ca76`. Finding or calculating this parameter is very complex. You can have a look at `http://docs.aws.amazon.com/general/latest/gr/signature-version-4.html` for further information.

Request body

The header is used for authentication and to provide metadata for the REST operation. The request body specifies what will be performed. For example, the header specifies that we need to perform table creation, but the table name, attribute configurations, and so on must be specified in the request body.

In most of the DynamoDB operations, the request header will remain same (except in places like `x-amz-target`, where it will change based on the operation we need to perform), but the request body will change for every request. So, we will discuss this in the last section of this chapter.

Response header

Response header generally has six parameters. Three of those parameters are common, such as `Content-Type`, `Content-Length`, and `Date`. The other three parameters are as follows:

- `HTTP/1.1`: This header will have the status of the response (to a request) from the server. If it is `200`, it means success. `4xx` means client-side error and `5xx` means server-side error. This will be covered in detail in the next section.
- `x-amzn-RequestId`: This returned request ID can be used for debugging the request.

- `x-amz-crc32`: This is the checksum returned by the DynamoDB server. We need to calculate the checksum of the returned data with this number. If the checksum doesn't match, it means somebody has done something nasty and data loss has occurred during data transfer. Then, we must run the request once again. In SDK, this retry happens automatically. If requests are sent over HTTPS, then man-in-the-middle attack is not an issue and the checksum serves to simply validate the output correctness, as they usually do. If requests are not sent over HTTPS, then having a checksum does nothing to prevent a man-in-the-middle attack.

Operations in DynamoDB

Most DynamoDB REST API supports all the possible operations. The possible operations are as follows:

- CreateTable
- PutItem
- UpdateItem
- GetItem
- Query
- Scan
- DeleteItem
- DescribeTable
- UpdateTable
- DeleteTable
- ListTables
- BatchGetItem
- BatchWriteItem

For all of these operations, only two things will change: the request body and `x-amz-target` that specifies what kind of table operation has to be performed. This attribute is same as that of the name of the operation (as shown in preceding bullet list). For example, for performing the DescribeTable operation, `x-amz-target` is `DynamoDB_20120810.DescribeTable itself`).

CreateTable

To perform the CreateTable operation, the request JSON will be as follows:

```
{
    "AttributeDefinitions": [
  { "AttributeName": "BookTitle", "AttributeType": "S" },
  { "AttributeName": "Author", "AttributeType": "S"},
  { "AttributeName": "PubDate","AttributeType": "S"},
  { "AttributeName": "Language", "AttributeType": "S"},
  { "AttributeName": "Edition", "AttributeType": "N" }   ],
    "TableName": "Tbl_Book",
    "KeySchema": [
  {"AttributeName": "BookTitle",  "KeyType": "HASH"},
  { "AttributeName": "Author", "KeyType": "RANGE"}   ],
    "LocalSecondaryIndexes": [
  { "IndexName": "Idx_PubDate", "KeySchema": [
{"AttributeName": "BookTitle", "KeyType": "HASH"  },
{ "AttributeName": "PubDate","KeyType": "RANGE" }  ],
          "Projection": {  "ProjectionType": "KEYS_ONLY"  }  }  ],
  "GlobalSecondaryIndexes": [
  { "IndexName": "Idx_Pub_Edtn", "KeySchema": [
{"AttributeName": "Language", "KeyType": "HASH"  },
{ "AttributeName": "Edition","KeyType": "RANGE" }  ],
          "Projection": {  "ProjectionType": "KEYS_ONLY"  }  }  ],
    "ProvisionedThroughput":
    {"ReadCapacityUnits": 2, "WriteCapacityUnits": 2}
}
```

The preceding code will create the table `Tbl_Book` with same schema that we have discussed in former chapters. The response to this request is same as that of the DescribeTable operation.

PutItem

The following code will put an item into the table `Tbl_Book`:

```
{
    "TableName": "Tbl_Book",
    "Item": {
        "BookTitle": {
            "S": "SCJP"
        },
```

```
            "Author": {
                "S": "Kathy"
            },
            "Publisher": {
                "S": "TMH"
            },
            "PubDate": {
                "S": "28-Dec-09"
            },
            "Language": {
                "SS": [
                    "English",
                    "German"
                ]
            },
            "Edition": {
                "N": "1"
            }
        },
        "Expected": {
            "BookTitle": {
                "ComparisonOperator": "NULL"
            },
            "Author": {
                "ComparisonOperator": "NULL"
            }
        }
    }
```

Here, we might be questioning the use of the new field in the JSON called `Expected`. By default, if `SCJP` (`BookTitle`) and `Kathy` (`Author`) are already available in the table, then the older items will be replaced with the newer ones. To prevent the newer item overwriting the older one, the `ComparisonOperator` must be set to `NULL`.

UpdateItem

The following code (put in request body) will update the item's (whose `BookTitle` is `SCJP` and `Author` is `Kathy`) `Language` attribute set to hold `English`, `German` and `Latin`. This updating will happen only if the older value of the `Language` set is `English` and `German`. This is the use of `Expected`.

```
    {
    "TableName": "Tbl_Book",
```

```
"Key": {
        "BookTitle": { "S": "SCJP"  },
        "Author": { "S": "Kathy"  }  },
"AttributeUpdates": {
        "Language": {
              "Value": { "SS": ["English","German","Latin"]},
              "Action": "PUT"  } },
"Expected": {
        "Language": {
      "ComparisonOperator":"EQ",
      "AttributeValueList": [ { "SS": ["English","German"]} ] }},
"ReturnValues": "ALL_NEW"
}
```

The following code will increment the `Edition` attribute value of the item whose `BookTitle` is `SCJP` and `Author` is `Kathy`.

```
{
"TableName": "Tbl_Book",
"Key": {
"BookTitle": { "S": "SCJP"  },
     "Author": { "S": "Kathy"  }  },
"AttributeUpdates": {
        "Edition": {"Action": "ADD","Value": {"N": "1"}  }  },
"ReturnValues" : "NONE"
}
```

GetItem

The following code will retrieve the item's (`BookTitle`, `Language`, and `Edition`) attributes whose `BookTitle` is `SCJP` and `Author` is `Kathy`.

```
{
    "TableName": "Tbl_Book",
    "Key": {
"BookTitle": { "S": "SCJP"  },
    "Author ": { "S": "Kathy" } },
    "AttributesToGet": ["BookTitle","Language","Edition"],
    "ConsistentRead": true,
    "ReturnConsumedCapacity": "TOTAL"
}
```

The preceding code will return the `ConsumedCapacity` value and result of the operation as follows:

```
{
    "ConsumedCapacity": {"CapacityUnits": 1,"TableName":
"Tbl_Book"  },
    "Item": {
        "BookTitle": { "S": "SCJP"},
        "Language": {"SS": ["English","German"]   },
        "Edition": {  "N": "1"  }}
}
```

Query

The following code will perform the Query operation on the index items, with `PubDate` between `2009-12-28` and `2012-07-28` and `BookTitle` as `SCJP`:

```
{
    "TableName": "Tbl_Book",
    "IndexName": "Idx_PubDate",
    "Select": "ALL_ATTRIBUTES",
    "Limit": 30,
    "ConsistentRead": true,
    "KeyConditions": {
        "PubDate": {
            "AttributeValueList": [
                {
                    "S": "2009-12-28"
                },
                {
                    "S": "2012-07-28"
                }
            ],
            "ComparisonOperator": "BETWEEN"
        },
        "BookTitle": {
            "AttributeValueList": [
                {
                    "S": "SCJP"
                }
            ],
            "ComparisonOperator": "EQ"
        }
    },
    "ReturnConsumedCapacity": "TOTAL"
}
```

Even though we specified the item limit of the query as `30`, there are only two items satisfying these conditions, so it returns those items, as shown in the following code snippet:

```
{
"Count": 2,
"Items": [
 {
  "BookTitle": {"S": "SCJP"}, "Author": {"S": "Kathy"},
  "Publisher": {"S": "TMH"},"PubDate": {"S": "2009-12-28"},
  "Language": {"SS": ["English", "German"]}, "Edition":
{"N": "1"}},
 {
  "BookTitle": {"S": "SCJP"}, "Author": {"S": "Khalid A M"},
  "PubDate": {"S": "2010-10-28"},"Language": {"SS": ["English"]}}],
  "ConsumedCapacity": {"TableName": "Tbl_Book","CapacityUnits":
1}}
```

The following code returns the number of items satisfying the condition (`BookTitle` equals `SCJP`):

```
{
    "TableName": "Tbl_Book",
    "Select": "COUNT",
    "ConsistentRead": true,
    "KeyConditions": {
   "BookTitle": {
            "AttributeValueList": [{"S": "SCJP"}],
            "ComparisonOperator": "EQ"}}
}
```

Here is the response for the preceding HTTP request:

```
{
    "Count":'3
}
```

Scan

The following code will scan the `Tbl_Book` table and return all the items. The response will look exactly like that of Query operation. So, we are skipping the explanation (to save paper). Here is the code:

```
{
    "TableName": "Tbl_Book",
    "ReturnConsumedCapacity": "TOTAL"
}
```

The following code will apply a scan filter (`Publisher` must be `TMH`) to the scan operation.

```
{
    "TableName": "Tbl_Book",
    "ScanFilter": {
   "Publisher": {
            "AttributeValueList": [{"S": "TMH"}],
            "ComparisonOperator": "EQ"}},
    "ReturnConsumedCapacity": "TOTAL"
}
```

DeleteItem

The following code will delete the item whose `BookTitle` is `SCJP` and `Author` is `Kathy`. Response to this request will display the deleted item attributes and its values. The code is as follows:

```
{
    "TableName": "Tbl_Book",
    "Key": {"BookTitle": { "S": "SCJP" },"Author": { "S": "Kathy"  }
},
    "ReturnValues": "ALL_OLD"
}
```

DescribeTable

The following code will display all the table schema of the table with name `Tbl_Book`. As hinted in the CreateTable operation, the response of the DescribeTable and CreateTable requests will be almost similar (except the `TableStatus`). After performing the CreateTable request, the `TableStatus` variable will have the `CREATING` status. So, if we describe the same table after some point of time (after the table has become active), its status will become `ACTIVE`.

To give table name, use the following code snippet:

```
{
    "TableName":"Tbl_Book"
}
```

Here is the output of DescribeTable operation with `TableStatus` as `ACTIVE`:

```
{
    "Table": {
    "AttributeDefinitions": [
    { "AttributeName": "BookTitle", "AttributeType": "S" },
    { "AttributeName": "Author", "AttributeType": "S"},
    { "AttributeName": "PubDate","AttributeType": "S"},
    { "AttributeName": "Language", "AttributeType": "S"},
    { "AttributeName": "Edition", "AttributeType": "N" }   ],
    "CreationDateTime": 1.363729002358E9,
    "ItemCount": 5,
    "KeySchema": [
          {"AttributeName": "BookTitle","KeyType": "HASH"},
          {"AttributeName": "Author","KeyType": "RANGE"}],
   "LocalSecondaryIndexes": [
    { "IndexName": "Idx_PubDate", "KeySchema": [
      {"AttributeName": "BookTitle", "KeyType": "HASH"  },
      { "AttributeName": "PubDate","KeyType": "RANGE" }  ],
    "Projection": {  "ProjectionType": "KEYS_ONLY"  }  }  ],
   "GlobalSecondaryIndexes": [
    { "IndexName": "Idx_Pub_Edtn", "KeySchema": [
      {"AttributeName": "Language", "KeyType": "HASH"  },
      { "AttributeName": "Edition","KeyType": "RANGE" }  ],
    "Projection": {  "ProjectionType": "KEYS_ONLY"  }  }  ],
    "ProvisionedThroughput": {
        "NumberOfDecreasesToday":0,
    "ReadCapacityUnits":5,"WriteCapacityUnits":5},
    "TableName": "Tbl_Book", "TableSizeBytes": 0,
    "TableStatus": "ACTIVE" }
}
```

UpdateTable

The following request will update a table's provisioned throughput capacity (both read and write) to `5`:

```
{
    "TableName": "Tbl_Book",
    "ProvisionedThroughput": {"ReadCapacityUnits":5,
"WriteCapacityUnits":5}
}
```

DeleteTable

To delete a table, use the following request:

```
{
    "TableName": "Tbl_Book"
}
```

Here is the response for the DeleteTable request, displaying information about the table being deleted:

```
{
    "TableDescription": {
        "ItemCount": 5,
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 5,"WriteCapacityUnits": 5},
        "TableName": "Tbl_Book", "TableSizeBytes": 0,
        "TableStatus": "DELETING"}
}
```

ListTables

The following code lists three tables whose names start with `Tbl_Book` (should be placed after `Tbl_Book`, that is, `Tbl_Library` because L comes after B).

```
{
    "ExclusiveStartTableName": "Tbl_Book ", "Limit": 3
}
```

We have only two tables in our account, so the response will have only two table names in the array:

```
{
    "LastEvaluatedTableName": "Tbl_Library",
    "TableNames": ["Tbl_Book","Tbl_Library"]
}
```

BatchGetItem

The following code tries to fetch three items from `Tbl_Library` and one item from `Tbl_Book`:

```
{
    "RequestItems": {
```

```
        "Tbl_Library": {
            "Keys": [
                {"Name":{"S":"Library of Congress"}},
                {"Name":{"S":"National Diet Library"}},
                {"Name":{"S":"Royal Danish Library"}}],
            "AttributesToGet": ["Country","City"]},
    "Tbl_Book": {
            "Keys": [
    {"BookTitle": { "S": "SCJP" }, "Author ": { "S": "Kathy" } }],
            "AttributesToGet": ["BookTitle","Language","Edition"]}},
    "ReturnConsumedCapacity": "TOTAL"
}
```

The response of BatchGetItem operation is as follows:

```
{
    "Responses": {
        "Tbl_Library": [
            {
                "Name":{"S":"Library of Congress"},
                "Country":{"S":"United States"},"City":{"S":"Washingt
onDC"} },
            {
                "Name":{"S":"National Diet Library"},
                "Country":{"S":"Japan"},"City":{"S":"Tokyo"} },
            {
                "Name":{"S":"Royal Danish Library"},
                "Country":{"S":"Denmark"},"City":{"S":"Copenhagen"} }]
    "Tbl_Book": [
            {
      "BookTitle": { "S": "SCJP" },
      "Language": { "SS": ["English","German"]   },
      "Edition": {  "N": "1"  } }]},
    "UnprocessedKeys": {},
    "ConsumedCapacity": [
        {"TableName": "Tbl_Library","CapacityUnits": 3},
        {"TableName": "Tbl_Book","CapacityUnits": 1}]
}
```

BatchWriteItem

The following request tries to write two items into `Tbl_Library` and one item into `Tbl_Book`. It is also possible to send DeleteRequest in place of `PutRequest` simultaneously for a table. The code is as follows:

```
{
    "RequestItems": {
        "Tbl_Library": [
            {
"PutRequest": {
  "Item": {
    "Name":{"S":"Harvard University Library"},
    "Country":{"S":"United States"},
    "City":{"S":"Massachusetts"} } } },
            {
"PutRequest": {
  "Item": {
    "Name":{"S":"Vernadsky National Library"},
    "Country":{"S":"Ukraine"},
    "City":{"S":"Kiev"} } } }]
    "Tbl_Book": [
            {
"PutRequest": {
        "Item": {
        "BookTitle": { "S": "SCJP" },
        "Author": { "S": "Brendon" },
        "Language": { "SS": ["English"] },
        "Edition": {  "N": "5" } }} } ]},
    "ReturnConsumedCapacity": "TOTAL"
}
```

The response for the BatchWriteItem request is as follows:

```
{
    "UnprocessedItems": { },
    "ConsumedCapacity": [
        {"TableName": "Tbl_Library","CapacityUnits": 2},
        {"TableName": "Tbl_Book","CapacityUnits": 1}]
}
```

Summary

In this chapter, we learned how you can use SDK for performing an EC2 instance launch and DynamoDB operations. You also learned about things that you had not come across in the previous chapters (for example, BatchWriteItem and BatchReadItem for an instance).

In the next chapter, you will learn how to migrate and host existing/new app on AWS and how to identify appropriate services for the app. You will also learn how to use the Elastic Beanstalk container service, AWS CloudTrail, and CloudFormation, and how to perform Auto Scaling based upon the requirements of the end user traffic.

8
Amazon Beanstalk, CloudTrail, and Data Warehouse Services

AWS provides application management and deployment services that help you build, deploy, and scale your applications instantly. You can use application management and deployment services to influence other AWS services without having to manage each of them discretely and manually.

In this chapter, you will learn about the following topics:

- Application deployment using AWS Elastic Beanstalk
- Getting started with Amazon Redshift
- Interacting with AWS CloudTrail
- Migrating an application to the Cloud

So, let's start with Amazon Elastic Beanstalk first.

Application deployment using AWS Elastic Beanstalk

To start with AWS Elastic Beanstalk, first you have to learn some of the basics of AWS SDK and toolkit. If you don't want to go with Eclipse, you can go ahead with the AWS Management Console to get the GUI feel. The AWS Toolkit for Eclipse will install and configure the modern form of the AWS SDK for the platform you have selected.

To start with Elastic Beanstalk using AWS SDK, you need the following things installed on your system:

- Eclipse (install client PC)
- AWS Toolkit for Eclipse
- AWS SDK for Java
- AWS IAM credentials (access key ID and secret access key)

From Eclipse, you can easily manage, customize, build, and deploy any of the illustrations incorporated in the SDK packages. The steps are as follows:

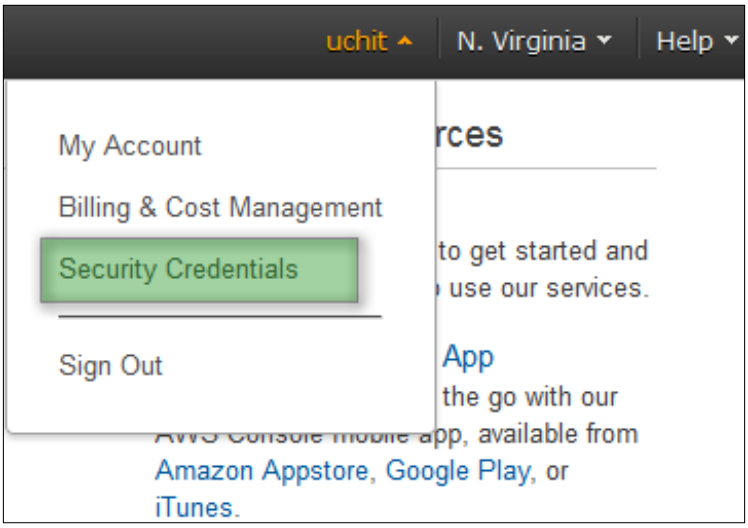
1. Go to `http://aws.amazon.com/sdk-for-java/`. Click on **AWS Toolkit for Eclipse**, as in the following screenshot:

2. There is another way to configure Eclipse Download Eclipse Juno/Luna+, from `https://www.eclipse.org/downloads/`.
3. Start Eclipse.
4. Go to **Help** | **Install New Software**.
5. In the **Work with** box, type `http://aws.amazon.com/eclipse`, and then press the *Enter* key.
6. In the list that emerges, expand **AWS Toolkit for Eclipse**.
7. Add a check mark next to **AWS Toolkit for Eclipse** to download.
8. Click on **Next** and the Eclipse wizard will take you through the steps of the installation processes by default.

To start AWS through Toolkit for Eclipse, you have to configure the Eclipse Toolkit with your access key ID and the secret access key that should be available in your AWS account. Apart from allowing Toolkit for Eclipse to start your account, your access keys can also be used to sign the web services-based requirements to AWS. Allowing web services requests ensures that only approved programs can make such requests. Moreover, by associating access keys with each web services request, AWS will be able to track service usage for billing and monitoring purposes.

The access keys have a combination of an access key ID and secret access key, which will be used to sign programmatic logical request that you will compose from the application source code to AWS to access assets. If you don't have access keys, you can obtain the keys from the AWS Management Console too. For that, go to **Security Credentials** and select **Access Key ID** from, as shown in the next screenshot:

Do not issue or use root account credentials. Always use IAM credentials for safety and security.

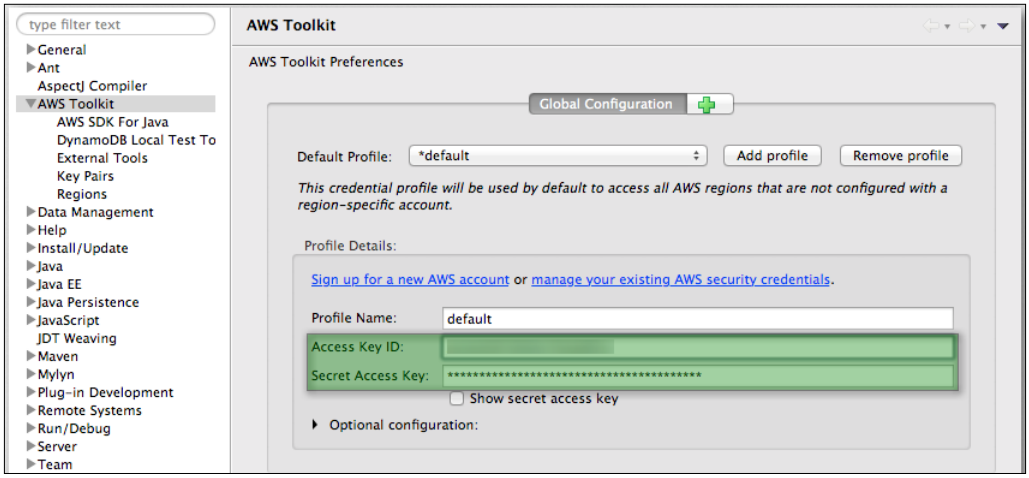


Keep your credentials confidential in order to protect your account and never e-mail it. Do not distribute it to any other individual from your organization, even if a query comes from AWS or from any other channel.

To configure your access keys in the Eclipse Toolkit, perform the following steps:

1. Open Eclipse's **Preferences** dialog box and click on **AWS Toolkit** located in the sidebar.
2. Type your access key ID in the **Access Key ID** box.
3. Type your secret access key in the **Secret Access Key** box.
4. Click on **Apply** or **OK** to store your access key information.

Here is an example of a configured **AWS Toolkit Preferences** screen with the **default** account:



The **AWS Toolkit Preferences** dialog box lets you include access information for multiple AWS account by choosing a profile. These accounts can be active if they empower developers and administrators to separate resources used in the development stage from those used in the production stage.

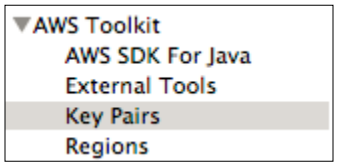
To supplement another set of access keys, perform the following steps:

1. On the **AWS Toolkit Preferences** screen, go to **Preferences** dialog box, and click on the **Add Account** button.
2. Add your new account details in the **Account Details** section.

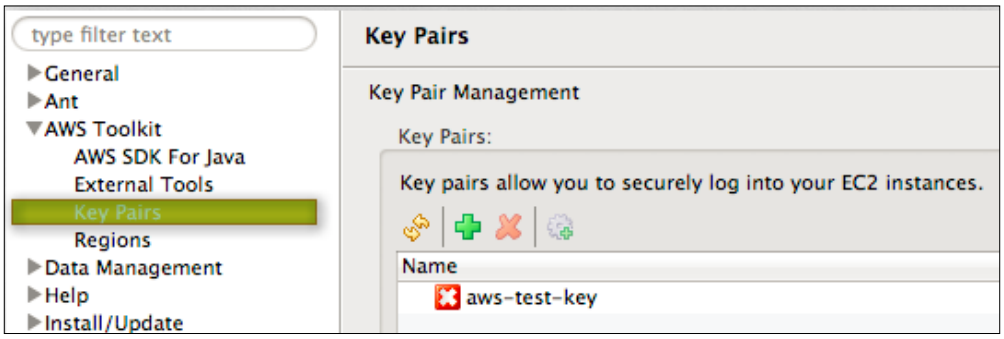
3. Pick an evocative name as the **Account Name** and enter your access key details in the **Access Key ID** and **Secret Access Key** boxes.
4. Click on **Apply** or **OK** to save your access key details. One can reiterate this procedure for as many sets of AWS account information as required.

Toolkit for Eclipse can also obtain your Amazon EC2 key pairs from the AWS account. However, you have to associate private keys with them to practice them in Toolkit for Eclipse manually. To examine your Amazon EC2 key pairs in Toolkit for Eclipse, follow the steps given here:

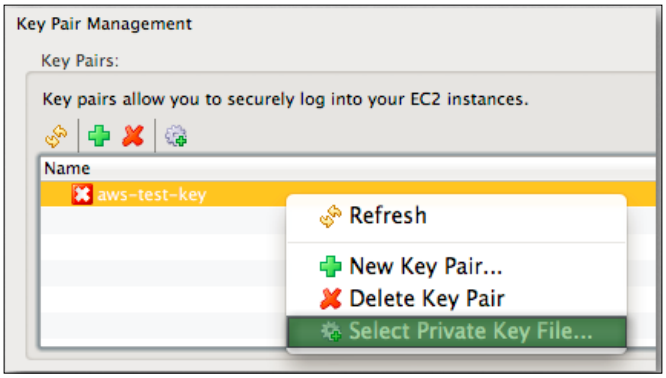
1. Open the Eclipse **Preferences** dialog box and click on the triangle next to **AWS Toolkit** in the sidebar to display further categories of Toolkit for Eclipse settings and configure them.



2. Go to **Key Pairs**., Eclipse will display a list of your available key pairs in that window. If a key pair has a red "X" mark next to it, you will have to link a private key with the key pair to use it with your current illustration. This is shown in the following screenshot:



3. Right-click on the key pair and click on the **Select Private Key File** option from the context menu:



4. Navigate to the private key file and choose it to acquaint it with your key pair.

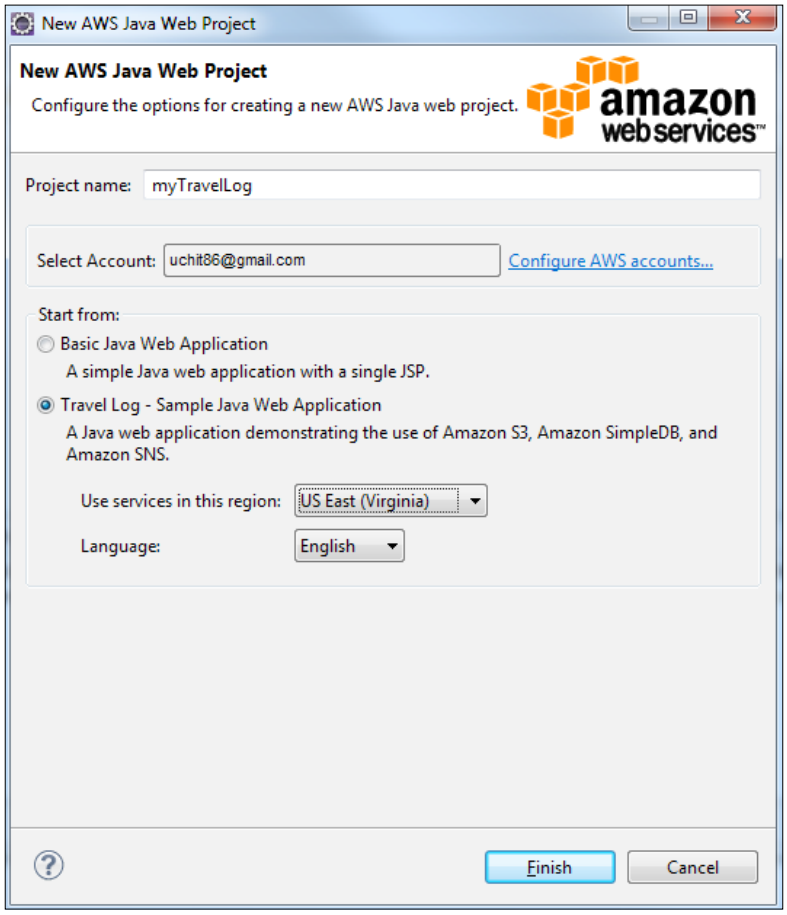
To deploy the web application, you have to perform the following steps:

1. On the Eclipse toolbar, click on the **AWS** icon, and click on **New AWS Java Web Project**.
2. In the **New AWS Java Web Project** dialog box, go to the **Start from** area of the dialog, click on **Travel Log** (which is a sample Java web application), and provide a name (such as `myTravelLog`) in the **Project name** box.

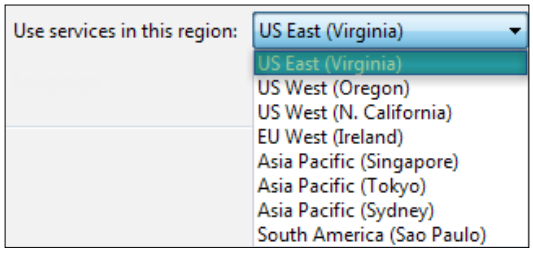
The application code can be downloaded from the Packt Publishing website.

3. Click on the **Finish** button. The Toolkit will generate the project and the project will be visible in **Project Explorer**.

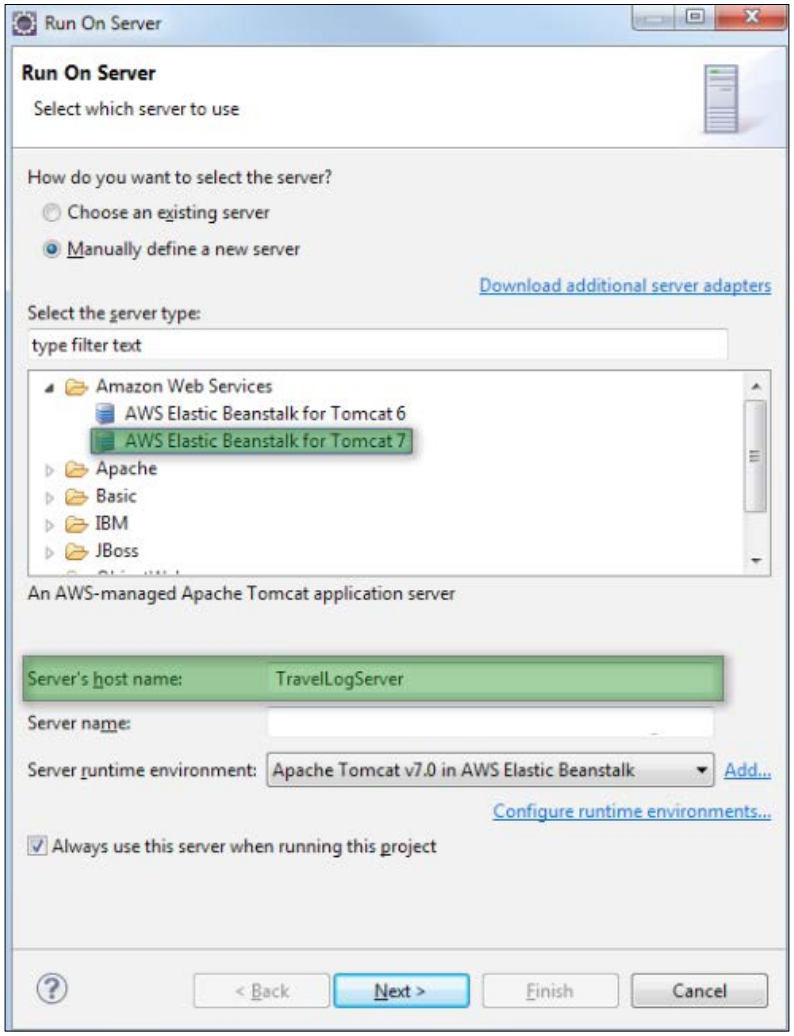
If **Project Explorer** isn't visible in Eclipse, navigate to **Window** | **Show View** | **Project Explorer**. This is shown in the following screenshot:



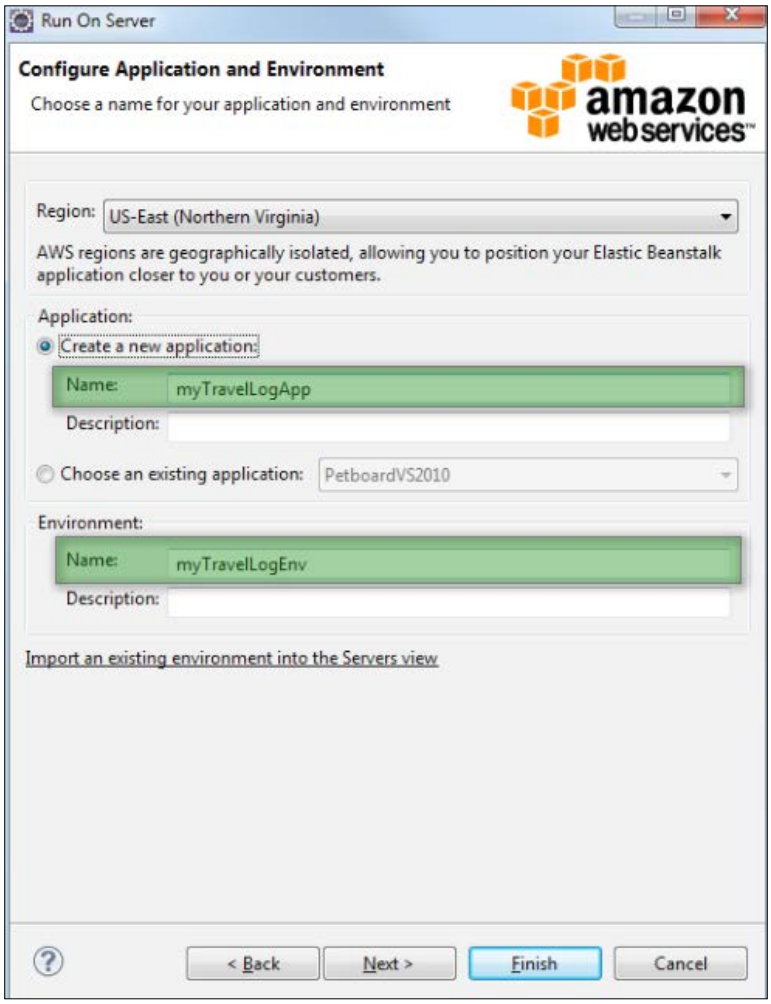
The **AWS Java Web Project** dialog box will permit you to select the region in which your web application will be deployed and run. Here, I am going to choose **US East (N. Virginia)** as it is the cheapest one in my case:



1. In **Project Explorer**, right-click on the **myTravelLog** application and navigate to **Run As | Run on Server**. You can use the sample code which is provided to run and test your deployment.
2. In the **Run on Server** dialog box, click on **Manually define a new server** and choose **AWS Elastic Beanstalk** for **Tomcat 7** from the list of specified server preferences.
3. Enter a name, such as `TravelLogServer`, into the **Server's hostname** box or give a name of your choice.
4. Finally, select **Always use this server when running this project** and click on **Next**.

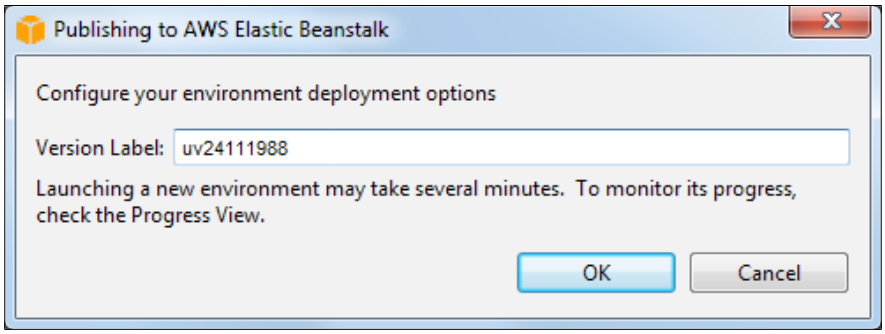


5. In the **Run On Server** box, set the application name as `myTravelLogApp` and the environment name as `myTravelLogEnv` for your reference. This is shown in the following screenshot:
6. Click on **Next**.

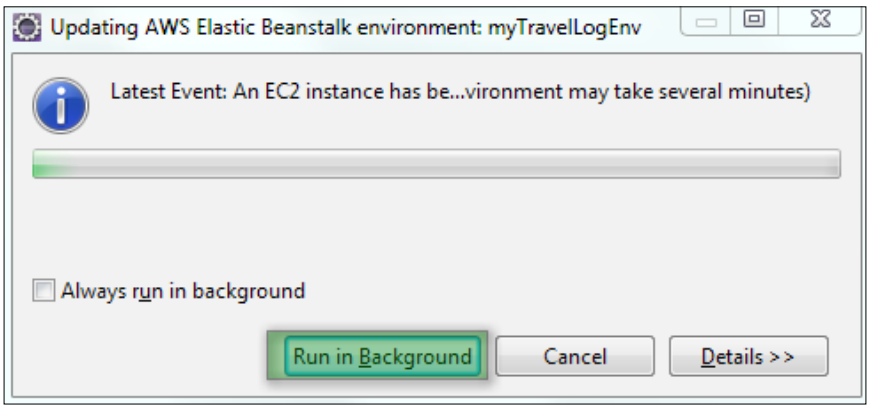


7. Go to **Run On Server** | **Advanced configuration** to stipulate further parameters for your web application deployment reference, such as IAM role, CNAME, notification email address, and so on.

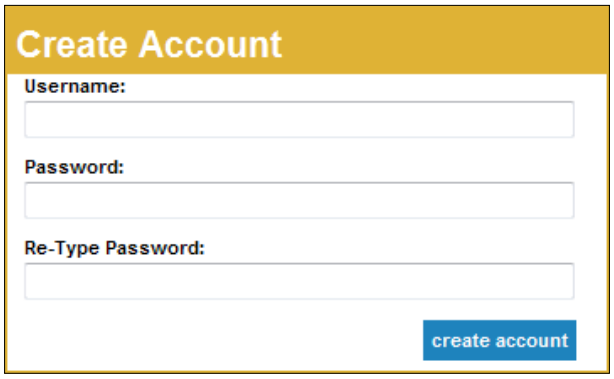
8. Before deploying your application to the AWS beanstalk container, the Toolkit will display a dialog box in which you can set a **Version Label** to give your deployment a version number. The AWS Toolkit will generate a distinctive version label based on the present time.
9. Click on **OK**.



10. While your application is deploying, the AWS Toolkit will show a progress status, as shown in the following screenshot:



When the deployment is complete, you will be able to see the next screen. This is the user interface for the "Travel Log" application, which will be running on your Amazon EC2 instance. But you don't need to bother about your EC2 instance, as you have deployed your application from Elastic Beanstalk.



So, the preceding example was with Elastic Beanstalk deployment using AWS SDK Toolkit. Now, let's move to another awesome service from Amazon for data warehouse.

Getting started with Amazon Redshift

Redshift came with AWS with the preliminary screening beta release in November 2012 and the full version was made available for acquisition on February 15, 2013. We will look more at the how and why of Redshift's performance. It has to do with how the data is being stored in a columnar data store and the work that has been done to reduce the complexity of computation.

Amazon Redshift can be considered a traditional data warehouse platform, but unlike traditional data warehouse platforms, Amazon Redshift is elastic and scalable. There are savings on the hardware side and on some of the human resources essential to run both the hardware and large-scale databases locally. Don't be under the impression that all management and maintenance tasks are taken away just by moving data to a hosted platform; it is still your job to complete? The hardware, software patching, and disk management (all of which are no lesser tasks) have been taken on by AWS.

Disk management, principally the programmed recovery from disk failure, and even the capability to instigate querying a cluster that is being restored (even before it is completed) are all prevailing and convincing things AWS has done to condense your workload and having intense up-time.

I am sure that by now you are speculating, why the name *Redshift*? If you predicted that it is with relation to the term from stargazing and the effort that Edwin Hubble ensured to describe the association of the astronomical phenomenon recognized as Redshift and the growth of our universe, you would have predicted correctly. The capability to do online resizing of your cluster as your data incessantly expands makes Redshift a very suitable name for this expertise.

Configuration options

There are two types of nodes you can select from when producing your cluster. The simple configuration of the huge Redshift (dw.hs1.xlarge) node is as follows:

- CPU with two virtual cores (Intel Xeon E5)
- 15 GB RAM
- Storage holds three HDD with 2 TB of storage
- Network type will be moderated
- Disk I/O will be moderated

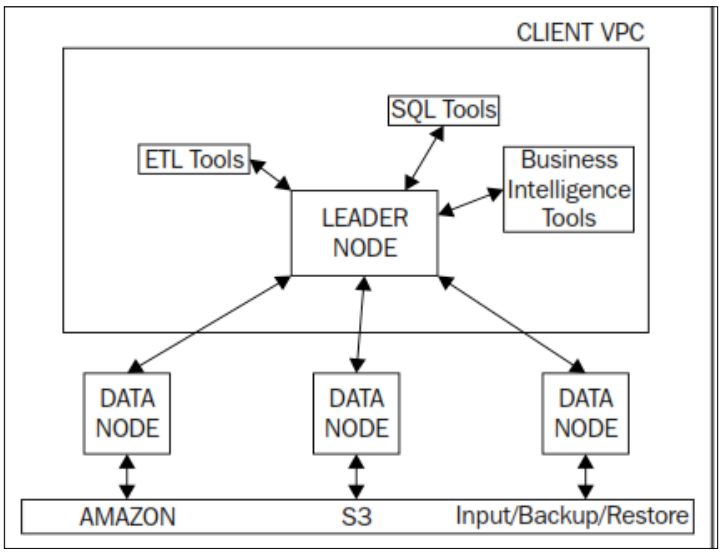
The elementary configuration of the extra-large Redshift (`dw.hs1.8xlarge`) node is as follows:

- CPU with 16 Virtual Cores (Intel Xeon E5)
- 120 GB RAM
- Storage holds 24 HDD with 16 TB storage
- Network with 10 GB Ethernet
- Disk I/O performance will be very high

The `hs` in the identification is the title AWS has used for great density storage.

An imperative point to note is that if you are involved in a single-node formation, the lone possibility you have is the smaller of the two possibilities. The 8XL extra-large nodes are only available in a multi-node configuration. We will look at how data is coming in the nodes and why multiple nodes are significant, in a later chapter. For fabrication purposes, we should have at least two nodes. There are performance motives as well as data protection explanations for this that we will discuss later.

The large node cluster provisions up to 64 nodes for an overall capability of anything around 2 to 128 TB of storage. The extra-large node cluster provisions from 2 to 100 nodes for a total volume of whatever between 32 TB and 1.6 PB. For the perseverance of conversation, a multi-node configuration with two large instances would have 4 TB of storage available and, consequently, would also have 4 TB of accompanying backup space. Before we get too ahead of ourselves; a node is a single host containing of one of the preceding configurations. When I talk about a cluster, it is a group of one or more nodes that are running together, as shown in the following figure.



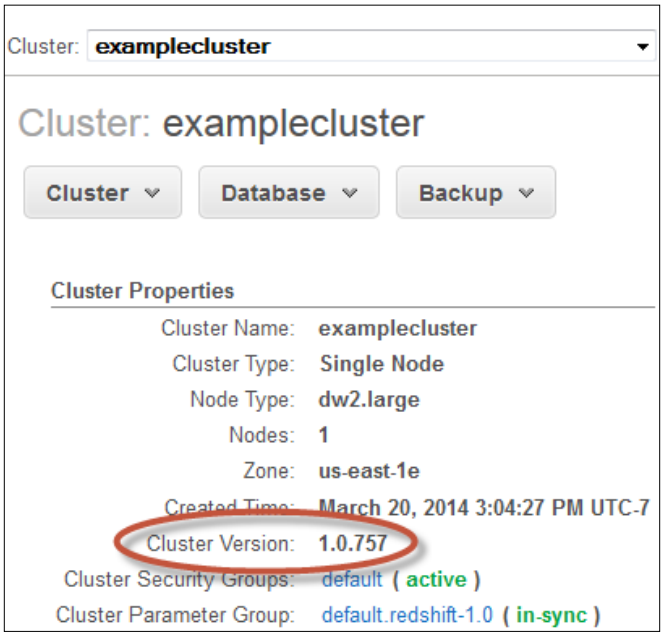
It's time now to get into some of the particulars and specifics to get up and running. As with most of the AWS products you have used in the past, there are just a few introductory things to take care of. Although the keys of your AWS account are not definitive to Redshift, be sure to hang on to both your public and secret key values from your AWS account. Those keys will be labeled "Access Key" and "Secret Key". You can view the Access Key public slice from the user security credentials on the **Security Credentials** tab. Once you have created your private key and secret key, the procedure to make the cluster is a console-driven step that you can start from the Amazon Redshift Management Console.

Cluster configurations

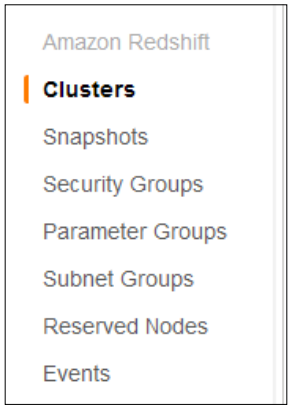
You can see that for most access requests for data you have to request via your transaction on Amazon Redshift, the evasion mode is one which provides this property. Creating objects will necessitate conceding permissions as well as granting permissions to use cluster management. Depending on the environment that you are approaching from, this may be annoying from time to time; nevertheless, bearing in mind the fact that you are tenuously hosting your data, I for one am glad with the superfluous steps required to access data. The prominence of data security, as a wide-ranging statement, cannot be excessive. You are answerable for your establishment's data as well as its image and reputation. The point that data has been wrongly retrieved has nothing to do with the position of the data (remote or local) if you use Amazon or some other benefactor, but rather it is dependent on the rules that have been set up to permit access to the data. Do not take your security group's formation lightly. Only open access to the resources or systems you really require and can maintain strict database rules on access.

To be honest, this must be something you are already doing (regardless of where your data is physically located). However, if you are not, take this as the chance to impose the required security to safeguard your data. You will have to supplement your IP ranges to permit access from the systems that you will be using to access your cluster. In short, you should augment the EC2 security group that contains the EC2 instances (if there are any) that you will be connecting from, as shown in the following screenshot. You will also require a parameter group. A parameter group relates to each database inside the cluster, so the decisions you choose reflect as global settings. If there are resources that you would like to regulate in these settings, you should to create your own parameter group. The creation of the new group may be done before you create your cluster. If you don't want to alter the default values, feel free to just use the parameter group that is already created. You can define the Amazon Redshift service and database versions for cluster in the **Cluster Version** field of the console. The first two sectional parts of the number are the cluster version, and the last part is the precise revision number of the database in the cluster.

In the following example, the cluster version is **1.0** and the database revision number is **757**. It can be vary based on versioning and latest editions of apps and databases.



In the following screenshot, you can see some of the high-level management functions related to backups, security groups, and so on.



You will essentially need to reflect as you bring your data and procedures to the Redshift environment, so you can start discovering it in a more precise way.

Interacting with AWS Trail

AWS CloudTrail is a web service that records AWS API calls for the AWS account and conveys log files to you. The recorded information contains the individuality of the API caller, the time of the API call, the root IP address of the API caller, the given parameters, and the reaction essentials returned by the AWS service.

Features and benefits

To start with AWS CloudTrail, let's have a look at following features and benefits:

Increased visibility: CloudTrail offers improved visibility of user action by recording AWS API calls.

Durable and inexpensive log file storage: CloudTrail employs Amazon S3 for log file storage and deliverance; consequently, log files are stored indestructibly and at an affordable price.

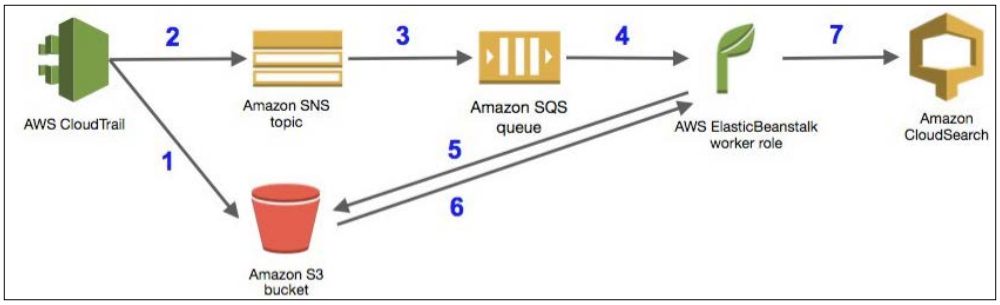
Easy administration: CloudTrail is an entirely managed provision; you can turn on CloudTrail for the account using the AWS Management Console, the CLI, or the SDK and start receiving CloudTrail log files in the Amazon S3 bucket that you state.

Timely delivery: CloudTrail characteristically carries events within 15 minutes of the API call.

Log file aggregation: CloudTrail can be cumulatively configured to log files across manifold accounts and regions so that log files are distributed to a single bucket.

You have gone through the basic overview of the AWS CloudTrail, so it's time for some real time example.

You can influence numerous AWS offerings to achieve a straightforward, scalable, and robust architecture to index AWS CloudTrail logs in AWS CloudSearch. You can start by configuring CloudTrail to transport SNS notification as soon as a new log file becomes available. Every notification is positioned into an Amazon SQS queue and taken by a simple AWS Elastic Beanstalk application with the worker role. An application will retrieve each file from the S3 bucket, extract logs, and add each log to a CloudSearch domain.



To start with AWS CloudTrail, follow the steps given here:

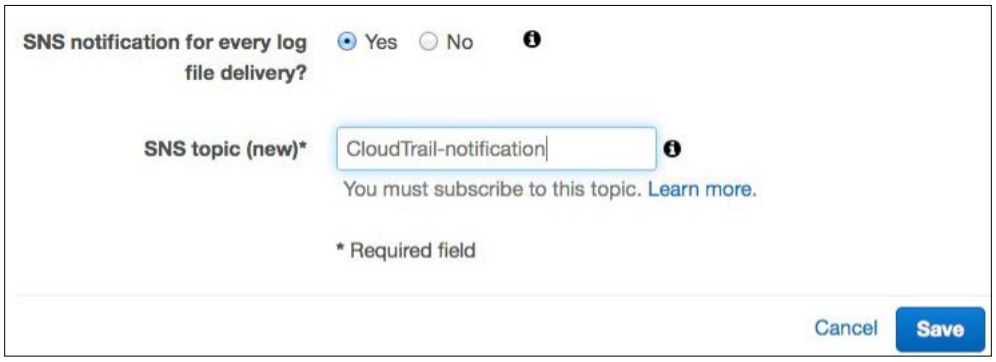
1. Go to the CloudTrail console and click on **Get Started**
2. Select **Yes** for **Create a new S3 bucket** and enter the desired bucket name.
3. Click on **Save**. CloudTrail will create bucket for you and will set the required policies automatically.

You need to activate CloudTrail for each AWS region.

4. Click on **Advanced** and create an SNS notification in the CloudTrail console.
5. For **SNS topic (new)**, give a name like `CloudTrail-notification`. Ensure that the **SNS notification for every log file delivery?** option is set to **Yes**.

If you are working on CloudTrail in multiple AWS regions, you should generate at least one SNS topic per region.

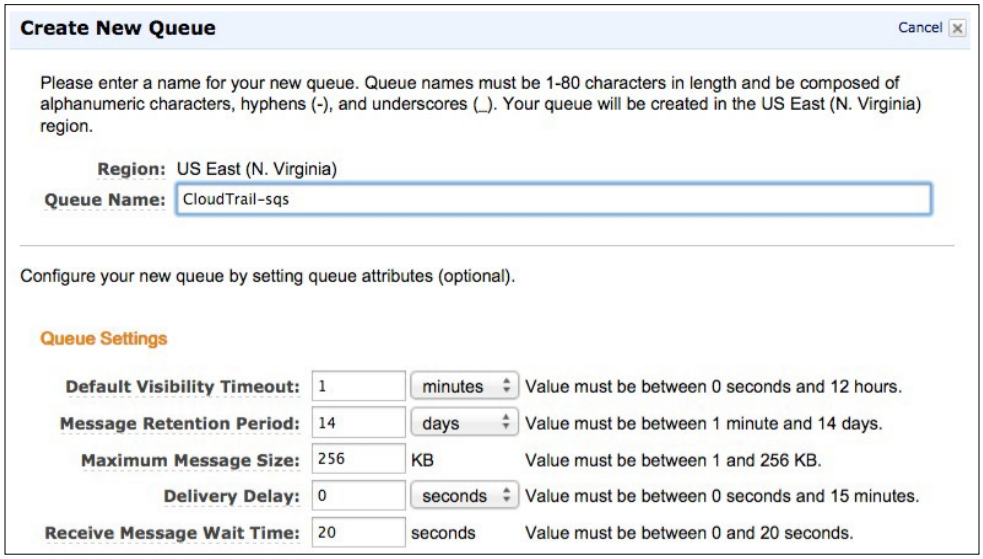
6. Follow the diagram here for reference on how to create new SNS topic:



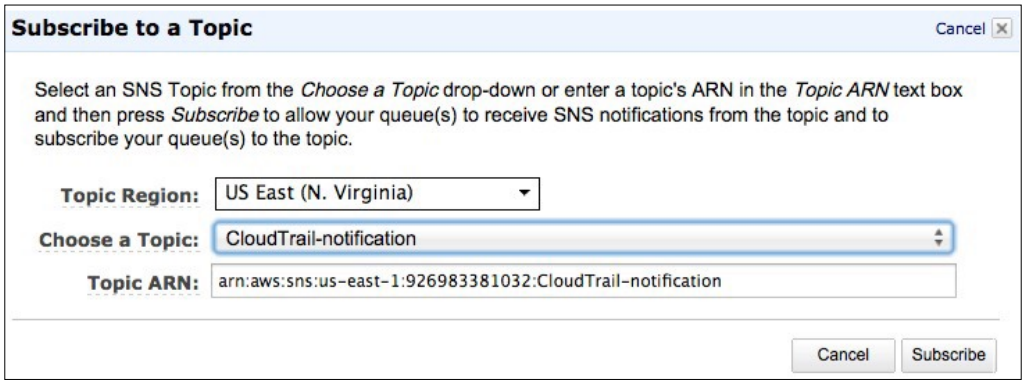
Once you are done with SNS topic creation, you have to create the AWS CloudSearch domain. Amazon CloudSearch creates it directly and it is inexpensive to set up, handle, and scale a common search solution for your application. It's simple to produce and configure a CloudSearch domain from the AWS Management Console. You can even create CloudSearch domain from the script. For that, you have to setup AWS CLI first, and then you can execute the given script as a code here. The script will take around 10 minutes to complete and your domain(s) will be ready. The script will create the domain and configure a default domain named `cloudtrail-1` which will be created in the "us-east-1" region.

Now, you have to create AWS SQS for queuing notifications. By integrating Amazon SNS with Amazon SQS, all the notifications delivered are recorded in an Amazon SQS queue where they are processed by an Elastic Beanstalk app that indexes these logs in CloudSearch. To create a queue from the AWS Management Console, follow these steps:

1. In the AWS SQS console, click on **Create New Queue** and specify the following parameters:
    - **Queue Name**: `CloudTrail-sqs`
    - **Default Visibility Timeout**: `1` minute
    - **Message Retention Period**: `14` days (maximum)
    - **Receive Message Wait Time**: `20` seconds
2. Finally, click on **Create Queue**.



3. Next, click on **Queue Actions**.
4. Click on **Subscribe Queue to an SNS Topic**.
5. To select a topic, select the SNS topic that you created in the CloudTrail console.
6. Click on **Subscribe**. The AWS Console will set up the required security policies without human intervention.



7. After some time, you should see that messages are starting to reach your destination (SQS queue).

Now, it's time to set up a single-file app written in Python using the Flask framework, for example, you will create the AWS Elastic Beanstalk worker role mode. A worker is basically an HTTP request handler so that Beanstalk deals with messages buffered via AWS SQS. Messages put in the queue will be forwarded via HTTP POST to a configurable URL on the AWS Elastic Beanstalk hosted app.

As the app wants to subject AWS API calls to Amazon S3 and AWS CloudSearch, you will also use an IAM role for EC2 to permit the app to construct secure API requests from your instances without supervising the security credentials that the app uses. Let's initially create the required role in the console. The steps to create an IAM role are as follows:

1. Go to the IAM console.
2. Click on **Roles** in the navigation pane.
3. Click on **Create New Role**. Provide the role name, such as `cloudsearch-index`, and click on **Next Step**.

4. Then, click on the **Select** button for Amazon EC2 under **AWS Services Roles**.
5. On the **Set Permissions** page, scroll down and click on **Custom Policy** and select **Select**.
6. Copy and paste the policy and give it a name.
7. Click on **Next Step**, and then click on **Create Role**.

This configuration results in a policy are shown in the following code. You can even check out `code_3632EN_08_01.txt` in the code bundle:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "cloudtrailworkerrole",
      "Effect": "Allow",
      "Action": [
        "cloudsearch:DescribeDomains",
        "cloudsearch:ListDomainNames",
        "cloudsearch:document",
        "s3:GetObject",
        "s3:ListBucket",
        "sqs:ChangeMessageVisibility",
        "sqs:DeleteMessage",
        "sqs:ReceiveMessage",
        "cloudwatch:PutMetricData"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

Now, it's time to use the console to launch the AWS Elastic Beanstalk application. You can download the source code from the Packt Publishing website. The given sources file `.ebextensions/cloudtrail.config` holds the CloudSearch domain name and region details. You can alter this file or later just modify `PARAM1` and `PARAM2` which is in Elastic Beanstalk. Check out `3632EN_08_02.txt` from code bundle:

```
option_settings:
  "aws:elasticbeanstalk:application:environment":
    PARAM1: cloudtrail-1
    PARAM2: us-east-1
```

It is recommended to deploy the Beanstalk app in an EC2 VPC environment to get all the benefits of micro instances when you are using it first time, and it's good for the production environment also.

Now, go to the AWS Elastic Beanstalk console and click on **Create a New Application**. Deploying the Elastic Beanstalk is pretty simple. Here are the required parameters:

- **Environment tier**: `Worker`
- **Predefined configuration**: `Python`
- **Environment type**: `Load balancing, Auto Scaling`

To start AWS Elastic Beanstalk, follow the steps given here:

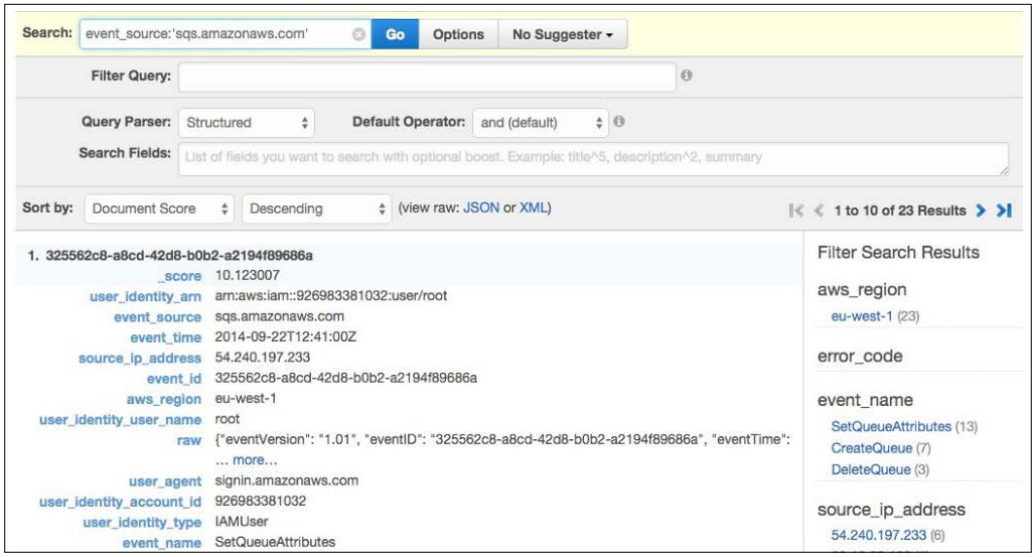
1. Click on **Browse** and upload the app `.zip` file you downloaded from the Packt Publishing website.
2. Click on **Next** and give a name to the **Environment**. The default name will be `cloudtrail1-env`.
3. Click on **Next** and select **Create this environment inside a VPC**.
4. Click on **Next**.
5. In the **Configuration Details** page, use following configuration:
    - **Instance type**: t2.micro
    - **Application health check URL**: /
    - **Instance profile**: cloudsearch-index
6. For `Worker` configurations, use following settings:
    - **Worker queue**: `CloudTrail-sqs`
    - **HTTP path**: `/sns/`
    - **MIME Type**: `application/json`
7. For **VPC security group**, you can use the default provided by AWS.

After some time, the app status will turn green and your configured CloudSearch domain will be populated.

After some time, AWS CloudTrail logs are directly "searchable" in AWS CloudSearch. You can use the AWS console to ask simple requests. In the AWS CloudSearch Console, pick your AWS CloudSearch domain and click on **Run a Test Search** in the navigation pane. From the drop-down menu, select the **Structured** query parser.

CloudSearch will display facets on the right-hand side column, which will assist you to explore the data.

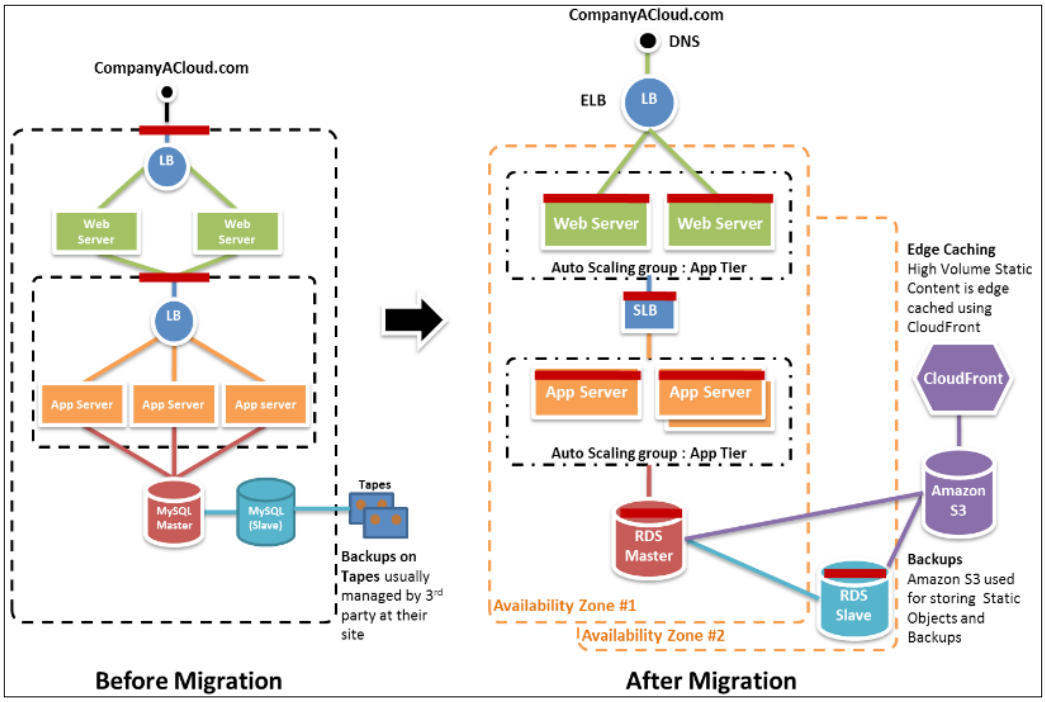
Here is a screenshot for your reference:



This example explains how you can generate a simple architecture designed to handle AWS CloudTrail logs as soon as they are produced and take them to real-time indexing tools like CloudSearch.

Case study: migrating applications to the Cloud

Now let's try to understand some migration scenarios for the Cloud. Consider the example of some company named "ComapnyACloud.com" that wants to shift its web application to the AWS Cloud. Here is the reference diagram that explains the architecture of the web application.



This company is a leading marketing product-based company which conveys the best marketing products to its customers over the globe. This company's application has a predictive spike for load on its architecture cyclically. The application is hosted at the company's datacenter now.

Application architecture

Using a three-tier architecture, the company is deploying a frontend load balancer, which is managing traffic across two web servers—each running on a separate box. The backend business logic is implemented in Java and uses Tomcat as app container and application server. On the database side, it consists of a master MySQL server and two slave servers for superior performance.

Motivation for migration

There are mainly three reasons for migration:

- To scale out application without using new hardware
- To cut down administration by auto deployment
- Do scaling for additional capacity when required

After the migration phase, you can user Route 53 service on top of ELB as DNS service.

Cloud assessment for migration

The team efforts that they could free up the present and existing infrastructure for further internal, ongoing, and forthcoming projects, or withdraw a maintenance treaty and reduce the operating cost by 40 percent. At the time of technical assessment, the technical team experienced that the app is compatible with Cloud using AWS EC2 with Linux instances.

Data Migration

During the migration phase, the technical team decided to transfer all the static content or part of the app (images, JS, CSS, video, audio, and static HTML content) to AWS S3 buckets. After that, they will align the bucket with AWS Cloud to give their users the content with low latency. Furthermore, they will transfer all their tape backups to S3 and Glacier. You can configure your database with RDS service and fetch data from the S3 bucket using any query tool or EC2 instance itself.

App migration

During the application migration phase, the development team launched both small and large instances for their web servers and Tomcat containers that hold app. The team has altered their build and deployment scripts to use the Cloud as an endpoint.

- While migrating an application, the company should be ready to change AWS-specific code as well
- The application logic needs to decouple so that the application can scale

The co-related infrastructure was not deployed instantly. The company has engaged a hybrid migration strategy for the migration. For a short term, the load balancer was routing traffic to the servers in the Cloud in as well as to the physical servers with its surviving infrastructure. After ensuring that the servers in the Cloud were presenting at the desired levels, the in-house servers were discharged one by one; the load balancers were updated on AWS, and all of the web traffic was being served up by the EC2 instances running in the Cloud. At last, once testing has been completed and the DNS has been switched to point to the Cloud-based web servers.

Leveraging the Cloud

Once the company has shifted its app to the Cloud, they started thinking about leveraging some of the progressive features of the AWS Cloud. So, they configured Auto Scaling by employing multiple AZs.

This way, the company was able to shift its existing app to the AWS Cloud. With least effort, the team was not only able to free up their physical hardware for other projects, but also diminished the operational expenditure by 30 percent.

Summary

In this chapter, you have learned about some of the helpful deployment and migration services. In the first section, you learned about AWS Beanstalk and its usages. Beanstalk is basically for those who don't want to worry about underlying resources for their applications. You can directly come with your code to Redshift and within a moment, your code will be deployed and running.

You also learned about application deployment using the AWS Elastic Beanstalk service via SDK and code libraries. We also discussed the basics of Redshift, how to create cluster of nodes, and how Redshift is helpful in Warehousing services. Next, you learned about AWS CloudTrail with a real-time example and AWS CloudWatch. Finally, we discussed a case study for app migration from in-house infrastructure to the AWS Cloud.

In the next chapter, you will learn how to bootstrap AWS EC2 instances with preconfigured commands for environment setup and how to use Chef for automation and deployment. Also, you will come to know that how the AWS CloudFormation service can work seamlessly with the application, and how SWF and OpsWorks service can be used with the AWS infrastructure.

9
Bootstrapping and Auto-configuration

Designing your instance deployment is like allowing your instances to ask you difficult questions about their existence at boot, for example, "Why am I created and what will be my role?" Each and every instance should have a specific part to play in the infrastructure and deployment environment, such as database servers, replica servers, web servers, cache server, and so on. These role features can be passed to the instance when you are booting or spinning up the instances from an AMI at runtime on air. At the time of booting the instance, we will download and configure necessary scripts and codes, as per the role requirements, and work automatically as defined in the scripts. In this chapter, we will cover following topics:

- Black belt booting
- Bootstrapping instances with AWS CloudFormation
- Bootstrapping Amazon instances with Chef
- Continuous integration and deployment
- Workflow execution of Amazon SWF
- Working with AWS OpsWorks

Here are the advantages of bootstrapping your instances:

- Reconstruct the development, test, and production environment with little snaps and nominal struggle
- Added control over your abstract Cloud-based capitals
- Reduce human-induced deployment mistakes
- Generate a self-healing and self-discoverable working environment that can withstand hardware failure

There are a number of advanced methods that give further supremacy and elasticity when bootstrapping AWS EC2 instances. For example, various organizations preserve a progression of generic instances and customize the AMIs upon launch. Widespread techniques are the following:

- Without human intervention, check for updates on each boot
- Look in a known place, such as in a S3 bucket, for data or a script to inform the instance which packages have to be loaded
- Pass user data to the instance to achieve each one of these, or perhaps as a substitute of the other advances

Black belt booting

There are a number of cutting-edge technologies that compromise on supplementary power and flexibility when booting Linux occurrences. For example, some official DOMs preserve sequences of standard instances and modify the imaginings upon promotion. Some common practices are as follows:

- Spontaneously check for updates upon every boot
- Look in a well-known place, such as in a **Simple Storage Service** (**S3**) bucket, for data or a script to tell the instance to load packages
- Permit user data to the instance to achieve each of the preceding goals, or perhaps as an alternative of the other approaches.

Before getting started with user data in instances, let's try to remember how to launch instances first. Instances are launched from AMIs. As soon as they are launched, they enter the pending state. An instance's hardware for the host computer is determined by the instance type chosen at launch time. The instance gets booted (where the billing starts) via the AMI chosen at launch time. It enters the running state as soon as it is ready. Billing continues as long as the instance is running.

Instances can be launched using multiple methods:

- Via the Amazon EC2 console with a chosen AMI
- Via the Amazon EC2 console with the created Amazon EBS snapshot
- From a backup of instance called AMI

- Via the Amazon EC2 console with an AMI that you purchased from the AWS Marketplace
- Launching an AWS Marketplace instance
- Via AWS CLI with a chosen AMI
- Using Amazon EC2 through the AWS CLI
- Via Amazon EC2 CLI with a chosen AMI
- Launching an instance using the Amazon EC2 CLI
- Via AWS tools for Windows PowerShell with a chosen AMI

When creating a new Linux EC2 instance on AWS, you're able to pass extra data to the server to be used during the boot progression. This can be a straightforward bash script or the URL(s) of a number of scripts. Alternatively, the data can be made trouble-free using a list of parameters that you require that instance to be familiar with.

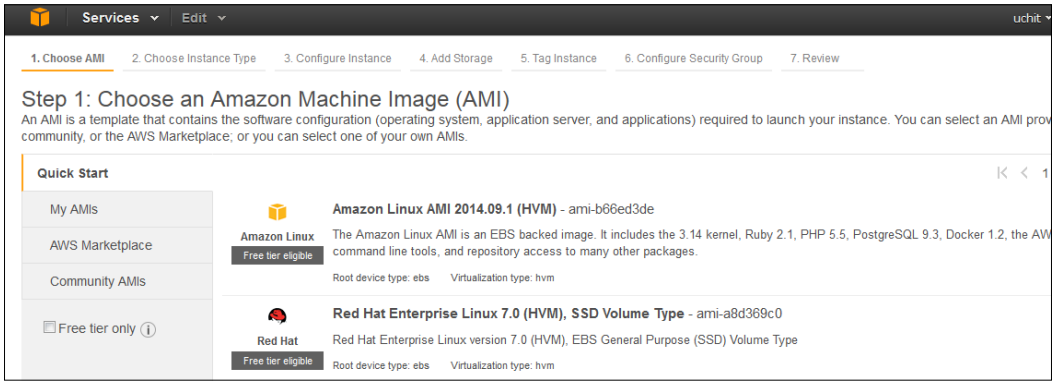
Now, if you write a script or the URL for a script as user data to your new instance, then it will run it you and you don't need to worry about it. On the other hand, if you pass it a catalog of parameters (for example, you may provide it the endpoint URI for your RDS instance so the database settings for your web application can be set up appropriately), you have to know how to read the user data. Along with reading the user data, you can also read in the metadata about your new instance, such as its AMI-ID or the public hostname of your instance. Reading the metadata can be done by querying a straightforward API.

The base URI of all your queries can be `http://172.254.160.254/`. This can be followed by the API version, or you can utilize the latest to obtain the most recent API. Then, the query includes the category of data you desire to retrieve (for example, metadata), followed by the category you desire to query. For example, the URI you would use to acquire the local hostname of your instance would be like this: `http://172.254.160.254/latest/meta-data/local-hostname`.

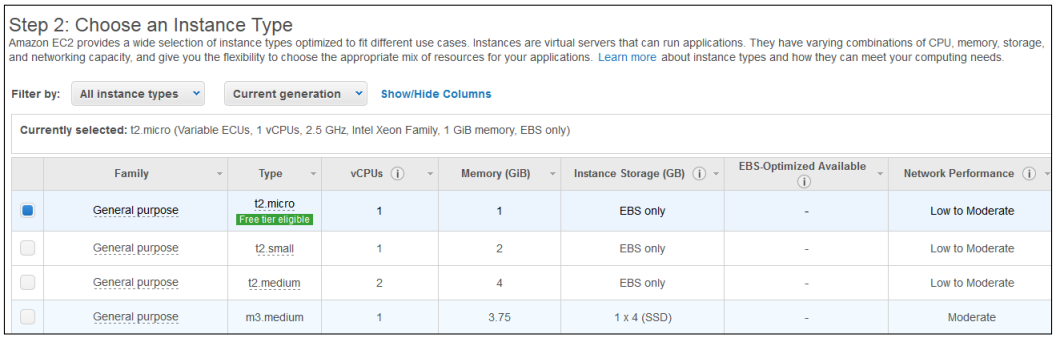
Even as the API is intended so that only your instance can interpret your user data, the requests are not encrypted. So, it is a good idea to avoid storing passwords or encryption keys using this technique. If a hacker was intelligent enough to expand admission to your EC2 instance via a non-honored user, they would still be able to read all the user data along with the supplementary metadata for your instance.

Let's see an example of how your user data script will work in an EC2 instance launch:

1. Log in to your AWS EC2 Management Console via a browser and click on the **Launch Instance** button. You will be redirected to the AMI selection page, as shown here:

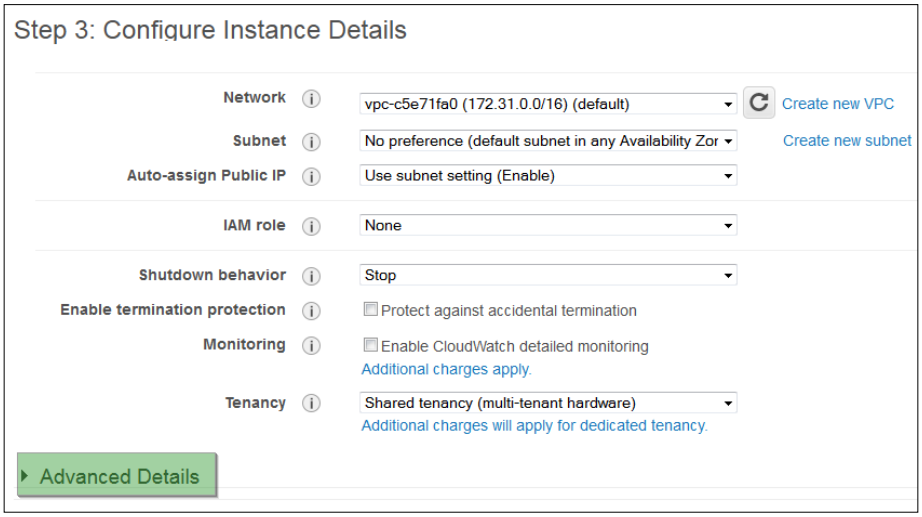


2. Select any Linux AMI. You will be redirected to the instance type selection page.

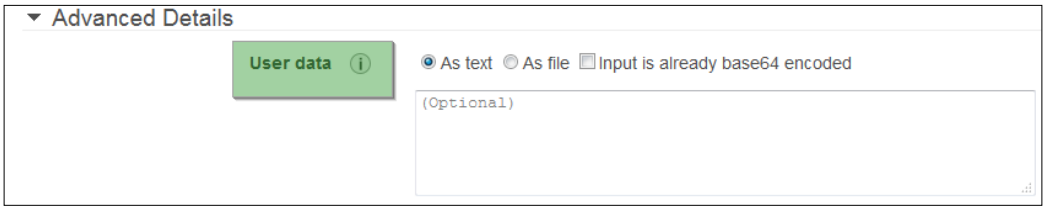


Instance type selection page

3. After selecting the instance type, you will be redirected to the instance configuration page. At the bottom of that page, there is one option called **Advanced Details**.



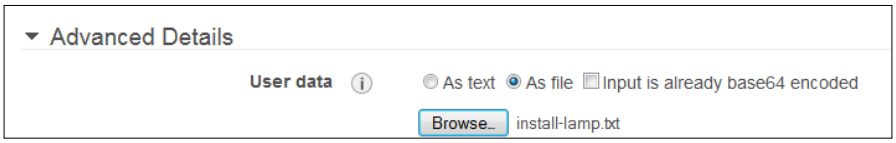
4. Click on **Advanced Details** and you will see the **User data** box where you can write your own scripts, commands, or URI:



5. Select the **As file** option. Create one file at your local machine called `install-lamp.txt` and append the following content in it:

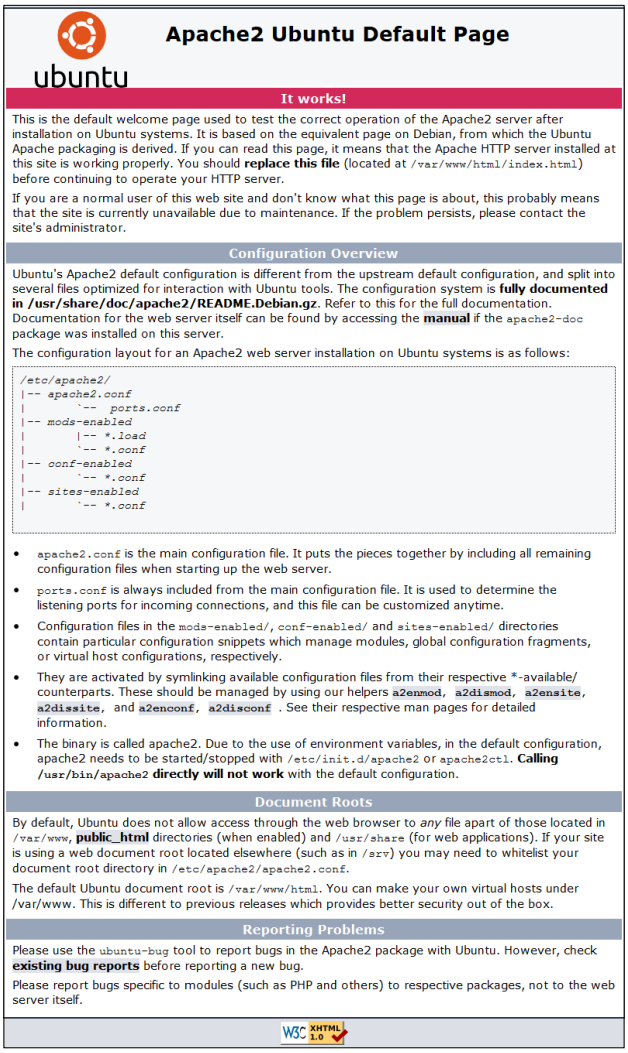
```
#!/bin/bash
set -x
export DEBIAN_FRONTEND=noninteractive
apt-get update && apt-get upgrade -y
tasksel install lamp-server
echo "Please remember to set the MySQL root password!"
```

6. Finally, the result will look like this on your EC2 console:



7. Launch your instance as you did in previous chapters.

   Once you are done with that, hit the URL in browser to get the Apache home page.



8. You can check for the Apache service even from CLI:

```
ubuntu@ip-172-31-39-224:~$ service apache2 status
 * apache2 is running
ubuntu@ip-172-31-39-224:~$
```

9. Even after logging in to the host successfully using SSH, you can check other services for LAMP stack. To check the MySQL service, use the following syntax:

   `mysql -u root`

   You will get the following output, which means MySQL is working properly:

```
ubuntu@ip-172-31-39-224:~$ mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 42
Server version: 5.5.40-0ubuntu0.14.04.1 (Ubuntu)

Copyright (c) 2000, 2014, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

10. Check whether PHP is installed properly or not using the next command:

    `php -version`

    You will get the following output for the PHP version:

```
ubuntu@ip-172-31-39-224:~$ php --version
PHP 5.5.9-1ubuntu4.5 (cli) (built: Oct 29 2014 11:59:10)
Copyright (c) 1997-2014 The PHP Group
Zend Engine v2.5.0, Copyright (c) 1998-2014 Zend Technologies
    with Zend OPcache v7.0.3, Copyright (c) 1999-2014, by Zend Technologies
ubuntu@ip-172-31-39-224:~$
```

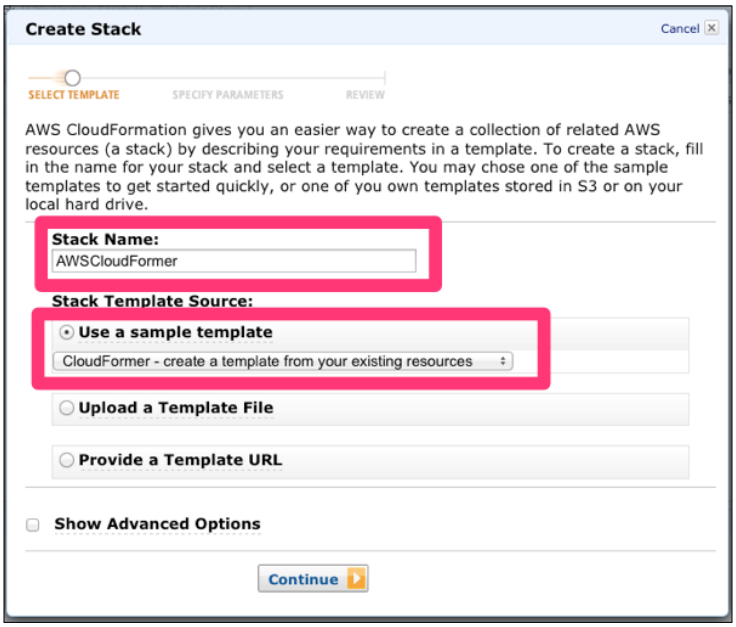
This way, you can use user data for dynamic configuration of your EC2 instance. You can check or debug for error when it fails by checking `/var/log/cloud-init.log`.

Bootstrapping instances with AWS CloudFormation

Amazon CloudFormation is a method to initiate environments easily. When you begin a CloudFormation environment, you will be able to launch precise AMIs with specific key pairs on predefined instance sizes and behind your AWS load balancers. If any segment of your environment fails to launch, the environment rolls itself over, terminating all the segments along the way. You're going to use a tool that analyzes your running environment and creates a CloudFormation template for you.

The tool that you are about to use will give you a template that will be stored in Amazon S3. Create a bucket to save the template. You'll use CloudFormer, which is a prototype tool that is designed to help you build these templates. You'll be able to twist the template in order to eliminate any unrelated instances. The CloudFormer tool is intended to create a starting point for your template. Once created, you can customize the template in every manner. CloudFormation and CloudFormer are both accessible via the AWS Console. Follow the next steps to start with Amazon CloudFormation:

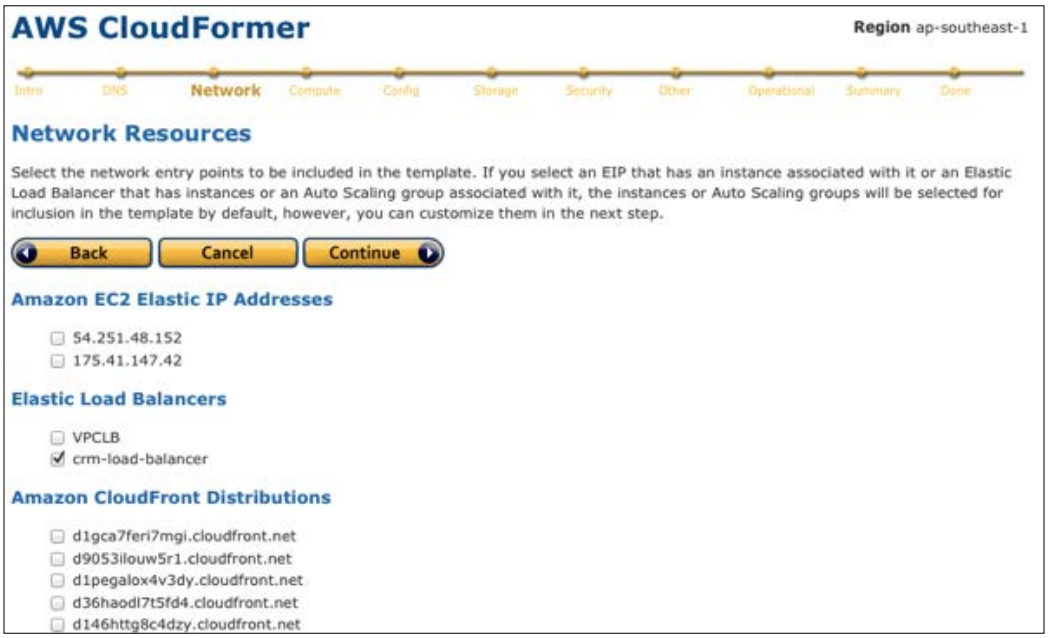
1. Select the **CloudFormation** service from the listed AWS services and click on the **Create New Stack** button.
2. Select the CloudFormer sample template from the listed templates and name it `AWSCloudFormer` before clicking on the **Continue** button:



3. Accept the terms and go to the **Review** page.
4. The console will display **CREATE_IN_PROGRESS** for a while. While it's getting completed, go to the **Outputs** tab and mark the URL address of the CloudFormer tool.
5. Now, try to connect to the running tool and start the wizard.

The wizard app provides no HTTPS and no authentication, so it can be used by any user anywhere in the world. Therefore, I recommend you to run through it as quickly as possible to complete the process.

6. Click on **Create Template** and add an explanation that classifies which template this is.
7. You'll be asked to select resources on a series of screens.
8. Select the load balancer that is associated with your environment, if any:



9. Choose your Auto Scaling group, if any:

10. Next, choose the launch configuration.
11. Select the **Security Group** that suits you and your environment.
12. Click on **Continue** for all the remaining steps.
13. The tool will generate a template and then suggest a S3 bucket to store it in.
14. Use the bucket that you fashioned at the beginning and click on the **Save Template** button.
15. You'll stop on the following screen, as this is the last one. Launch the stack now.

16. At this point, you can simply click on **Launch Stack** to test.

Bootstrapping Amazon instances using Chef

Nowadays, the majority of the topmost Cloud computing players present an easy-to-use UI to build your IT infrastructure on the Cloud. Nevertheless, unlike provisioning once on on-premise infrastructure, you may have to dynamically provision a number of **virtual machine** (**VM**) instances, a small number of instances of dynamic storage and several SaaS-based services. Software releases require to be pushed on a regular basis (weekly, daily, or hourly in various cases).

A way to do away with this is to create VM images for all changes and produce a new VM instance to push it. It will be difficult and it affects the frontend of the delivery. Also, what about the storage, databases, network configuration, and the underlying *architecture*? As your practice of Cloud infrastructure for development, QA, staging, and production environments grow, it becomes an operational challenge to supervise the complete infrastructure. The following operational tasks can become a problem for a system administrator:

- Spawning new instances
- Configuring newly created instances with storage, services, firewall, and preconfigured app/software
- Integrating monitoring and management services and removing instances
- Ensuring all (working) available instances in a layer (web/app) are in identical state

This is when you need a configuration management system that essentially provides capabilities to deploy, update, and refurbish your entire app infrastructure using predefined and automated events. In an ideal world, you wish for automatic stipulation of the complete environment from the base to running industry services entirely from a predefined pattern, counting the network configuration.

Chef is an infrastructure automation framework that makes it easy to set up, configure, deploy, and supervise servers and apps to all environments (physical, virtual, Cloud, and hybrid).

With Chef, you can write the code for your infrastructure. These are called **recipes** and you can use the recipes to create more complex infrastructure. Once automated, you will be holding a blueprint for your working infrastructure with automation, which allows you to construct or reconstruct automatically in a few moments.

Apart from Chef, there are other tools that support Cloud environments, such as Puppet, Ansible, or Salt. AWS also provides management and deployment services such as OpsWorks, which is an app management service that makes it simple for DevOps to model and supervise the complete app from end user facing load balancers to backend databases. Amazon OpsWorks supports Chef as its backend to provision resources as per the demand or deployment scenarios.

Using Chef, you can perform the following operations:

- Deal with servers by writing recipes for their configuration and management
- Provide firm integration with apps, various databases, and more (such as network configuration and behavior)
- Create replica of development, QA, or preproduction environments

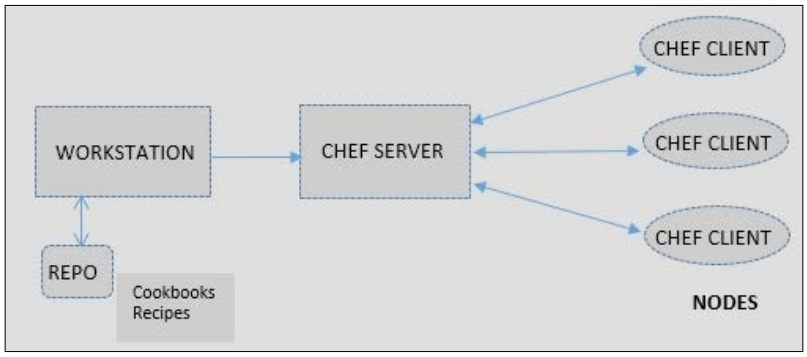
Before jumping into the sea of Chef, you should go through the terms used regularly in Chef. Chef consists of three main fundamentals: a server, one (or more) nodes, and at least one workstation.

The **server** works as a hub that is accessible to all nodes. All Chef client nodes should be registered using the Chef server. The server consists of all the cookbooks, recipes, roles, environment configuration, and policies.

The **workstation** is the development appliance from which configuration elements like cookbooks, recipes and policies are defined. Configuration essentials are corresponding with the chef-repo and uploaded to the server with the Knife command utility.

Nodes includes chef-client, which carries out all the infrastructure automation.

Have a look at the following flow diagram:



There are three types of Chef servers:

- **Hosted Chef**: This is an adaptation of the Chef server that is hosted by Chef. It is on Cloud with great scalability and can use services with resource-based access control. There is no need to run a supplementary server and supervise it.
- **Enterprise Chef**: This is similar to Hosted Chef, but the Chef server is situated on premise.
- **Open Source Chef**: This is a free version of the Chef server that you will be using in this chapter.

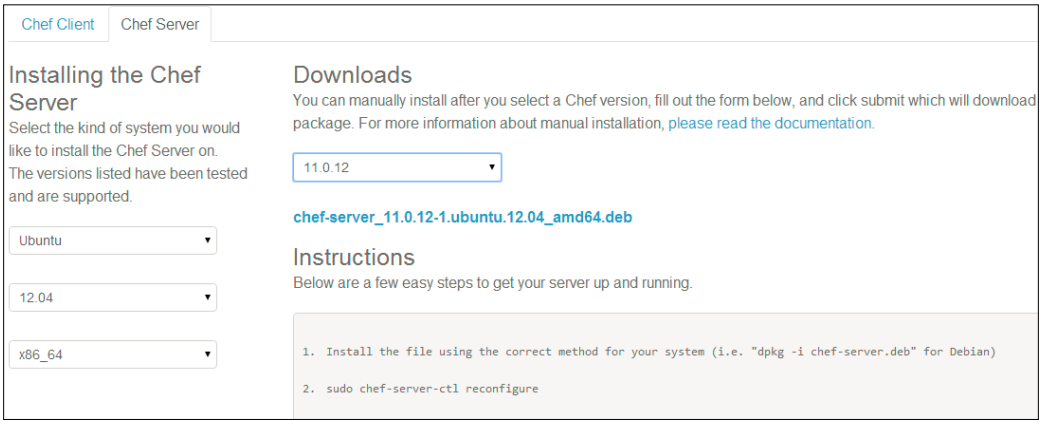
From all the possible ways of using Chef, setting up a personal Chef server is the least recommended approach but we will go through Open Source Chef to understand all the functionalities. Chef Solo or Chef Zero (`http://goo.gl/B7guas`) would be *so much easier* to run.

Common technical terms that are the fundamentals of Chef are as follows:

- **Recipe**: This is the configurable aspect within an organization of Chef. Recipes are used to set up, configure, and deploy applications in any given environment.
- **Cookbook**: This is the primary element of configuration and rule distribution. Every cookbook characterizes circumstances, such as the entire environment needed to set up and configure MongoDB.
- **Knife**: This is a command-line utility that offers an interface flanked by chef-repo and the Chef server.
- **chef-repo**: This is positioned on the workstation and includes cookbooks, recipes, and roles.
- **Workstation**: This is a system that is configured to execute Knife commands to coordinate with the chef-repo and cooperate with a single server.
- **Node**: This is any substantial, virtual, or Cloud system that is configured to be preserved by a chef-client.
- **run_list**: This is a prearranged catalog of roles and/or recipes that are executed in an accurate order.
- **chef-client**: This is an agent that executes locally on each node.

Now it's time to set up Chef on AWS to understand how Chef works with EC2. Follow the steps given here for installation and configuration (they are performed on Ubuntu):

1. Download the Chef server from `http://downloads.getchef.com/` and choose the suitable version, as shown in following screenshot:



2. Spin up a new EC2 instance that consists of Ubuntu OS from the AMI selection page in your AWS account and SSH to the server with the key file and username Ubuntu:

```
# switch to your home folder
cd ~
# Download the Chef Server Package using below command
wget https://opscode-omnibus-packages.s3.amazonaws.com/
ubuntu/12.04/x86_64/chef-server_11.0.10-1.ubuntu.12.04_amd64.deb
# Install the Chef Server using below command
sudo dpkg -i chef-server*
# reconfigure the service for your machine to configure all the
services of chef
sudo chef-server-ctl reconfigure
```

   After performing the preceding steps, you can access the web interface by typing `https://` from your browser. As the SSL certificate is signed by an authority not recognized by the browser, you will obtain a warning.

3. Click on the **Proceed anyway** button. Make sure that the port `443` is open in the security group allied with the server.

4. Log in with the given default admin credentials with username `admin` and password `p@ssw0rd`.
5. You will be asked to change the password after the first login.

You have set up the Chef server, and now it's time to set up the workstation. It can be on the same machine or on a different machine. Follow the next few steps to set up the workstation:

1. Install Git as Chef commonly uses it. You can use any other VCS tool too. The commands are as follows:

```
sudo apt-get update
sudo apt-get install git
# Download and run the client installation script from the Chef official website.
curl -L https://www.opscode.com/chef/install.sh | sudo bash
```

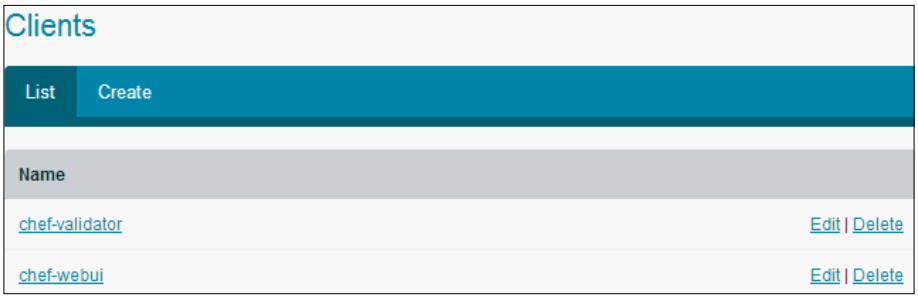
2. Clone the chef-repo skeleton directory using the following commands:

```
cd ~
git clone https://github.com/opscode/chef-repo.git
```

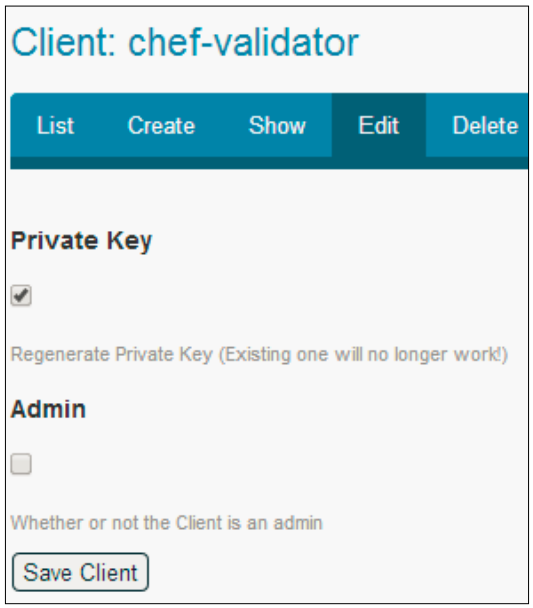
3. The preceding command will produce a directory called `chef-repo` in your home location. At this location, the complete configuration will be saved.
4. Make a `.chef` directory within `chef-repo` to keep the authentication and configuration files. The command is as follows:

   ```
   mkdir ~/chef-repo/.chef
   ```

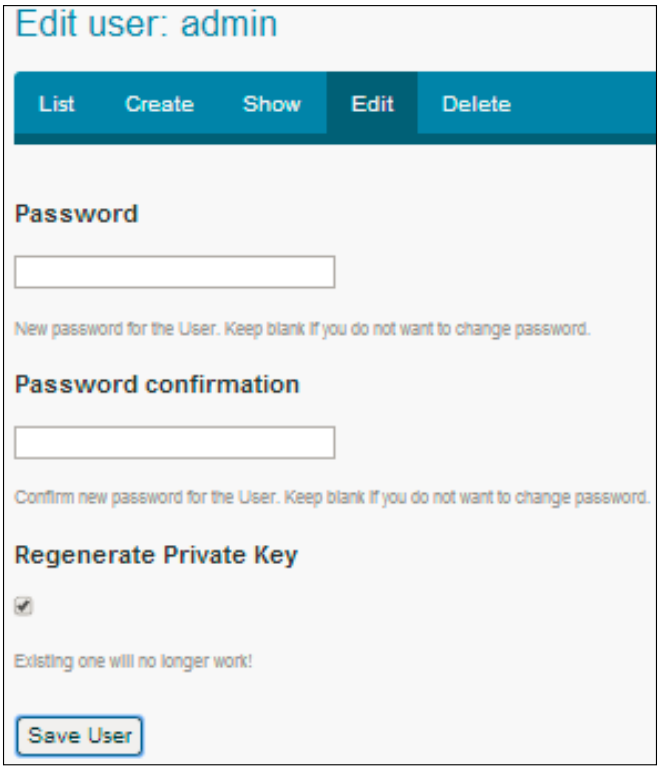
5. Log in to the Chef server with the admin identification and click on the **Clients** tab in the top navigation bar.



6. Click on the **Edit** button linked with the **chef-validator** client. Regenerate the private key by selecting that check box and clicking on **Save Client**.



7. Copy your newly generated private key and save it as the `chef-validator.pem` file in the `~/chef-repo/.chef` location.
8. In the same way, click on the **Users** tab in the navigation bar and click on the **Edit** hyperlink related to the **admin** user to regenerate the private key, as shown here:



9. Copy the private key and save it in the `admin.pem` file in the `~/chef-repo/.chef` location.
10. Now, it's time to configure the Knife tool that will provide an interface between a local repo and the Chef server. Use the following command to initialize the Knife command utility:

```
knife configure -initial
```

11. The previous command will prompt you for the path of the `.pem` files that you configured previously, server URL, username, and password.

```
ubuntu@ip-10-134-189-237:~$ knife configure --initial
WARNING: No knife configuration file found
Where should I put the config file? [/home/ubuntu/.chef/knife.rb] /home/ubuntu/chef-repo/.chef/knife.rb
Please enter the chef server URL: [https://ip-10-134-189-237.ap-southeast-1.compute.internal:443] https://122.248.204.10:443
Please enter a name for the new user: [ubuntu] santhosh
Please enter the existing admin name: [admin]
Please enter the location of the existing admin's private key: [/etc/chef-server/admin.pem] /home/ubuntu/chef-repo/.chef/admin.pem
Please enter the validation clientname: [chef-validator]
Please enter the location of the validation key: [/etc/chef-server/chef-validator.pem] /home/ubuntu/chef-repo/.chef/chef-validator.pem
Please enter the path to a chef repository (or leave blank): /home/ubuntu/chef-repo
Creating initial API user...
Please enter a password for the new user:
Created user[santhosh]
Configuration file written to /home/ubuntu/chef-repo/.chef/knife.rb
```

12. To make sure that everything is fine, run the following command to list all the registered users:

    ```
    knife user list
    ```

    It will show the following output with the registered users:

```
ubuntu@ip-10-134-189-237:~/chef-repo$ knife user list
admin
```

The bootstrapping process includes configuring a Chef client on a given node. The Chef client will talk with the Chef server to obtain directions for its individual configuration. After the client obtains the policy or let's say recipe, it will apply to the node to make sure the client is configured as per the given directions from the Chef server.

Knife-ec2 is an authorized Chef Knife plugin for EC2. This plugin will offer Knife the skill to create, bootstrap, and direct EC2 instances.

1. To install Knife-ec2 on your workstation, use the following commands one by one with sudo:

    ```
    apt-get install gcc g++ make autoconf


    apt-get install libxml2 libxml2-dev libxslt1-dev


    /opt/chef/embedded/bin/gem install nokogiri -v '1.5.2' -- --with-
    xml2-lib=/usr/lib/i386-linux-gnu --with-xml2-include=/usr/include/
    libxml2 --with-xslt-lib=/usr/lib/i386-linux-gnu --with-xslt-
    include=/usr/include/libxslt


    /opt/chef/embedded/bin/gem install knife-ec2 -V
    ```

2. The plugin will give the following subcommands. You can use the `--help` flag to discover definite command options, for example, look at the following commands:

```
knife ec2 server create
knife ec2 server delete
knife ec2 server list
```

3. On the workstation, generate or download the desired cookbooks and upload them to the Chef server. If you want a specific version, you can explicitly specify that. You can use following commands to download specific cookbooks that are available in the OpsCode repository:

```
cd ~/chef-repo/cookbooks/
#clone the repositories from Opscode github by cloning repos
git clone https://github.com/opscode-cookbooks/apt/
git clone https://github.com/socrata-cookbooks/java
git clone https://github.com/opscode-cookbooks/openssl
git clone https://github.com/opscode-cookbooks/tomcat

# upload the cookbooks to the chef server using below commands
knife cookbook upload java apt
knife cookbook upload openssl tomcat
```

Now, you have to specify how these cookbooks will be executed. For that, you have to create a role for it. A role is a method to describe definite patterns and processes that should be present at nodes in an organization for a single purpose. Each role consists of one or more attributes and a run list. Each node can have zero or multiple roles associated with it.

1. You will generate a `webapp` role and then utilize it to bootstrap an EC2 instance on AWS:

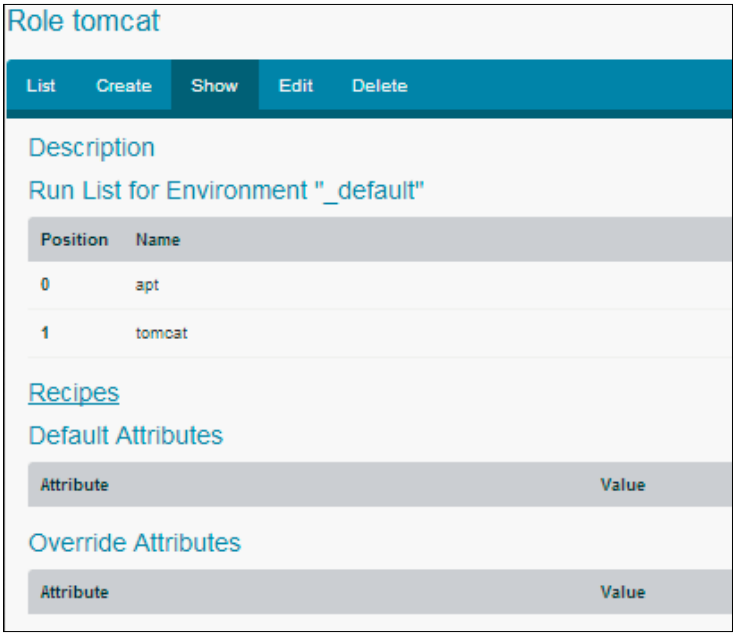
```
# Create webapp role
knife role create webapp
```

   The preceding command will open up the default editor (can be Notepad) and you can edit the file contents as shown here:

```
{
"name": "webapp",
 "description": "Install and configure Java and Tomcat",
"json_class": "Chef::Role",
"default_attributes": {  },
"chef_type": "role",
```

```
"run_list": [ "recipe[apt]",  "recipe[tomcat] ],
"env_run_lists": {   },
"override_attributes": {  }
}
```

2. Try to log in with the Chef server and navigate to the **Roles** tab; you should see the role you created:

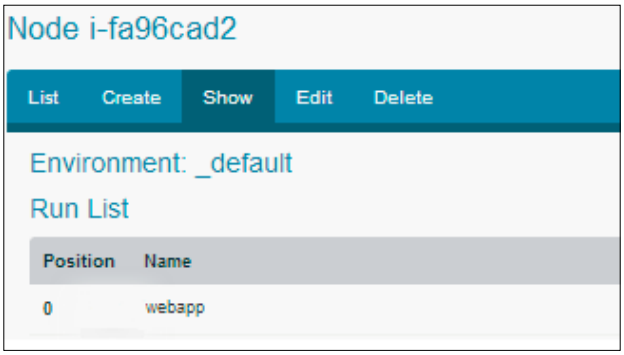


3. Now, you are ready to bootstrap an EC2 instance with the `webapp` role using the following command:

```
knife ec2 server create -I ami-3c39686e -r "role[webapp]" -Z ap-
southeast-1b --groups default -S chef -i demo1.pem -f m1.medium -A
'AKIXXXXXXXXXXXXXXXXX' -K "XXXXXXXXXXXXXXXXXXXXXX" --region ap-
southeast-1 --ssh-user Ubuntu
```

An `m1.medium` EC2 instance will be created in the `ap-southeast-1b` zone with the specified group and key name. The `webapp` role will be functional to the instance as well. When the `webapp` role will run in alignment with the node, the configuration information of that node will be compared adjacent to the attributes of the role, and then the contents of the role's run list are applied to the node's configuration details. When a chef-client will run, it merges its individual attributes and run lists with those enclosed within each assigned role.

4. On the Chef server, you can list the nodes and see the node with the `webapp` role.



In this way, you can use Chef to bootstrap EC2 instances.

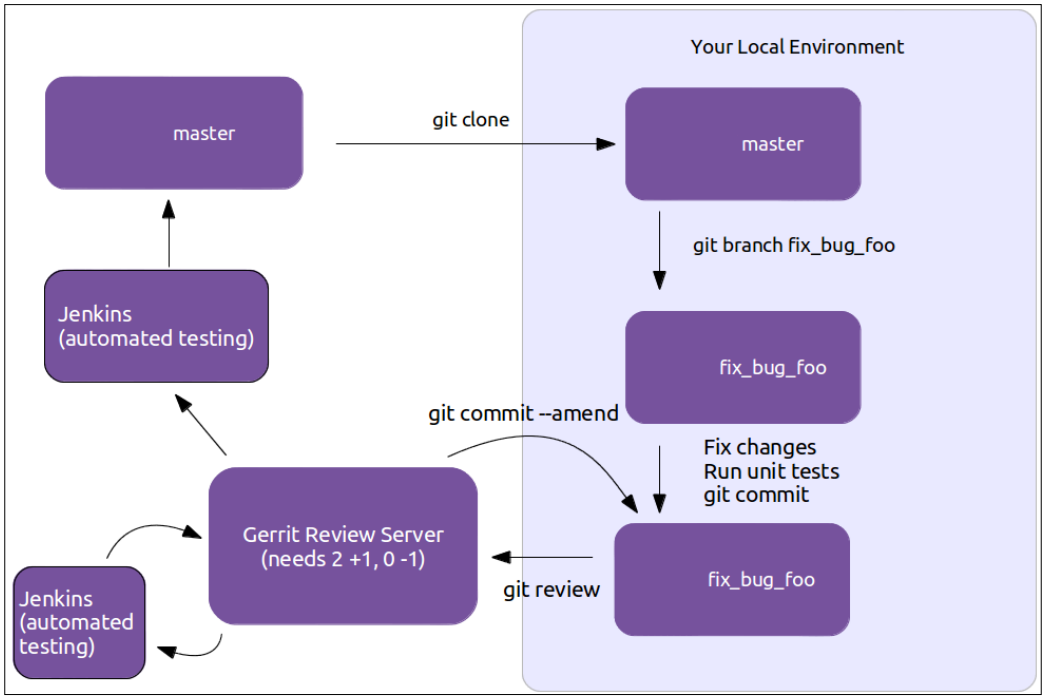
Continuous integration and deployment

Continuous integration (**CI**) is a development practice that requires developers to integrate code into a shared repository several times a day. Each checking is verified by an automated build, allowing them to identify problems early. By integrating early, you can detect errors quickly and locate them more easily.

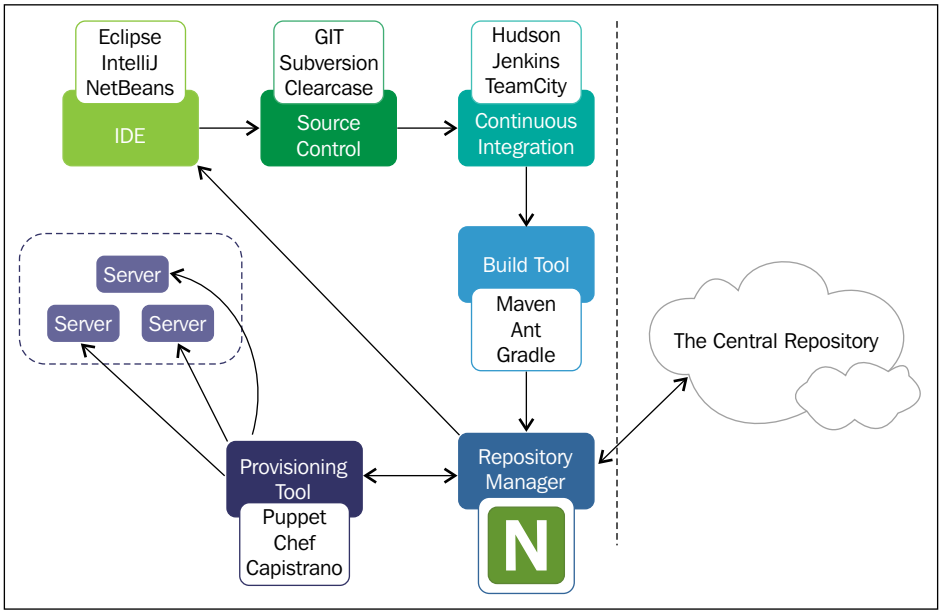
Based on your business requirements, you can select various tools for CI pipeline creation. For this chapter, I am describing the tools and flow for one of my clients' best working architecture. To understand the flow of CI, go though the following diagram and explanation that discusses the CI pipeline:

- **Jenkins**: This is an open source CI tool that supports the distributed build environment. When a developer commits the code, Jenkins is responsible for validating the build.
- **Promotional workflow**: This feature of Jenkins distinguishes good quality builds from bad quality builds. To put it simply, promotional build is the build that is selected for QA. If testing is successful, then it is passed to stage and production release. A user will have to log in to Jenkins to use promotional UI.
- **Gerrit**: This is a web-based code review system that facilitates code reviews for projects using the Git version control system.

- **Nexus**: This is an open source artifact repository management tool. When a build is successful and Jenkins passes it, the last step in each Jenkins job is to upload final artifact to Nexus.



Continuous delivery (**CD**) makes it possible to continuously adopt software in line with user feedback, shifts in the market, and changes to business strategy. Test, support, development, and operation work together as one delivery team to automate and streamline the build, test, and release processes. In following diagram, you can understand the flow of the process from IDE to source control, source control to Continuous Integration server, from Continuous Integration server to build tools, build tool to provisioning tool and provisioning tool to end server.



You can use the preceding tools to start as CI/CD best practices.

Automation with Amazon SWF

AWS **Simple Workflow** (**SWF**) from Amazon is an inimitable workflow as compared to conventional workflow products such as JBPM and OS Workflow. AWS SWF is tremendously scalable and engineer-gracious (flow is defined with Java code), while it comes with boundaries and lots of gotchas.

The very first thing to know is that it's almost impossible to build a SWF app properly without Flow Framework. Although the low-level SWF RESTful service API is public and available in SDK, for the majority workflow with parallelism, timer, or notification, think about the potential of how each event can interlace with a different event. It's easy to write down exact code with low-level API to wrap all use cases. For this reason, SWF is fairly distinctive in comparison with other thin-client AWS technologies.

There are basic concepts with respect to SWF that you need to understand to learn about the flow and workings of SWF:

- **Workflow starters**: This can be any app that can kick off workflow executions.
- **Activity workers**: This is a procedure or thread that carries out the activity jobs that are a fraction of the workflow. An activity task is one thread in the workflow. To exercise an activity task, you have to register it using the Amazon SWF console or the `RegisterActivityType` action.
- **Deciders**: This is an accomplishment of a workflow's synchronization logic. Deciders manage the flow of activity jobs in a workflow execution.

The workflow execution of Amazon SWF

To start with the flow, you have to follow the next few steps:

1. Write activity workers that employ the processing steps in the workflow. Write a decider to put into practice the harmonization logic of workflow.
2. Register activities and workflow with Amazon SWF. You can do this programmatically or through the AWS Management Console.
3. Initiate activity workers and a decider. These actors can start on any compute system that can contact an Amazon SWF endpoint. For example, you could use any compute service in the Cloud, such as Amazon EC2 or servers in your data center. The decider and activity workers will start polling Amazon SWF for tasks.
4. Initiate more than one executions of workflow. Executions can be initiated either programmatically or via the SWF Management Console. Each one runs separately and you can endow each one with its own set of input data. When an execution is in progress, Amazon SWF schedules the preliminary verdict task. In reaction, the decider commences generating decisions that embark on activity tasks. Execution persists in anticipation of your decider making a decision to shut the execution.

You can view the workflow execution from the AWS Management Console. You can pass through a filter and view absolute particulars of running over and above the completed executions.

1. To start with SWF programming, please go through the following code base to understand the logic and flow. The first file, `app.config`, which will define the region and profile to bring into play:

```
xml version="1.0"?>
<configuration>
    <appSettings>
       <add key="AWSProfileName" value="AWS PS Default"/>
       <add key="AWSRegion" value="us-east-1" />
    </appSettings>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
    </startup>
</configuration>
```

2. Another file called `main`, which does the setup initially, launches deciders and workers. It also initiates the workflow process.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using System.Linq;
using Amazon;
using Amazon.SimpleWorkflow;
using Amazon.SimpleWorkflow.Model;
```

3. In the following code snippet, `Activity1A` and `Activity1B` are serviced by one set of workers. `Activity2` is serviced by an additional set of workers.

```
static string domainName = "HelloWorldDomain";
static IAmazonSimpleWorkflow swfClient =
            AWSClientFactory.CreateAmazonSimpleWorkflowClient();

public static void Main(string[] args)
{
  string workflowName = "HelloWorld Workflow";

  // Setup
  RegisterDomain();
  RegisterActivity("Activity1A", "Activity1");
  RegisterActivity("Activity1B", "Activity1");
  RegisterActivity("Activity2", "Activity2");
  RegisterWorkflow(workflowName);

  // Launch workers to service Activity1A and Activity1B
  //  This is acheived by sharing same tasklist name (i.e.) "Activity1"
  Task.Run(() => Worker("Activity1"));
  Task.Run(() => Worker("Activity1"));

  // Launch Workers for Activity2
  Task.Run(() => Worker("Activity2"));
  Task.Run(() => Worker("Activity2"));

  // Start the Deciders, which defines the structure/flow of Workflow
  Task.Run(() => Decider());

  //Start the workflow
  Task.Run(() => StartWorkflow(workflowName));

  Console.Read();
}
```

Now, you have to write code to register your workflow and activities. Go through the following steps:

1. To register a domain, go through the following code:

```
static void RegisterDomain()
{
    // Register if the domain is not already registered.
    var listDomainRequest = new ListDomainsRequest()
    {
    RegistrationStatus = RegistrationStatus.REGISTERED
    };

    if (swfClient.ListDomains(listDomainRequest).DomainInfos.Infos.FirstOrDefault(
                                            x => x.Name == domainName) == null)
    {
    RegisterDomainRequest request = new RegisterDomainRequest()
    {
        Name = domainName,
        Description = "Hello World Demo",
        WorkflowExecutionRetentionPeriodInDays = "1"
    };

    Console.WriteLine("Setup: Created Domain - " + domainName);
    swfClient.RegisterDomain(request);
    }
}
```

2. To register your activity, use the following code snippet:

```
static void RegisterActivity (string name, string tasklistName)
{
    // Register activities if it is not already registered
    var listActivityRequest = new ListActivityTypesRequest()
    {
    Domain = domainName,
    Name = name,
    RegistrationStatus = RegistrationStatus.REGISTERED
    };

    if
(swfClient.ListActivityTypes(listActivityRequest).ActivityTypeInfos.TypeInfos.FirstOrDefault(
                                x => x.ActivityType.Version == "2.0") == null)
    {
    RegisterActivityTypeRequest request = new RegisterActivityTypeRequest()
    {
        Name = name,
        Domain = domainName,
        Description = "Hello World Activities",
        Version = "2.0",
        DefaultTaskList = new TaskList() { Name = tasklistName },//Worker poll based on this
        DefaultTaskScheduleToCloseTimeout = "300",
        DefaultTaskScheduleToStartTimeout = "150",
        DefaultTaskStartToCloseTimeout = "450",
        DefaultTaskHeartbeatTimeout = "NONE",
    };
    swfClient.RegisterActivityType(request);
    Console.WriteLine("Setup: Created Activity Name - " + request.Name);
    }
}
```

3. Finally, you have to register your workflow. To do that, follow the code snippet shown here:

```
static void RegisterWorkflow(string name)
{
    // Register workflow type if not already registered
    var listWorkflowRequest = new ListWorkflowTypesRequest()
    {
    Name = name,
    Domain = domainName,
    RegistrationStatus = RegistrationStatus.REGISTERED
    };
    if
(swfClient.ListWorkflowTypes(listWorkflowRequest).WorkflowTypeInfos.TypeInfos.FirstOrDefault
(
                                    x => x.WorkflowType.Version == "2.0") == null)
    {
    RegisterWorkflowTypeRequest request = new RegisterWorkflowTypeRequest()
    {
        DefaultChildPolicy = ChildPolicy.TERMINATE,
        DefaultExecutionStartToCloseTimeout = "300",
        DefaultTaskList = new TaskList()
        {
        Name = "HelloWorld" // Decider need to poll for this task
        },
        DefaultTaskStartToCloseTimeout = "150",
        Domain = domainName,
        Name = name,
        Version = "2.0"
    };

    swfClient.RegisterWorkflowType(request);

    Console.WriteLine("Setup: Registerd Workflow Name - " + request.Name);
    }
}
```

4. That's it for now to register your workflow and activities.

5. Now, you have to write a code to start your workflow. Go through the following code snippet to start the workflow.

```
static void StartWorkflow (string name)
{
    IAmazonSimpleWorkflow swfClient = AWSClientFactory.CreateAmazonSimpleWorkflowClient();
    string workflowID = "Hello World WorkflowID - " + DateTime.Now.Ticks.ToString();
    swfClient.StartWorkflowExecution(new StartWorkflowExecutionRequest()
    {
    Input = "{\"inputparam1\":\"value1\"}", // Serialize input to a string

    WorkflowId = workflowID,
    Domain = domainName,
    WorkflowType = new WorkflowType()
    {
        Name = name,
        Version = "2.0"
    }
    });
    Console.WriteLine("Setup: Workflow Instance created ID=" + workflowID);
}
```

6. To define your worker, you have to write the following code:

```
static void Worker(string tasklistName)
{
    string prefix = string.Format("Worker{0}:{1:x} ", tasklistName,
                        System.Threading.Thread.CurrentThread.ManagedThreadId);
    while (true)
    {
    Console.WriteLine(prefix + ": Polling for activity task ...");
    PollForActivityTaskRequest pollForActivityTaskRequest =
        new PollForActivityTaskRequest() {
                                    Domain = domainName,
                                    TaskList = new TaskList()
                                    {
                                        // Poll only the tasks assigned to me
                                    Name = tasklistName
                                    }
                                };
    PollForActivityTaskResponse pollForActivityTaskResponse =
                    swfClient.PollForActivityTask(pollForActivityTaskRequest);

    RespondActivityTaskCompletedRequest respondActivityTaskCompletedRequest =
                new RespondActivityTaskCompletedRequest() {
                            Result = "{\"activityResult1\":\"Result Value1\"}",
                            TaskToken = pollForActivityTaskResponse.ActivityTask.TaskToken
                        };
    if (pollForActivityTaskResponse.ActivityTask.ActivityId == null)
    {
        Console.WriteLine(prefix + ": NULL");
    }
    else
    {
        RespondActivityTaskCompletedResponse respondActivityTaskCompletedResponse =
            swfClient.RespondActivityTaskCompleted(respondActivityTaskCompletedRequest);
        Console.WriteLine(prefix + ": Activity task completed. ActivityId - " +
            pollForActivityTaskResponse.ActivityTask.ActivityId);
    }
    }
}
```

7. Finally, you have to define the decider and schedule your activity for execution flow. To schedule an activity, go through following code snippet:

```
static void ScheduleActivity(string name, List<Decision> decisions)
{
    Decision decision = new Decision()
    {
    DecisionType = DecisionType.ScheduleActivityTask,
    ScheduleActivityTaskDecisionAttributes =  // Uses DefaultTaskList
        new ScheduleActivityTaskDecisionAttributes() {
                    ActivityType = new ActivityType()
                    {
                    Name = name,
                    Version = "2.0"
                    },
                    ActivityId = name + "-" + System.Guid.NewGuid().ToString(),
                    Input = "{\"activityInput1\":\"value1\"}"
                }
    };
    Console.WriteLine("Decider: ActivityId=" +
                decision.ScheduleActivityTaskDecisionAttributes.ActivityId);
    decisions.Add(decision);
}
```

8. To initialize the decider, follow the code snippet shown here:

```
static void Decider()
{
    int activityCount = 0; // This refers to total number of activities per workflow
    IAmazonSimpleWorkflow swfClient = AWSClientFactory.CreateAmazonSimpleWorkflowClient();
    while (true)
    {
    Console.WriteLine("Decider: Polling for decision task ...");
    PollForDecisionTaskRequest request = new PollForDecisionTaskRequest()
                            {
                            Domain = domainName,
                            TaskList = new TaskList() {Name = "HelloWorld"}
                            };

    PollForDecisionTaskResponse response = swfClient.PollForDecisionTask(request);
    if (response.DecisionTask.TaskToken == null)
    {
        Console.WriteLine("Decider: NULL");
        continue;
    }

    int completedActivityTaskCount = 0, totalActivityTaskCount = 0;
    foreach (HistoryEvent e in response.DecisionTask.Events)
    {
        Console.WriteLine("Decider: EventType - " + e.EventType +
            ", EventId - " + e.EventId);
        if (e.EventType == "ActivityTaskCompleted")
        completedActivityTaskCount++;
        if (e.EventType.Value.StartsWith("Activity"))
        totalActivityTaskCount++;
    }
    Console.WriteLine(".... completedCount=" + completedActivityTaskCount);

    List<Decision> decisions = new List<Decision>();
    if (totalActivityTaskCount == 0) // Create this only at the begining
    {
        ScheduleActivity("Activity1A", decisions);
        ScheduleActivity("Activity1B", decisions);
        ScheduleActivity("Activity2", decisions);
        ScheduleActivity("Activity2", decisions);
        activityCount = 4;
    }
    else if (completedActivityTaskCount == activityCount)
    {
        Decision decision = new Decision()
        {
        DecisionType = DecisionType.CompleteWorkflowExecution,
        CompleteWorkflowExecutionDecisionAttributes =
            new CompleteWorkflowExecutionDecisionAttributes {
                        Result = "{\"Result\":\"WF Complete!\"}"
                    }
        };
        decisions.Add(decision);

        Console.WriteLine("Decider: WORKFLOW COMPLETE!!!!!!!!!!!!!!!!!!!!!!");
    }
    RespondDecisionTaskCompletedRequest respondDecisionTaskCompletedRequest =
        new RespondDecisionTaskCompletedRequest() {
                    Decisions = decisions,
                    TaskToken = response.DecisionTask.TaskToken
                };
    swfClient.RespondDecisionTaskCompleted(respondDecisionTaskCompletedRequest);
    }
}
```

9. Once you have run the previous snippets, you will get the following outputs on your CLI console.
10. Initially, domain and activity will be created and workflow will be registered:

```
Setup: Created Domain - HelloWorldDomain
Setup: Created Activity Name - Activity1A
Setup: Created Activity Name - Activity1B
Setup: Created Activity Name - Activity2
Setup: Registerd Workflow Name - HelloWorld Workflow
```

11. Then, polling for activity task will start:

```
WorkerActivity1:b : Polling for activity task ...
WorkerActivity1:c : Polling for activity task ...
WorkerActivity2:10 : Polling for activity task ...
WorkerActivity2:11 : Polling for activity task ...
```

12. Next, polling for the decision task will start:

```
Decider: Polling for decision task ...
Setup: Workflow Instance created ID=Hello World WorkflowID - 63537218519191930 8
Decider: EventType - WorkflowExecutionStarted, EventId - 1
Decider: EventType - DecisionTaskScheduled, EventId - 2
Decider: EventType - DecisionTaskStarted, EventId - 3
.... completedCount=0
Decider: ActivityId=Activity1A-175b5973-2915-40af-8da3-be970797d401
Decider: ActivityId=Activity1B-fe3eea16-e45e-42d0-b43c-d2bb105ed0c6
Decider: ActivityId=Activity2-77fca339-09b3-4cd8-ad64-86274de50d89
Decider: ActivityId=Activity2-71d972be-a835-41d0-afac-a1d722faa85d
Decider: Polling for decision task ...
WorkerActivity2:10 : Activity task completed. ActivityId - Activity2-77fca339-09b3-4cd8-
ad64-86274de50d89
WorkerActivity2:10 : Polling for activity task ...
WorkerActivity1:c : Activity task completed. ActivityId - Activity1A-175b5973-2915-40af-
8da3-be970797d401
WorkerActivity1:c : Polling for activity task ...
WorkerActivity1:b : Activity task completed. ActivityId - Activity1B-fe3eea16-e45e-42d0-
b43c-d2bb105ed0c6
WorkerActivity1:b : Polling for activity task ...
WorkerActivity2:11 : Activity task completed. ActivityId - Activity2-71d972be-a835-41d0-
afac-a1d722faa85d
WorkerActivity2:11 : Polling for activity task ...
Decider: EventType - WorkflowExecutionStarted, EventId - 1
Decider: EventType - DecisionTaskScheduled, EventId - 2
Decider: EventType - DecisionTaskStarted, EventId - 3
Decider: EventType - DecisionTaskCompleted, EventId - 4
Decider: EventType - ActivityTaskScheduled, EventId - 5
Decider: EventType - ActivityTaskScheduled, EventId - 6
Decider: EventType - ActivityTaskScheduled, EventId - 7
Decider: EventType - ActivityTaskScheduled, EventId - 8
Decider: EventType - ActivityTaskStarted, EventId - 9
Decider: EventType - ActivityTaskStarted, EventId - 10
Decider: EventType - ActivityTaskStarted, EventId - 11
Decider: EventType - ActivityTaskStarted, EventId - 12
Decider: EventType - ActivityTaskCompleted, EventId - 13
Decider: EventType - DecisionTaskScheduled, EventId - 14
Decider: EventType - ActivityTaskCompleted, EventId - 15
Decider: EventType - ActivityTaskCompleted, EventId - 16
Decider: EventType - ActivityTaskCompleted, EventId - 17
Decider: EventType - DecisionTaskStarted, EventId - 18
.... completedCount=4
Decider: WORKFLOW COMPLETE!!!!!!!!!!!!!!!!!!!!!
Decider: Polling for decision task ...
```

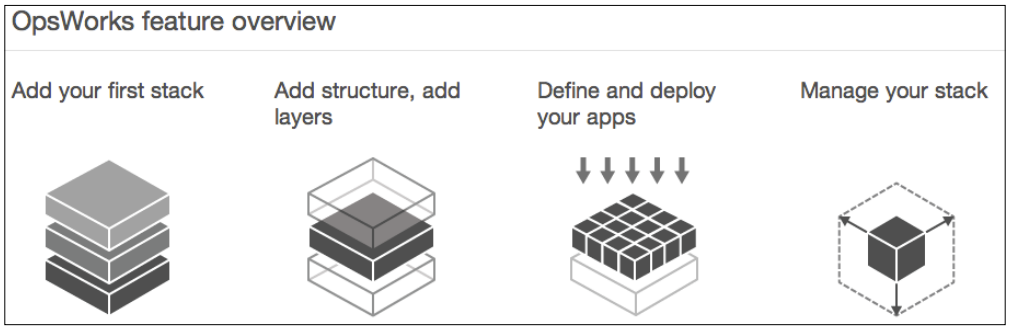
You can check this on your AWS SWF console for a better understanding of the output. You can download the whole code from the Packt Publishing website to run the preceding example code.

Working with AWS OpsWorks

AWS OpsWorks is a new service from Amazon that present high-level tools to handle your EC2-based deployment. AWS OpsWorks has some new terms and key theories:

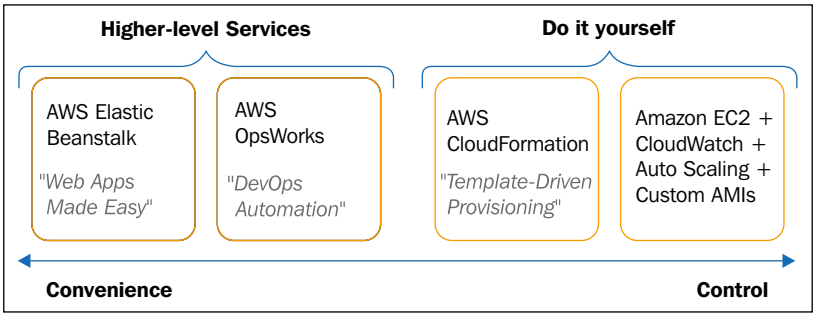
- A **stack** is known as the highest level container. It combines custom configuration with one or more apps. To deal with a simple to-do list site, you'd produce a *todo* stack, even though you might prefer to have split up *todo-production* and *todo-staging* stacks.
- Each stack has at least one **layer**. Consider these as classifications for different server roles. A simple static website might have a solitary Nginx layer. A typical web app might have a software load balancer layer, a Rails layer, and a DBMS called MySQL layer as substitutes. AWS OpsWorks describes ample number of incorporated layers (for example, Rails, HAProxy, PHP, Node, Memcached, MySQL, and more), but you can also classify your individual layer.
- **Apps** are your code base and are sourced from any SCM repository, an S3 bucket, or even an exterior web. A classic Rails site might have a single app working, but you can characterize various apps if you'd akin to configure, extent, and monitor them collectively.
- Finally, you characterize EC2 **instances** and allocate each to one or more configured layers. You can initiate instances yourself, or configure them to initiate and stop on a schedule or in retort to load patterns.

The following diagram shows the AWS OpsWorks fundamentals:



OpsWorks proposes further elasticity and control power, permitting you to adapt the types of servers you make use of and the layers or services that build up your app.

By focusing on the widely used AWS services, AWS EC2 instance types, and architectures, it can present superior automation and additional robust tools for configuration, authorization, scaling, management and monitoring. Even Amazon CTO, Dr. Werner Vogels, described it as follows:



Let's try to understand AWS OpsWorks using an example to get a deeper insight into its workings. After this example, you should be able to deploy "Todo Sample App" to AWS using OpsWorks, with your app and database running on various system instances.

The following are some prerequisites that should be fulfilled before proceeding further:

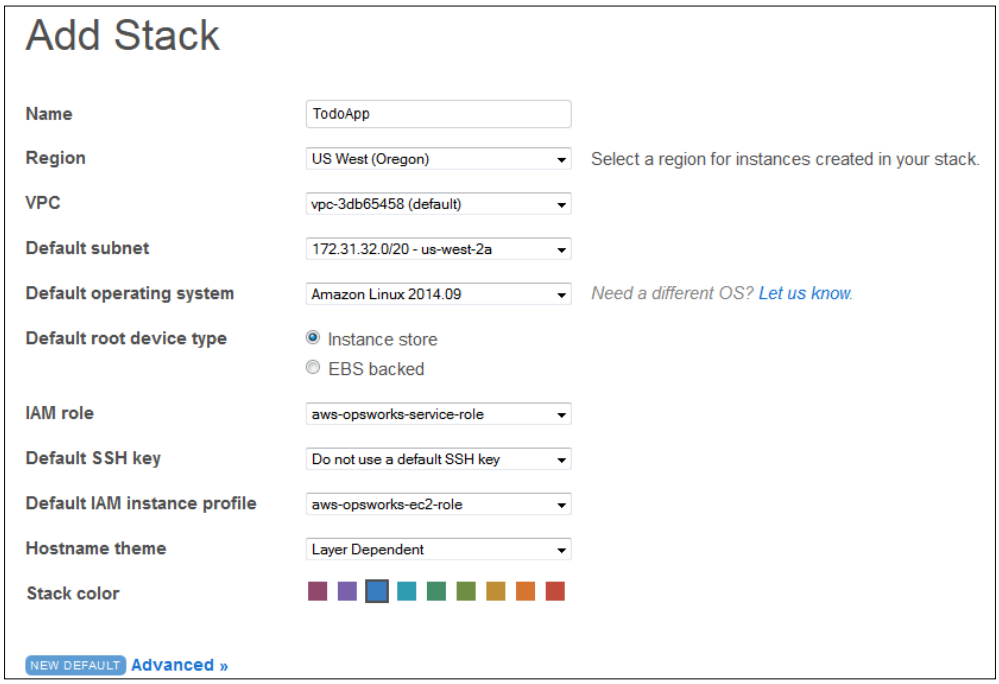
- Provide the `RAILS_SECRET_TOKEN` environment variable in the `config/secrets.yaml` file for your app servers.
- The `mysql2` gem is required to interface with a MySQL database.
- The `unicorn` gem is required to use Unicorn as our app server.

Let's start with the following example implementation now.

Creating an OpsWorks stack

To create an OpsWorks stack, follow the steps given here:

1. Log in to the AWS and navigate to the **AWS OpsWorks Console**.
2. Click on **Add Stack** and supply the required information:



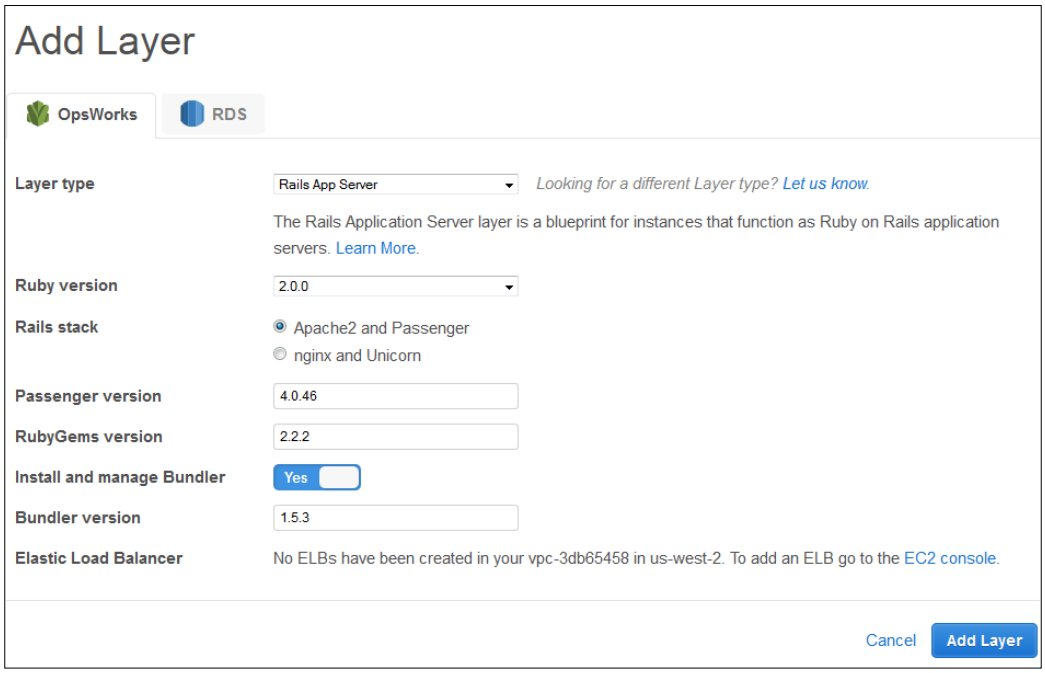
3. Once you've filled out the form, click on **Create Stack**.

You are done with stack creation.

Creating the Rails App Server layer

To create the Rails App Server layer, perform the given steps:

1. To start, click on **Add a layer**. You will be taken to the following page. Configure the required details of version and proceed further.

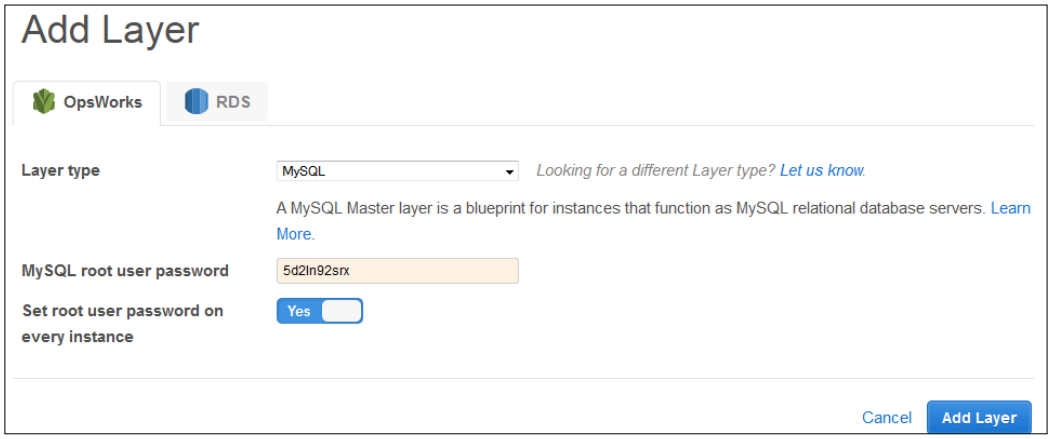


2. Once you're all done, click on **Add Layer**. You'll be redirected to the next page called **Layers**.
3. It's time to add the database layer.

Creating the database layer

To add the database layer, follow the given steps:

1. On the **Layers** screen, click on **+ Layer**.
2. Choose **MySQL** from the drop-down menu:

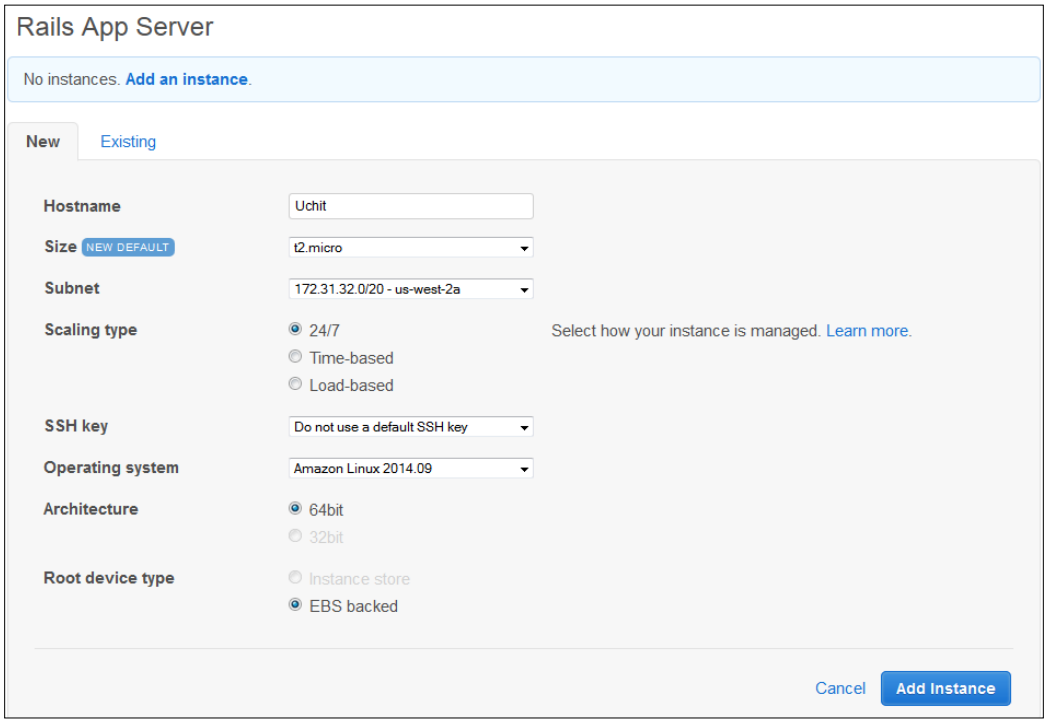


For this demo example, we are going to utilize a MySQL layer. You could replace it with an RDS layer if you want to.

Adding instances

You've completed layers for your app servers and database machines, but you do not have app servers or a database up and running yet. Next, we have to create an instance of each:

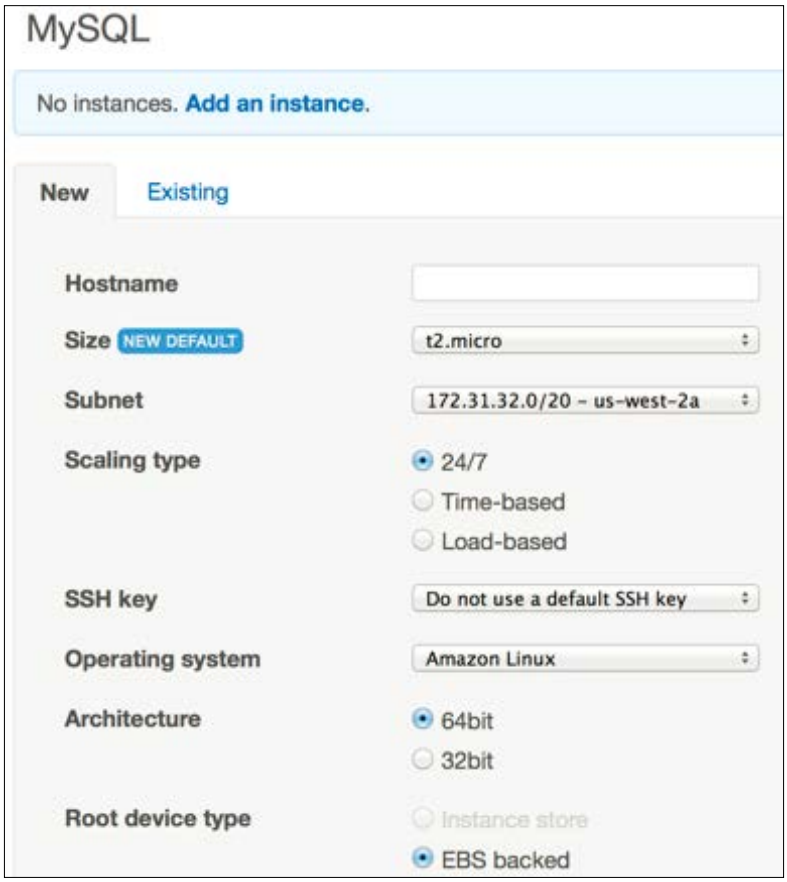
1. From the **Layers** part, click on **Add instance** in the Rails App Server layer.



Deploy App page

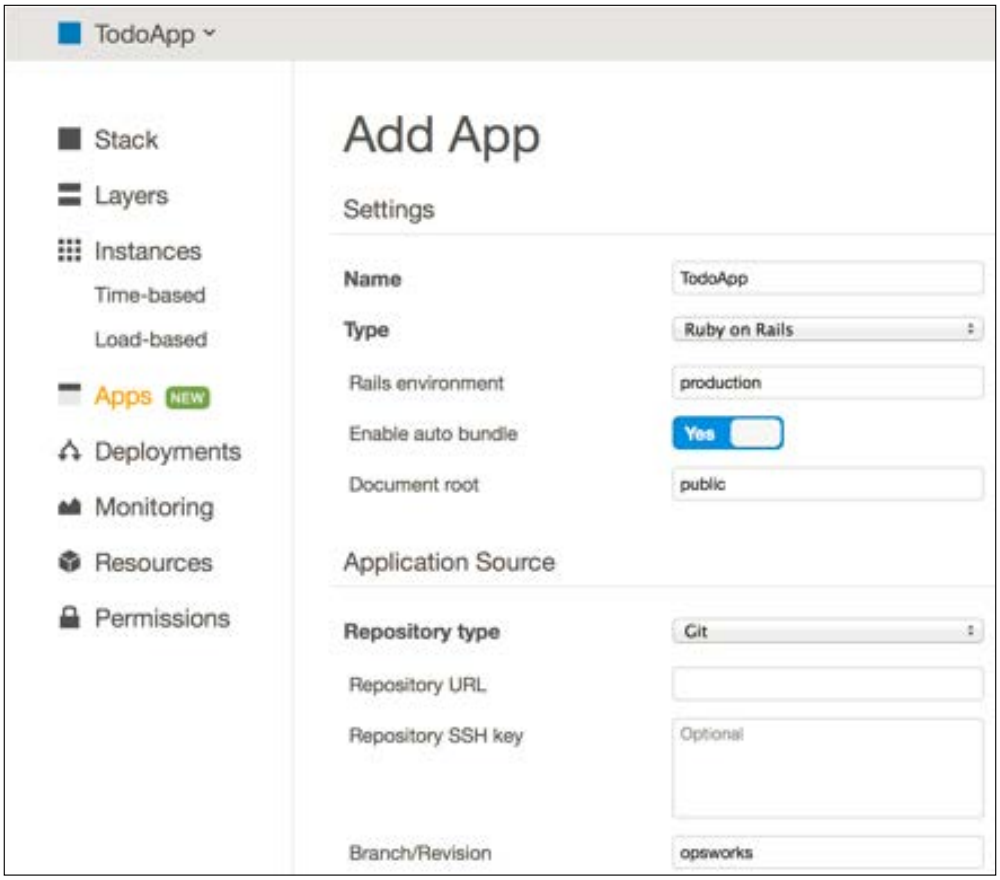
2. Click on **Add Instance** once you're done, and then click on **Start** to start the instance setup progression.

3. From here, or from the **Layers** page, click on **Add an instance** under **MySQL**.



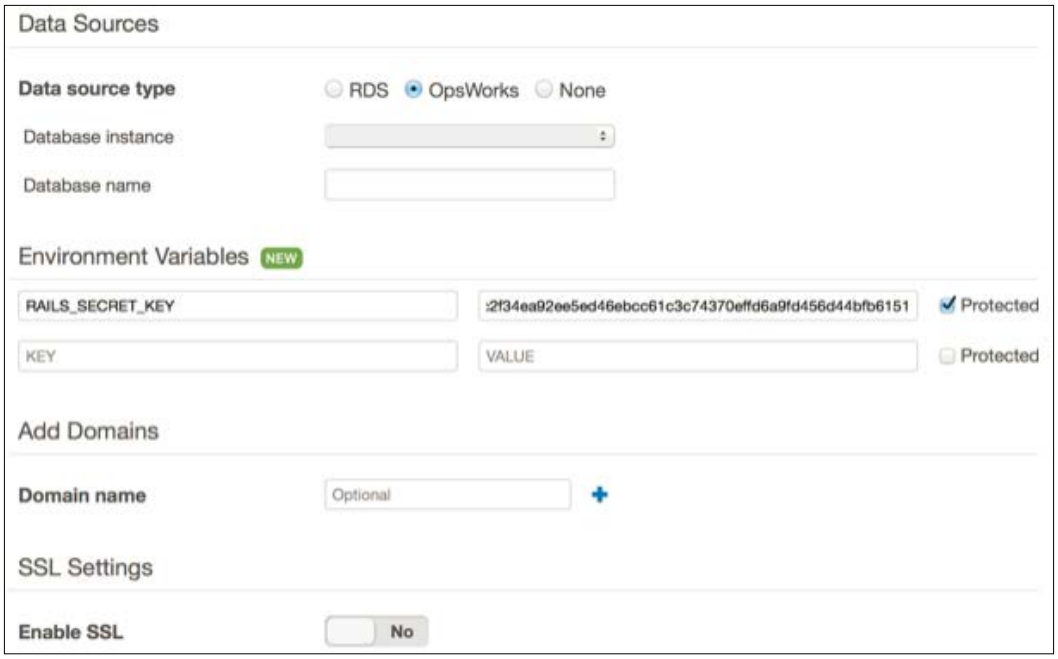
4. Click on **Add Instance** to produce the instance and click on **Start** to commence the instance setup.

5. While instances are set up, let's include your app. Click on the **Apps** link on the sidebar, and then click on **Add an app**.



For this demo example, you're using the Git repository at `https://github.com/awslabs/todo-sample-app.git` as our app source, and using the `opsworks` branch to guarantee that you're deploying the identical code.

6. You can give any name to the app. However, the `TodoApp` name will be equal with fields we will fill out afterward, so if you do modify the name, make sure that you employ that novel name going ahead wherever we utilize `TodoApp`.



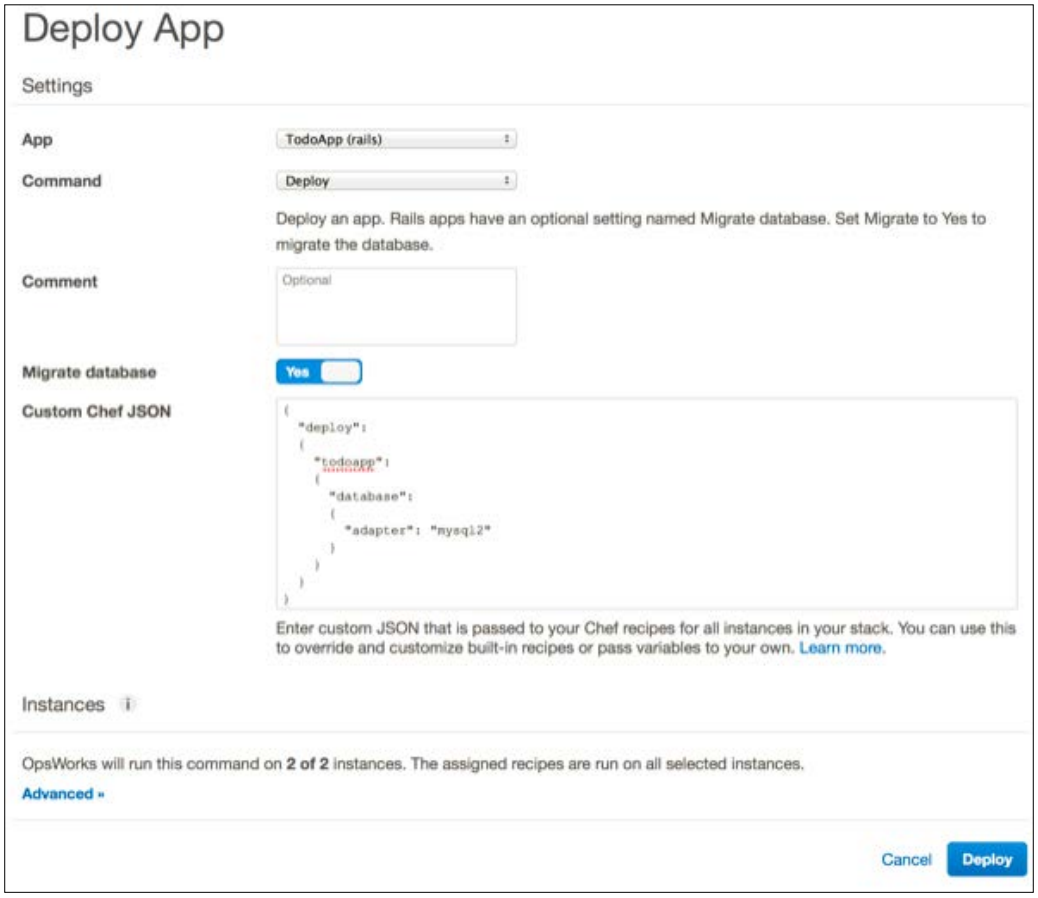
7. To create a value for the `RAILS_SECRET_KEY` environment variable, you can employ the following command surrounded by your replica of the repo:

```
rake secret
```

8. Click on **Add App** once you are done.

It looks like your instances are ready and set up, so it's time to deploy the app:

1. Click on the **Deployments** link on the sidebar and click the **Deploy an App** button. You will get the following screen:



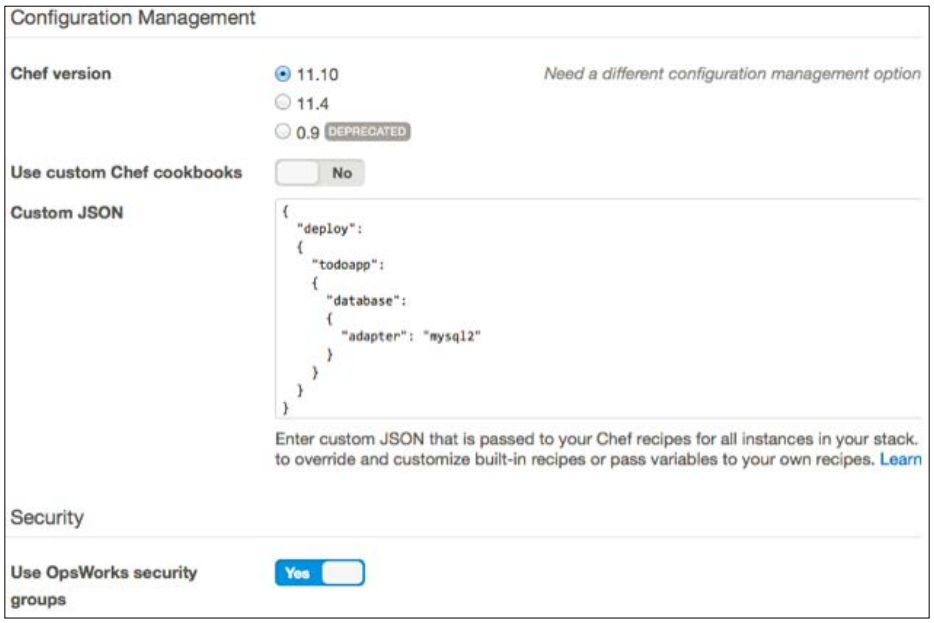
2. As you have not done so yet, keep in mind to select **Yes** for the **Migrate database** setting. You will also require the following custom JSON to make sure that the `mysql2` adapter is used as projected:

```
{
  "deploy":
  {
    "todoapp":
    {
      "database":
      {
        "adapter": "mysql2"
      }
    }
  }
}
```

3. Click on **Deploy** and you are done.

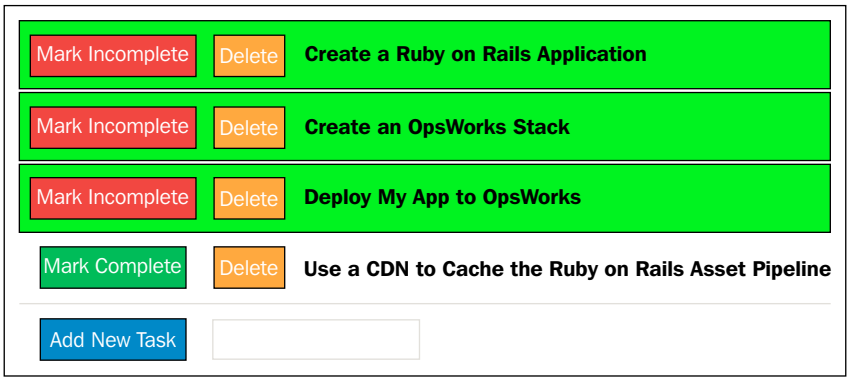
You almost certainly don't wish to be filling in the custom JSON for the adapter option with each deployment. Thankfully, you can shift this custom JSON into your stack settings to have it go with each deployment.

1. Click on the **Stack** link on the sidebar, open **Stack Settings**, and click on **Edit**:



2. Append the custom JSON that you used for your deployment before and click on **Save**.
3. To see the app in action, click on your app server's name on the deployment screen. Click on the link to **Public DNS**, and you should notice the front page of the app:

4. You can insert tasks, mark them finished, and delete them as per your requirements:



That's it. You are done with a basic example of OpsWorks.

Summary

In this chapter, you learned about bootstrapping and user data for EC2 instances. Later on, you went through instance bootstrapping using CloudFormation. Then, you used Chef for continuous deployment on AWS. You also learned about continuous integration and deployment fundamentals. We discussed the process of getting a basic Ruby on Rails app up and running on AWS with the OpsWorks service. At last, we saw AWS Simple Workflow service to understand its process and deployment flow.

In the next chapter, you will learn how to do programming for AWS billing that can be accessed from the application and how to perform cost allocation reporting. You will also learn about the cost control architecture designs to cut down the cost.

10
AWS Billing and Amazon CDN Service

In this chapter, you will learn how to program for AWS billing, which can be accessed from the application, and how to do cost allocation reporting. Also, you will learn cost control architecture designs to cut down the cost. Here is the list of topics that will be covered in the chapter:

- Programmatic AWS billing
- Cost allocation reporting
- Cost control architectures
- CDN service from AWS

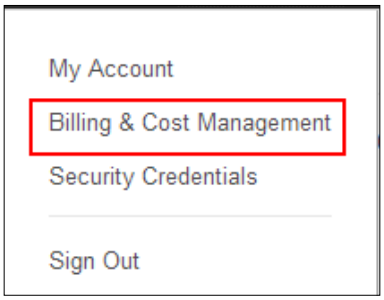
Programmatic AWS billing

If you are a hardcore/frequent user of AWS, then managing and monitoring the billing manually for your account will be very difficult. Therefore, AWS provides a provision for storing granular information (hourly, weekly, monthly, and so on) in our S3 location as a CSV file. In this way, a user can programmatically access this report through SDK or CLI. There are a few steps involved in this billing operation:

1. Spinning-on complete billing reports
2. Choose the in-depth billing reports as you want
3. Referencing complete billing report data
4. Controlling access to billing report files

Turning on detailed billing reports

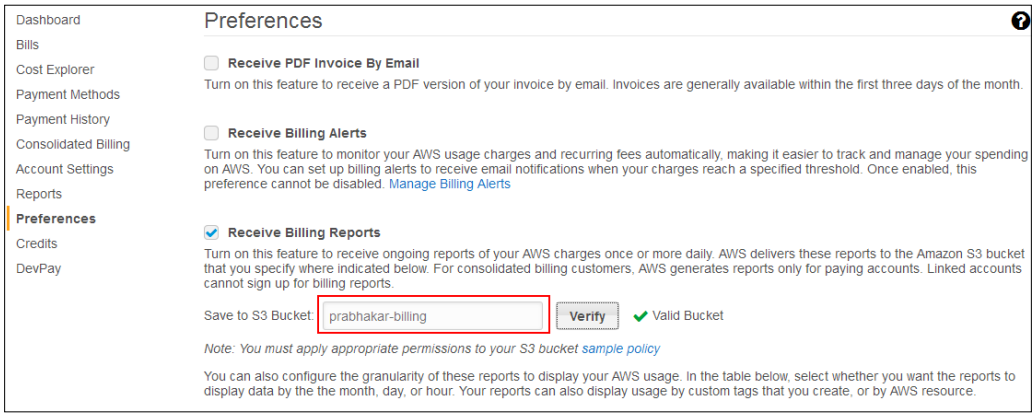
We should first enable detailed billing. The good news is that this operation is free. However, while enabling this, we should have a dedicated S3 bucket to store this report. Standard storage charges apply for this S3 bucket. So, the first step is to create a bucket. For this purpose I have already created a bucket with name `prabhakar-billing`. The second step is to access the billing dashboard by clicking on the **Billing & Cost Management** option in the AWS header.



This will take us to the billing management page. In this page, we need to click on the **Preferences** tab.

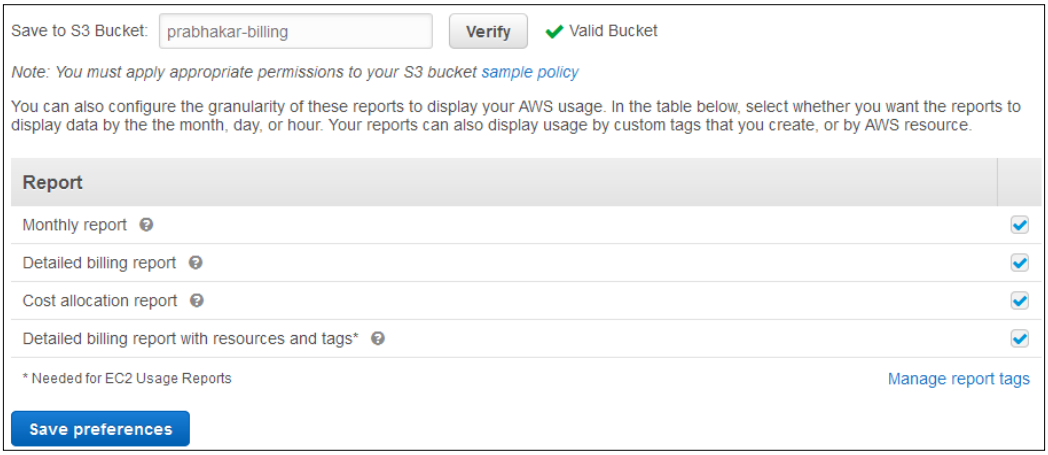
For programmatic billing, we need to check the third check box (**Receive Billing Reports** check box). Enabling this will ask us to provide the S3 bucket name in which the billing details will be stored.

In order to check whether the correct permissions are there for AWS resources to write their reports into this bucket, click on the **Verify** button. This should return a green tick.



Select the detailed billing reports you want to receive

Once the bucket permissions are validated, the page will list the following reports. Based on what kind of granularity is needed, we can choose one or more of the options. The following reports will have billing reports starting from the point when detailed billing is enabled:



Here are the billing conventions and details:

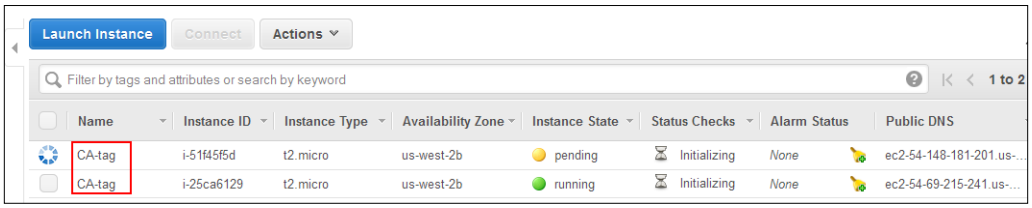
- **Monthly report**: This inclines AWS convention for all product aspects used by an account and its IAM users on monthly basis. It can be transferred from the **Bills** page of the **Billing and Cost Management** console.
- **Detailed billing report**: This tilts AWS convention for every merchandise measurement used by an account and its IAM users on hourly basis.
- **Cost allocation report**: This comprises the identical data as the monthly report, but also contains whichever charge provision tags that you've produced.
- **Detailed billing report with resources and tags**: This holds the data such as the thorough billing report, contains any charge provision tags you've fashioned, and has ResourceID values for the AWS resources used by your story.

Referencing your detailed billing report data

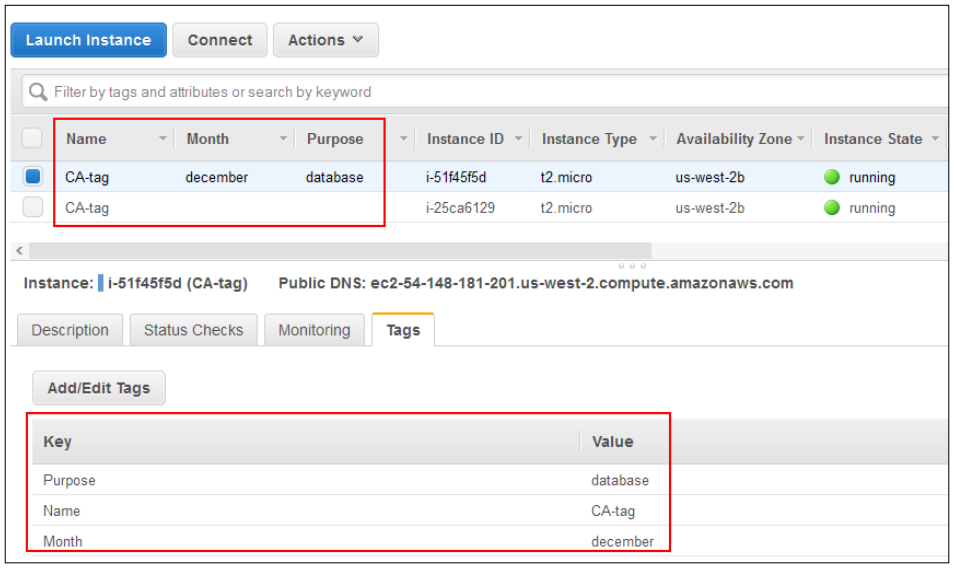
The size of the CSV file grows bigger as we start using more and more AWS resources. After some point, the file will become too big to be read using simple MS Excel software. So, we have to read those reports using SDK, CLI, the S3 REST API, or any other tool that knows how to process CSV files.

Cost allocation reporting

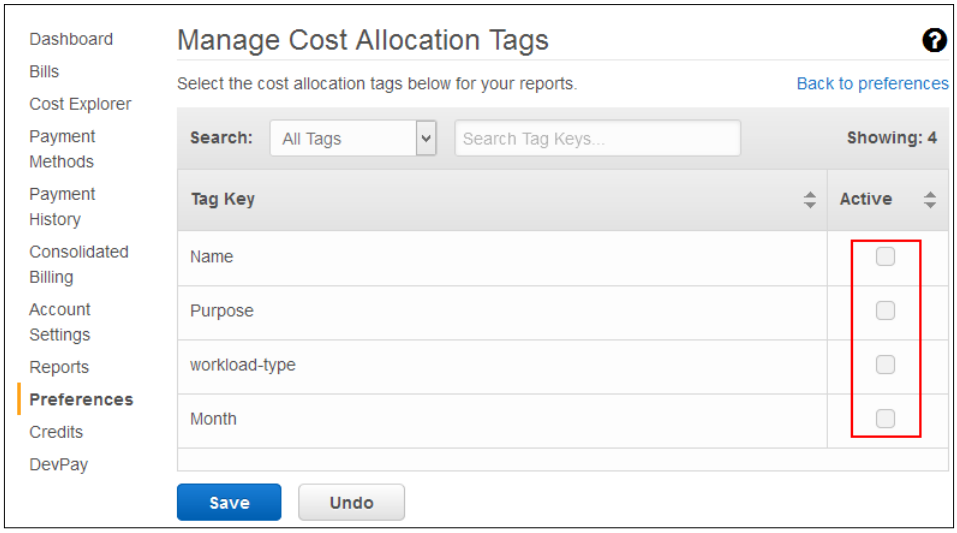
The cost allocation report (as already discussed) will have the detailed billing cost information about the tag created by us. Now, the question is, what is a tag? Tag is an indicative name that we give to our AWS resources. For example, the following screenshot shows two instances that perform almost same function. If the user wants to know for which function the billing cost is more, the best way is to provide the same tag name to both the instances. This is the function of tagging resources.
In an nutshell, the cost allocation report will provide the billing information grouped by tag name.



AWS allows us to have up to 10 tags for each resource. The following screenshot shows two more tags with the keys **Purpose** and **Month** added to first resource.
In this way, we can easily get the aggregated billing for each tag. This tagging is possible in almost all the AWS resources.



In the **Report** column of the **Billing Preferences** page (already discussed in the *Select the detailed billing reports you want to receive* section), there is a button in bottom-right labeled **Manage report tags**. It will open the following page:



By default, none of the tags will be active. We can enable the check box corresponding to the tag for which we need cost allocation report. After enabling the proper tag, click on the **Save** button. Now, these tags will be ready for report.

Cost control architectures

AWS billing and cost management can easily integrate with IAM service. This allows us to specify which user (specifically in our organization) can have access to our billing data. In the following section, we will see how to give permission to IAM user `016883241246`.

Controlling access to your billing report files

Even though the report (S3 bucket) can be made public and readable by everyone, for security reasons, it's better to restrict access to wrong hands. We can restrict access to the report bucket using the following code, which restricts access only to the specified IAM user:

```
{
  "Version": "2012-10-17",
  "Statement": [
  {
    "Effect": "Allow",
    "Principal": {
      "AWS": "016883241246"
    },
    "Action": [
      "s3:GetBucketAcl",
      "s3:GetBucketPolicy"
    ],
    "Resource": "arn:global:s3:::prabhakar-billing"
  },
  {
    "Effect": "Allow",
    "Principal": {
      "AWS": "016883241246"
    },
    "Action": "s3:PutObject",
    "Resource": "arn:global:s3:::prabhakar-billing/*"
  }
  ]
}
```

The preceding code will restrict permission to other users. This will keep our reports foolproof. Even for cost reduction, you can follow the next statements that highlight some of the common techniques to save money:

- Shutting down unused/idle instances
- Using Reserved and Spot instances
- Removing unnecessary EIPs and EBS volumes

So far, we have seen about cost reduction and billing things on AWS. Now, we are moving to the next important service on AWS platform—Amazon CloudFront.

CDN service from AWS – CloudFront

The success of certain networking web applications and online training providers depends on how well they can deliver their content to the users. The content can be as small as a 1 byte picture or as big as custom software, or training session videos that can be in the range of several GBs. If the users are located in single region (let's say Asia), then our server can be located in any of the Asian locations, whichever is closer to the users. But if the users are located across the globe, then the user located further away will feel some latency while accessing the content. The same (increase in) latency problem will occur if more number of users are accessing the content simultaneously. So, this requires the companies to host their sites in different regions and perform sync across those servers, which is complicated in both design and management.

For this kind of scenario, the companies usually go for **Content Delivery Network** (**CDN**) providers. Each provider has their own SLA and provides service over only few or all regions. In this chapter, we are going to discuss one such CDN provider Amazon, which provides the CloudFront service that provides CDN services. It easily integrates with other AWS services and helps distribute the content to end users with low latency and high data transfer speed.

How CloudFront works

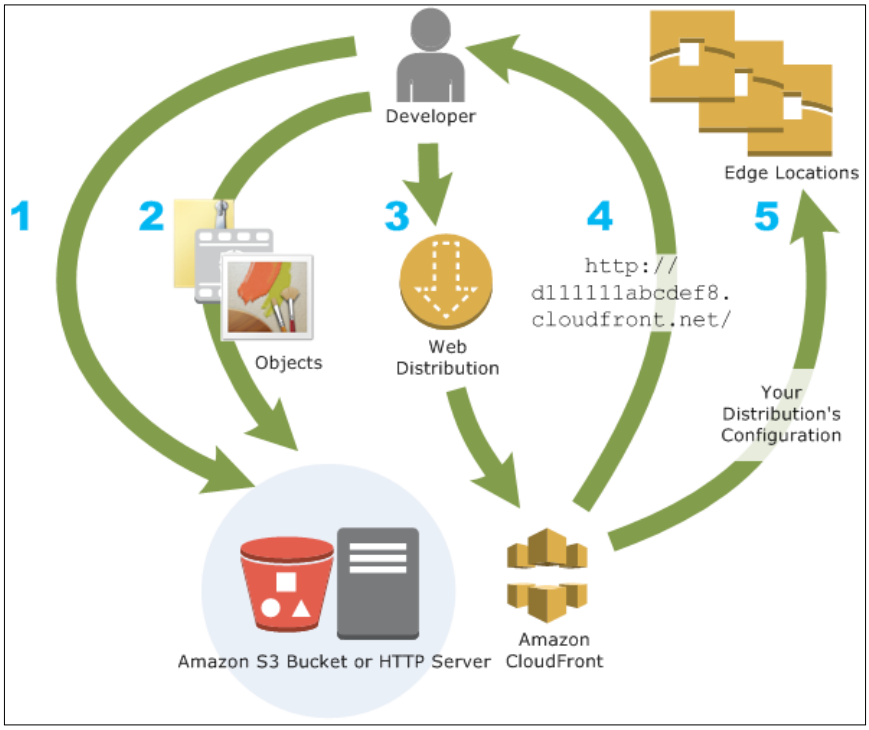
In case of networking sites, whenever a picture is uploaded, all the users tagged in the picture will be notified. Assuming that the user has uploaded the picture from Boston and it got uploaded to the server located at Massachusetts, whenever the user is accessing it from Washington, the image will be served to the user through many server hops (the image is transmitted to New York first, then to Philadelphia, and finally, to Washington).

For 435 miles, it took three hops. But if the user is located in another continent and the data transfer is between the U.S. server and an other continent, then it will add more hops and latency of a few seconds. This is where CloudFront comes into the picture. Irrespective of whether the users are located across regions, or are only in single region, CloudFront will serve it faster. Before diving deep into CloudFront, let's have a look at a few CloudFront nomenclatures:

- **Origin servers**: This is the server whose files are to be delivered using CloudFront. CloudFront supports two broad categories of origin servers. The first is from the S3 bucket and the second is from the HTTP server. There is support for the HTTP server running in EC2 as well as the custom server managed by the company (such a server is termed as custom origins). If the files are available in S3, then we can granularly control (make the file available publicly or privately) the access of the individual files.
- **Objects**: This is the word representing the files that are to be distributed using CloudFront. It can be any static/dynamic content such as images, videos, files, or HTML pages.
- **Distribution**: This is a collection of one or more origin servers. Every distribution has a unique domain name, which will be randomly assigned by CloudFront, if we have purchased a domain name from Route 53 or any other provider.
- **Edge locations**: This is a collection of servers where CloudFront caches copies of objects in geographically dispersed datacenters. CloudFront does this to serve the objects faster simultaneously to users accessing it from multiple regions.
- **Expiration time**: This is the time after which the cached data at the edge location is cleared. The default value is 24 hours, minimum value is 0 seconds, and the maximum value (limit) is almost infinity.

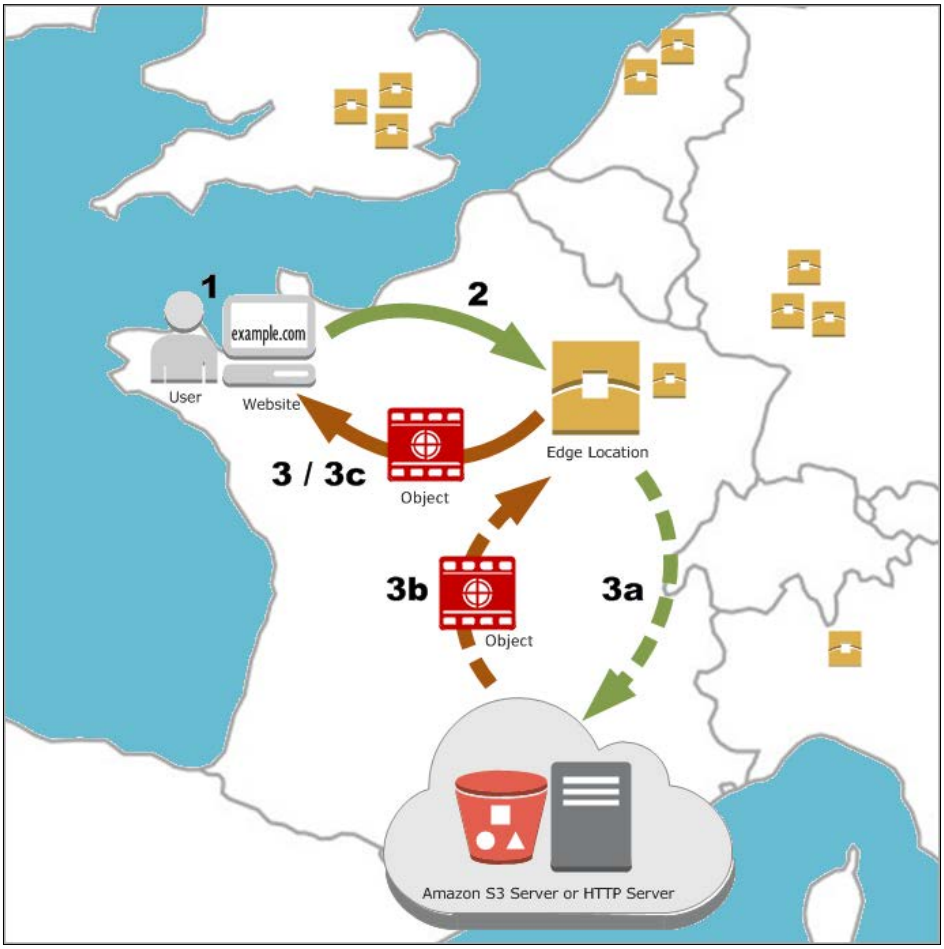
The following figure explains the overall working of CloudFront. The first step is configuring the origin server to the distribution (by specifying the S3 bucket names or HTTP server location). The second step is to put the objects (videos, images, and so on) into the origin server. For the S3 bucket, we can control each object granularly. In the third step, the developer has to create a CloudFront distribution. As soon as the distribution is created, the user will be notified with the CloudFront DNS for the distribution. If the following distribution belongs to an S3 bucket (which has a file named `1.jpg`), then the file can be accessed in two ways: by just appending `/1.jpg` to the CloudFront DNS or by `https://<region-identifier>/<bucket-name>/1.jpg` (provided the file is public).

In the final step, we need to specify the regions where edge locations will be created for faster access speeds. This can be simply the U.S. and Europe; the U.S., Europe, and Asia, or all edge locations. Depending on how the users are distributed across these regions, we need to choose one of these options. The first option is less costly than the third one.



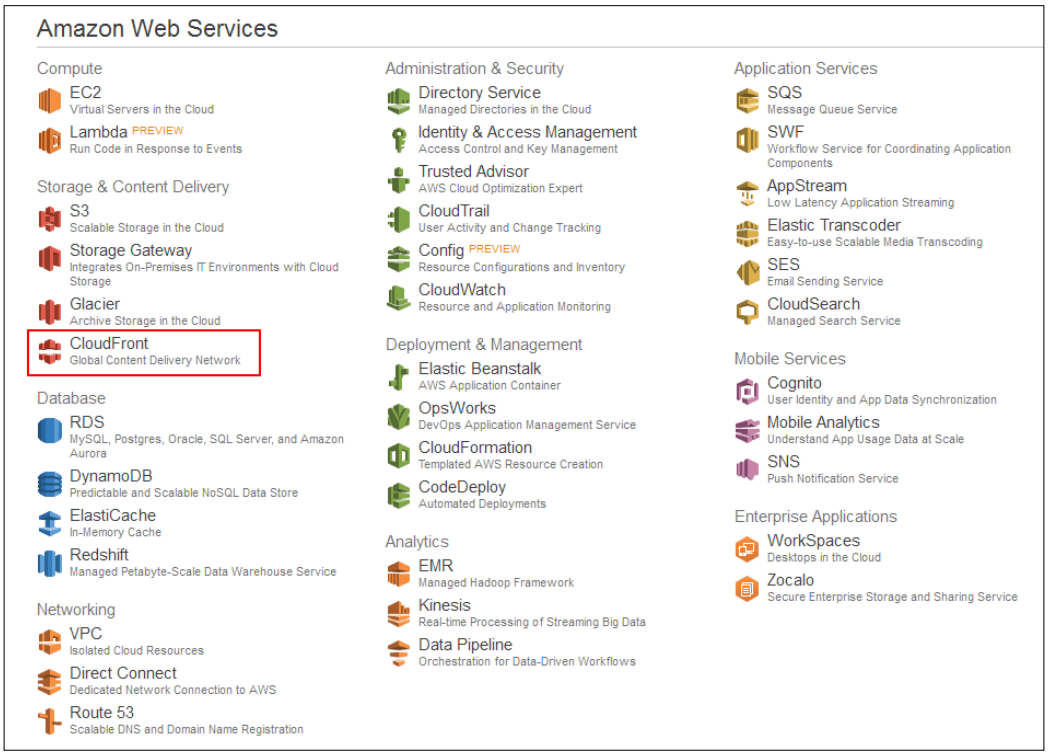
The next figure explains how objects will be served to the end user from edge locations. First of all, the origin server will be located in one, or many, regions as per company preferences. If we choose the distribution objects to be made available in all edge locations, then whenever the object is requested by a user near an edge location, CloudFront will try to fetch it from the edge location (cache). If the object is not found in the edge location, then the object is fetched (by edge location) from the origin server and stored in the edge location (until the expiration time). For further requests from users near this edge location, it will be served directly from the edge location. This process happens for every expiration period. This reduces the number of hops involved in serving the objects.

By default, the expiration time is 24 hours. So, the first user request after 24 hours will refresh the edge location cache (by fetching latest object from origin server, if the file in edge location is not latest) and serve the user request.

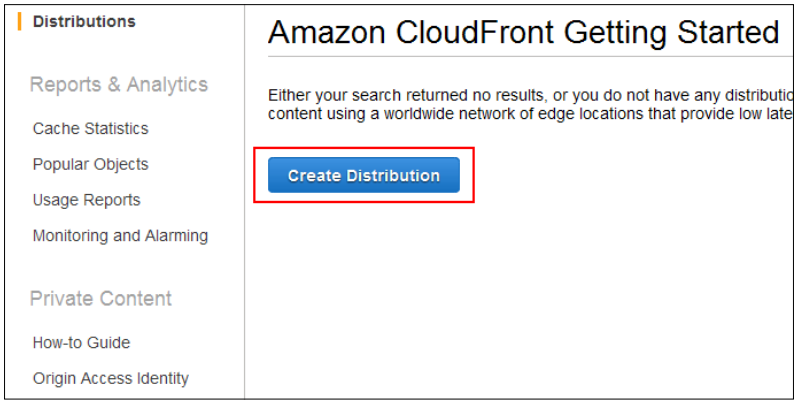


Getting started with CloudFront

To start with CloudFront, all we need is either an EC2 instance with an application running in it or a S3 bucket, whose files are going to be delivered using CloudFront. CloudFront is an easy service to configure and start with. As shown in the following screenshot, it is available in the management console under the **Storage & Content Delivery** category. Just click on the CloudFront icon in the management console.

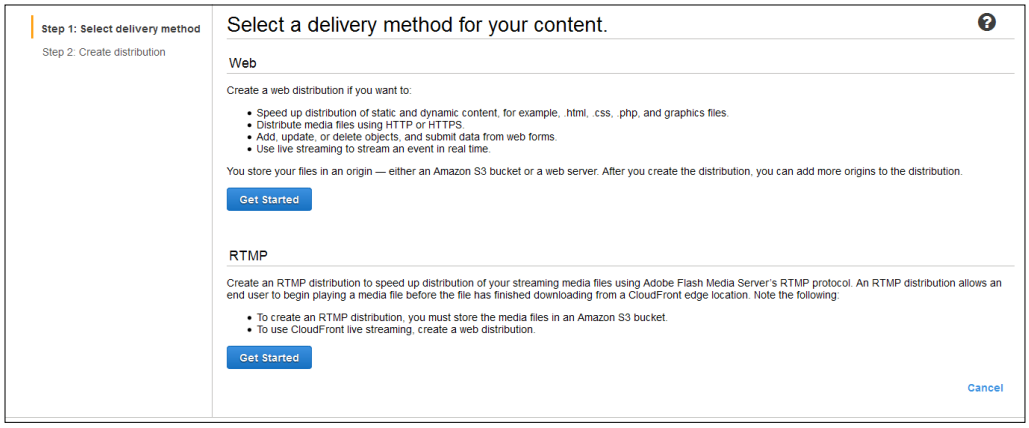


Clicking on the CloudFront icon will take us to the following **Distributions** page. Since this is the first time we are accessing CloudFront, it will show a **Create Distribution** button. Click on this button to start creating our first distribution.



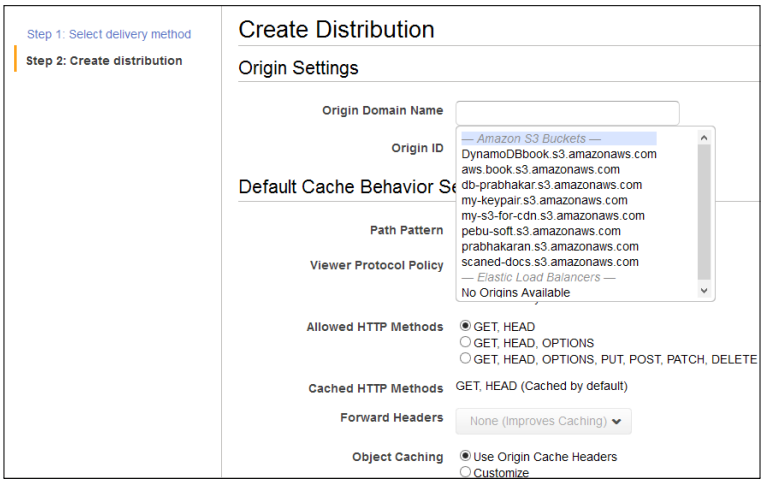
In the **Distributions** page, **Create Distribution** will be visible only when no distribution exists for the account. If some distributions are already created, then the details such as domain name, status, and state of the distributions are shown. There are only two steps involved in creation of a distribution:

1. Choose a proper delivery method. The delivery method can be either web or RTMP. The web method is useful for non-streaming content such as images, HTML pages, or files. On the other hand, RTMP is for streaming content such as video. RTMP allows the video to be streamed even while the video download is in progress. Here, for the sake of simplicity, we click on the **Get Started** button of the **Web** method:

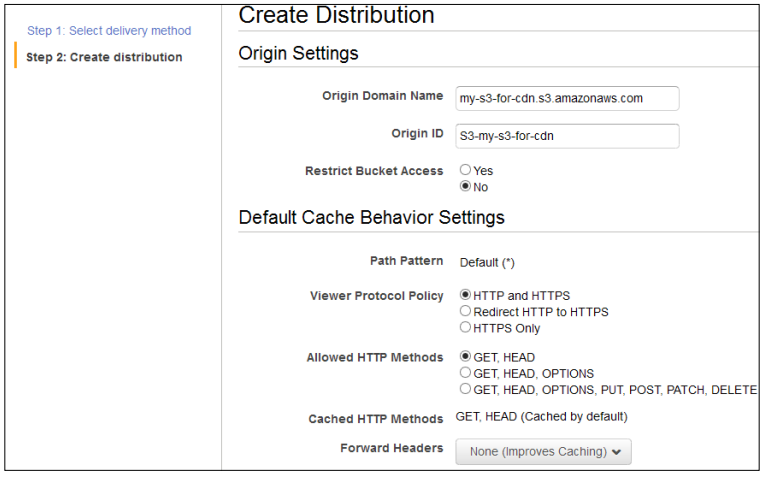


Delivery methods

2. Configure the distribution. There are three broad configuration categories involved in this step. The first step is to specify the origin settings. Here, there will be two text boxes:
3. The first text box asks us to provide the URL of the origin server.



4. Click in the **Origin Domain Name** field and specify the domain name for your origin—the Amazon S3 bucket or web server from which you want CloudFront to get your web content. The drop-down list enumerates the AWS resources associated with the current AWS account. To use a resource from a different AWS account, type the domain name of the resource. For example, for an Amazon S3 bucket, type the name in the format `bucketname.s3.amazonaws.com`. The files in your origin must be publicly readable.



5. Enter a description for the origin. This value lets you distinguish multiple origins in the same distribution from one another. The description for each origin must be distinctive surrounded by the supply.

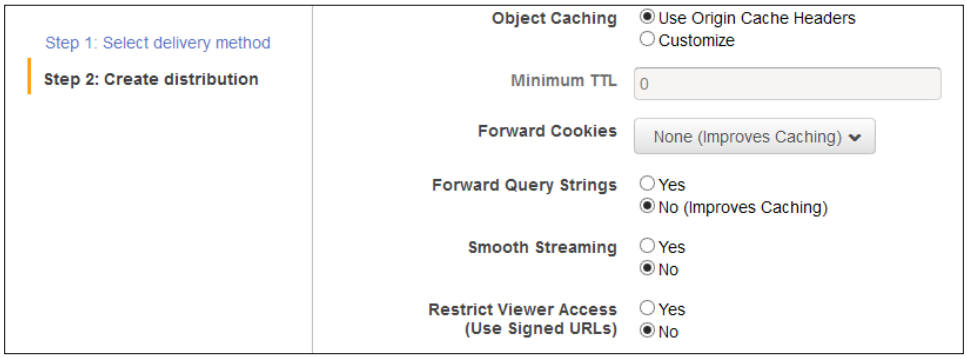
If you need to restrict users to permanently access your Amazon S3 content by means of CloudFront URLs and not Amazon S3 URLs, select the **Yes** radio button. This is beneficial when you're using signed URLs to limit admittance to your content. In the **Help** section, you can see **Serving Private Content through CloudFront** option. In **Help** section with CloudFront, you can do following things:

1. To involve that users permanently access your Amazon S3 content by means of CloudFront URLs, you allocate a distinct CloudFront user—an origin access individuality—to your origin. You can either produce new origin access individuality or reprocess a prevailing one. Supplementary configuration is essential. In the **Help** section, you can check and see the **Serving Private Content through CloudFront** option. Enter a comment that you can use to identify the new origin access identity later, for example, "Static content for example.com".
2. If you want CloudFront to automatically grant read permission to the origin access identity when you create the distribution so that CloudFront can access objects in your Amazon S3 bucket, click on **Yes**, and then **Update My Bucket Permissions**. Whichever option you choose, you should review the permissions on the bucket.

   The default cache behavior only allows a path pattern of `*` (forward all requests to the origin specified by origin). To change the behavior or the routing for other requests (for example, `*.jpg`), add more cache behaviors after you create the distribution.

3. If you want CloudFront to allow viewers to access your web content using either HTTP or HTTPS, specify it. If you want CloudFront to redirect all HTTP requests to HTTPS, specify that too. If you want CloudFront to require HTTPS, specify **HTTPS Only**.
4. Select the list of HTTP methods you want to allow for this cache behavior.
5. Select whether you want CloudFront to forward the headers sent in viewer requests and to cache your objects based on header values.

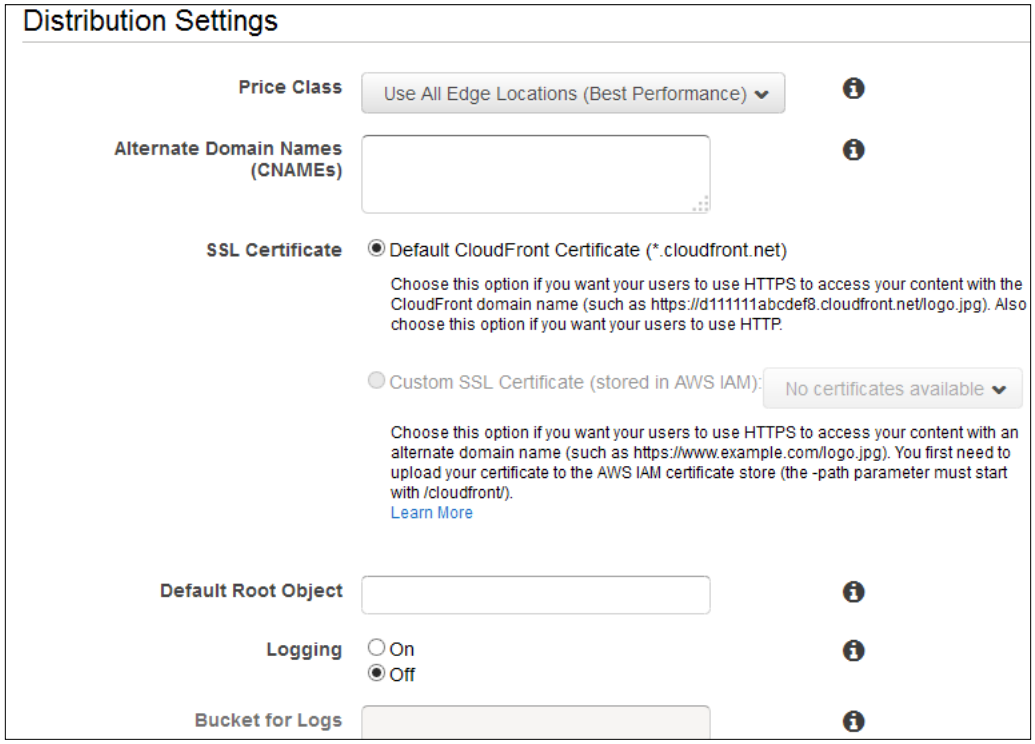
6. Choose **Use Origin Cache Headers** if your origin server is totaling a Cache-Control header to control how long your objects will remain in the CloudFront cache. Choose **Customize** to postulate a lowest time that objects halt in the CloudFront cache irrespective of Cache-Control headers.



   The minimum amount of time for an object is in a CloudFront cache; beforehand, CloudFront forwards additional demand to your origin to govern whether a modernized version is obtainable. The default time is 24 hours. To alter the time that an object is in the cache, shape your origin to enhance a Cache-Control max-age directive.

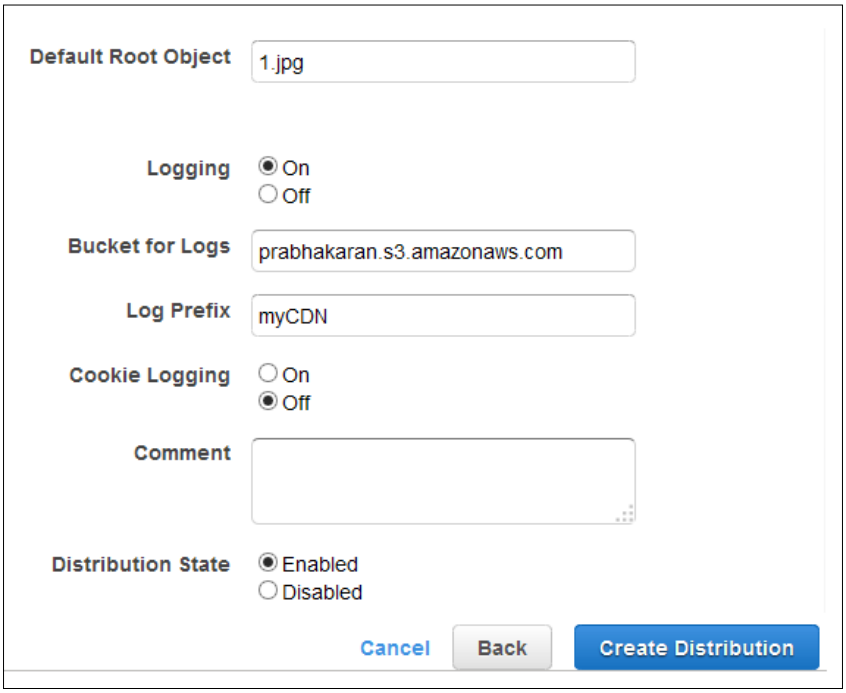
7. Choose whether you want CloudFront to include all user cookies in the request URLs that it forwards to your origin, only selected cookies, or no cookies. If you choose **Whitelist**, enhance the names of the cookies to the Whitelist Cookies arena.
8. Choose whether you want CloudFront to contain query strings in the request URLs that it forwards to your origin. If you want the reappearance dissimilar forms of an object based on the query string, select **Yes**.
9. If you want to use Microsoft Smooth Streaming for on-demand streaming, click on **Yes**.

10. Choose whether you need CloudFront for users to access your content using a signed URL or not. If you choose to limit viewer access, users would require to use signed URLs to access your content.



11. Choose the price class linked with the maximum price that you need to pay for CloudFront service. If you choose a price class as a first priority, some of your users may face difficult latency.
12. If you want to use your own domain name instead of the CloudFront domain name for the URLs for your files, stipulate up to 100 CNAMEs. Separate CNAMEs with commas or place each one in a new line. You also need to create a CNAME record with your DNS service to course queries for `www.example.com` to `u1234.cloudfront.net`.
13. In the **Default Root Object** text box, stipulate the title of the object that you need CloudFront to return (for example, `index.html`) when a viewer appeal points to your root URL (`http://www.example.com`) as an alternative of a specific object in your delivery.

14. Choose whether you require CloudFront to log all viewer requests for files in your distribution. You will be charged for access logs additionally.
15. Click in the field and define the Amazon S3 bucket in which you need CloudFront to save web access logs. To practice a bucket from a diverse AWS account, provide the bucket name in the following format: `bucketname.s3.amazonaws.com`.

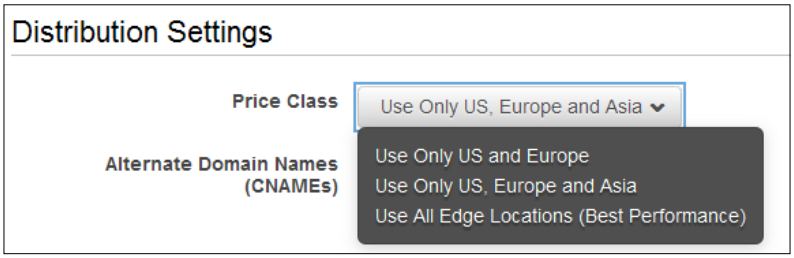


The prefix for the names of log files (for example, `myCDN/`) is optional. The slash after the prefix is optional but recommended to simplify browsing your log files.

16. Select whether you want CloudFront to include cookies in access logs. When the distribution is enabled, CloudFront processes viewer requests for the content associated with this distribution. When the distribution is disabled, CloudFront does not accept any requests for the content associated with this distribution.

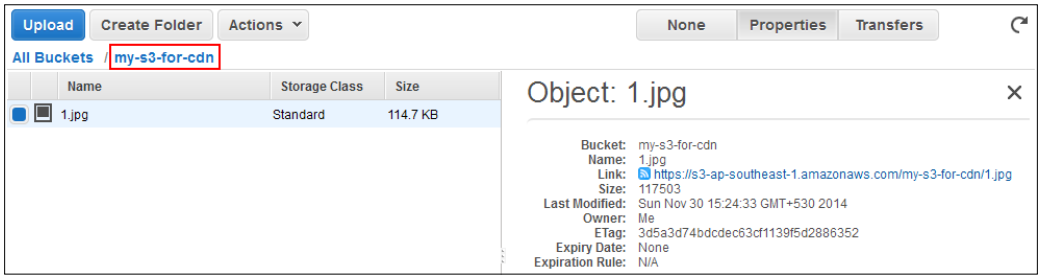
17. Clicking on the **Create Distribution** button will start the distribution creation and take us to the **CloudFront Distributions** page. In the following screenshot, we can see that the random domain name assigned to the distribution is **d1o5k7ird1xt97.cloudfront.net**, the status is **In Progress**, and the state is **Enabled**:

This creation of CloudFront distribution will take some time (10-15 minutes). Right now, the distribution is available in all the edge locations across the world. If the users accessing these objects are only located in the U.S. and Europe, then we are unknowingly placing the objects in edge locations (Asia and many other regions), which would never be accessed by the users. This will add more cost and more read operations from our origin servers. In order to avoid this over-billing, it is possible to choose one of the three options (or price classes) for the distribution. If we feel that the user is located only in the U.S. and Europe, then it's better to select first option, which is cost-effective. Even if few requests for the objects come from other locations, it will be served directly by the U.S./European edge location.

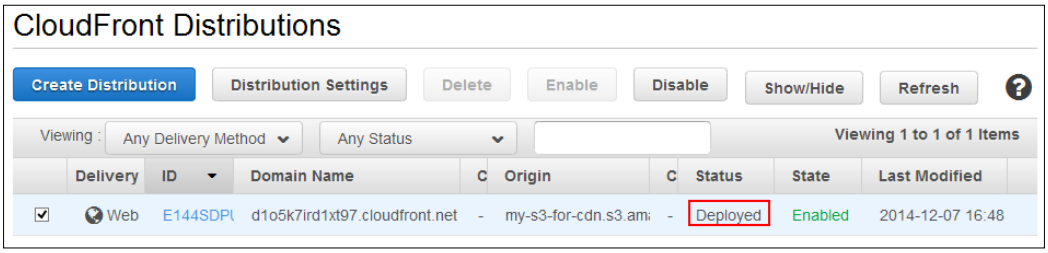


Meanwhile, we will have a look at the S3 bucket acting as the origin server for our distribution. The following screenshot shows the content of the **my-s3-for-cdn** bucket. It has a file named as **1.jpg**, which will act as the index or welcome object. For example, this file can be accessed by giving two URLs:

1. First, by simply entering the CloudFront domain name, which is `http://d1o5k7ird1xt97.cloudfront.net/` in our case.
2. Second, by appending `1.jpg` with the CloudFront domain name, which is `http://d1o5k7ird1xt97.cloudfront.net/1.jpg` in our case. One URL is given in following screenshot for your reference.



3. If the status of the distribution changes to **Deployed**, then we can be sure that our distribution is now made available. If we want to change any of the distribution parameters, all we need to do is select the distribution and click on the **Distribution Settings** button. This will allow us to specify all the parameters, which we have discussed so far, and perform a redeployment. Again, the status of the distribution will become **In Progress** and it will be **Deployed**.



It is not possible to delete a distribution when it is enabled. That is the reason why the delete button is disabled for our distribution. In order to delete the distribution, we must first disable it and then we can delete it.

4. Copy the domain name and paste it in the browser. You will be able to see the `picture 1.jpg` (which is available in the S3 bucket, origin server, and we have set it as root object). Even if we append `1.jpg` to the domain name, we will see the same picture:

So, you see the output as a clear image that is coming from the CDN network.

Even when we are changing any of the distribution parameter and the distribution is in the **In Progress** status, it can serve the user requests.

These were the simplest steps to configure the AWS CloudFront service with your content.

Streaming

For the streaming of content, AWS CloudFront offers the following features:

- **On-demand Smooth Streaming**: Use CloudFront to distribute video using the Smooth Streaming format deprived of the requirement to setup, configure, and control any media servers.
- **Live Streaming for Amazon CloudFront using Adobe Media Server 5.0**: This has a provision for both Flash-based and Apple iOS campaigns with Adobe Media Server 5.0.
- **Live Streaming for Amazon CloudFront Using Windows Media Services**: This allows you to distribute live media over HTTP to Microsoft Silverlight consumers and Apple iOS expedients.
- **Live Streaming with Wowza**: This offers Live Streaming Support with Wowza Media Server.

Summary

In this chapter, you learned how to do programming for AWS billing, which can be accessed from the application, and how to do cost allocation reporting. Furthermore, we discussed the AWS billing configuration and cost control tips to cut down the cost. Finally, we discussed the AWS CloudFront service in detail. In the next chapter, you will come to know about big data and Apache Hadoop on AWS cloud. You will also learn how to use the EMR and Kinesis services with big data analytics and for Hadoop solutions.

11
Analyzing Big Data with AWS

The data we work with is heterogeneous. Users produce content such as blog posts, tweets, social media interactions, and images. Servers log messages continuously. Scientists generate comprehensive data about the ecosphere around us. The Internet, a vital source of data, is incomprehensibly bulky. This astounding development in data has affected businesses. Old-fashioned database systems, such as relational databases, have been pushed to the boundary. These systems are breaking under the forces of "Big Data." Old-fashioned systems and the data management practices accompanying them have failed to scale up to Big Data. In this chapter, we will cover the following topics:

- An introduction to Big Data and Hadoop
- An introduction to Amazon Elastic MapReduce
- Working with Hive
- Amazon Kinesis

To hold the encounters of Big Data, a new strain of skills has emerged. Many of these new technologies have been grouped under the term NoSQL. In some ways, these innovative technologies are more complex than traditional databases and, in some ways, they are modest. These systems can scale to massively large sets of data, but these technologies need an essentially innovative set of techniques. They are not one-size-fits-all explanations. Many of these Big Data systems were initiated by Google, along with distributed filesystems, the MapReduce computation framework, and distributed locking services. Another prominent contributor to this space was Amazon, which created its own product; a groundbreaking, distributed key-value store called DynamoDB. The open source community answered in the following years with Hadoop, HBase, MongoDB, Cassandra, and many other projects.

Introducing Big Data and Hadoop

Big Data carries with it two key challenges: how to store and work with ample data sizes and, more importantly, how to comprehend data and improve it. Hadoop fills a breach in the market by storing and providing computational proficiencies over considerable amounts of data. It's a distributed system made up of a dispersed filesystem, and it offers a way to parallelize and execute programs on a cluster of machines. You've most probably come across Hadoop, as it's been embraced by technology titans like Yahoo!, Facebook, and Twitter to meet their Big Data requirements. It's building inroads across all business sectors.

Hadoop is a platform that delivers distributed storage and computational proficiencies. Hadoop was first developed to answer a scalability concern that occurred in Nutch, an open source crawler and search engine technology. At that time, Google had issued papers that termed its novel distributed filesystem, the **Google File System** (**GFS**), and MapReduce, a computational framework for parallel processing. The successful implementation of these concepts in Nutch led to two separate projects, the second of which became Hadoop—a first-class Apache project. Hadoop is a distributed master-slave architecture that consists of the **Hadoop Distributed File System** (**HDFS**) for storage and MapReduce for computational competences. Properties essential to Hadoop are data splitting and parallel computation of huge datasets. Its storage and computational proficiencies scale with the accumulation of hosts in a Hadoop cluster. It can extend volume sizes in PB on clusters with thousands of hosts to fulfill its Big Data requirements. It is also building inroads into all business segments.

Introducing Amazon Elastic MapReduce

Amazon AWS provides Hadoop as a PaaS. Establishments and people can access Hadoop clusters on the fly, run their workloads, and download outcomes. Provisioning a Hadoop cluster using **Elastic MapReduce** (**EMR**) takes a few minutes and a few steps.

The common steps to form and run workloads on EMR are as follows:

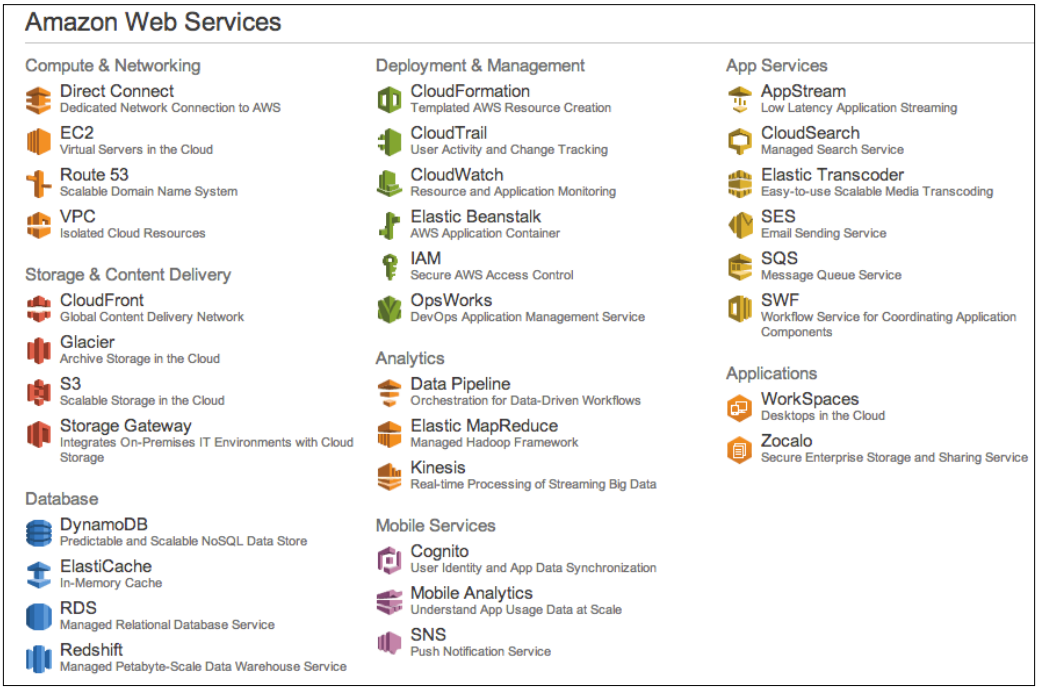
1. The application is developed locally in Java using Hadoop's MapReduce APIs, Hive (**Hive** is a data warehouse product that facilitates querying and managing huge datasets residing in distributed storage) or a language of the user's choice. Languages not based on Java can be executed in a Hadoop cluster using Hadoop Streaming.

2. The application and the relevant data are stored in Amazon S3. Numbers of clients for data upload are displayed on web interface can also be used. Data can be written directly to HDFS on the EMR cluster as sound.
3. From the AWS Management Console, the cluster configuration is stated and launched. Cluster configuration displays the huge number of machines in the cluster, the version of Hadoop to use, and supplementary applications that have to be installed on the cluster. The movements to be achieved once the cluster is provisioned are also a part of this stage.
4. The cluster will be launched after the data processing is finished, and the outcomes are either moved into S3 or can be read off HDFS on the cluster.

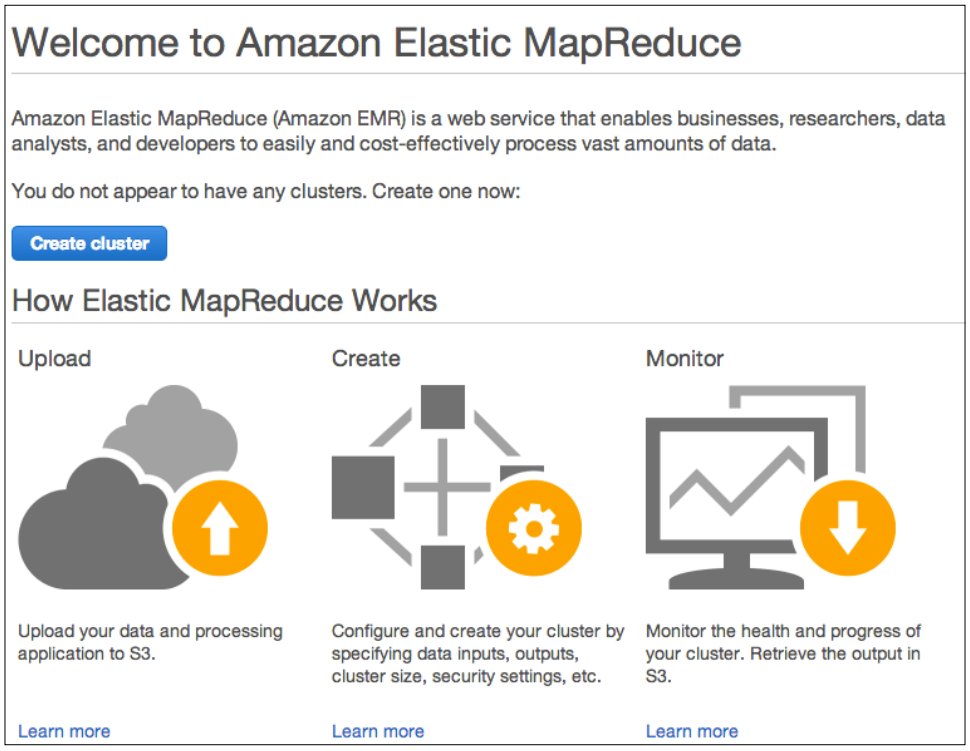
Provisioning a Hadoop cluster on EMR

Follow the steps given here to start with EMR:

1. The services of importance to us are **S3** and **Elastic MapReduce**:

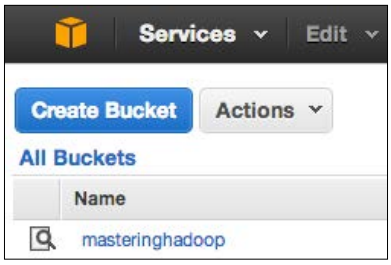


2. Clicking on the **Elastic MapReduce** service will take you to the EMR service management page. It provides a brief outline of EMR and the key stages to launch a cluster. The **Create Cluster** button is used to launch the Hadoop cluster wizard:

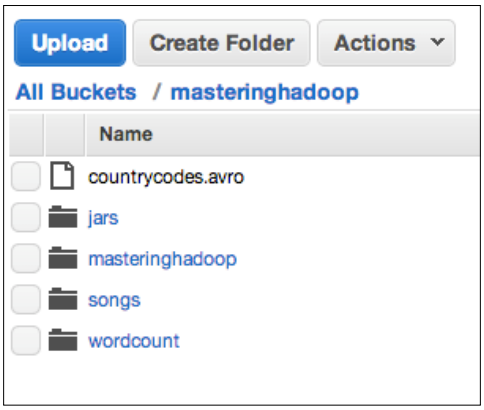


3. Before starting the launching process of Hadoop cluster, we have to upload the data and the equivalent application onto S3.
4. Files can be uploaded to S3 via the **Upload** button when a specific bucket is designated. There is a button to form folders inside the bucket.

The files show metadata such as size, storage class, and the last altered date and time of that specific file. Files can be put in the S3 bucket using a number of file manager programs that are available for S3 or via the web interface as follows (you have to upload Hadoop JAR from web to S3 bucket):



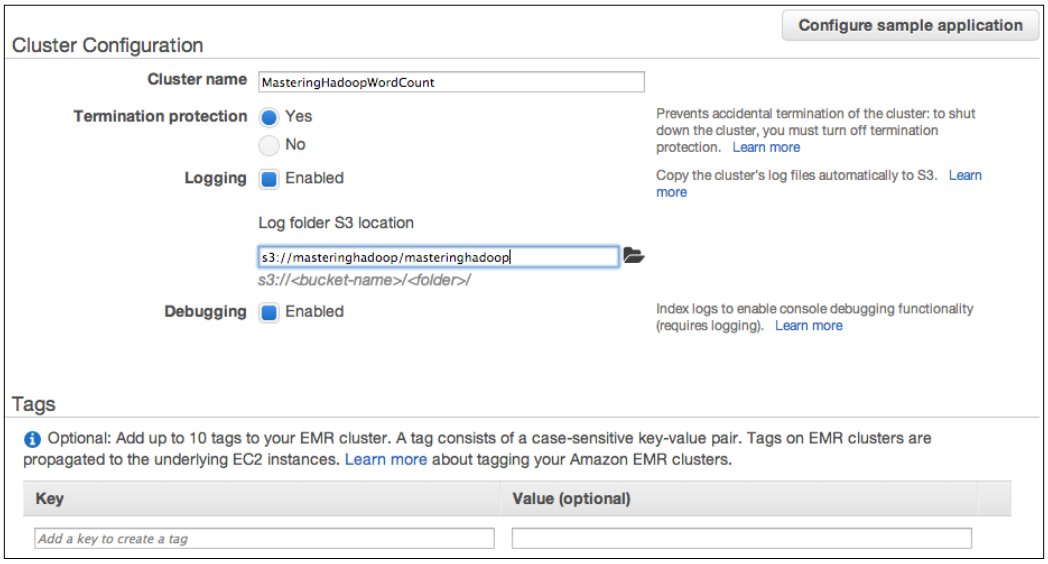
The following image shows the **masteringhadoop** bucket with folders and files in an S3 account:



Amazon EMR delivers a number of model jobs that can be run on the provisioned clusters. A nice functionality of Hadoop is streaming for word count. The program is written in Python and counts the words of a text file. The files whose words need to be counted and the program are already on an unrestricted bucket on S3 and can be used by anybody to experiment on the EMR cluster. A sample word count example can be viewed on the following URL:

`http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html`.

1. Let's continue by going back to the EMR console and clicking on the **Create Cluster** button on the console. The **Create Cluster** page has numerous subdivisions. Each division configures a specific facet of the cluster.
2. The main division is the **Cluster Configuration** section, where the cluster properties are declared. It is shown in the following screenshot of this section. Some of the components of **Cluster Configuration** are as follows:
    - **Cluster name**: An approachable name for the cluster. This name will help identify and manage a cluster based on the name
    - **Termination protection:** This is set to `Yes` in the following example. When turned on, it precludes the cluster from terminating when failures are met. If the cluster needs to be dismissed, it has to be overtly set to `No` before termination
    - **Logging**: This can be turned on and an S3 path can be stated to save the logs. Logs are printed on the `/mnt/var/log` directory of the "Master Node". These are copied onto S3 at intervals of 5 minutes
    - **Debugging**: By empowering debugging, an index of the log files will be fashioned in SimpleDB
3. The next division is the **Tags** division. Up to 10 key-value strings can be associated with the EMR cluster. These tags are preserved in the underlying EC2 instances that run the Hadoop cluster.



4. The next divisions configure the software and the hardware of the EMR cluster we would require.

   In the Software Configuration segment:

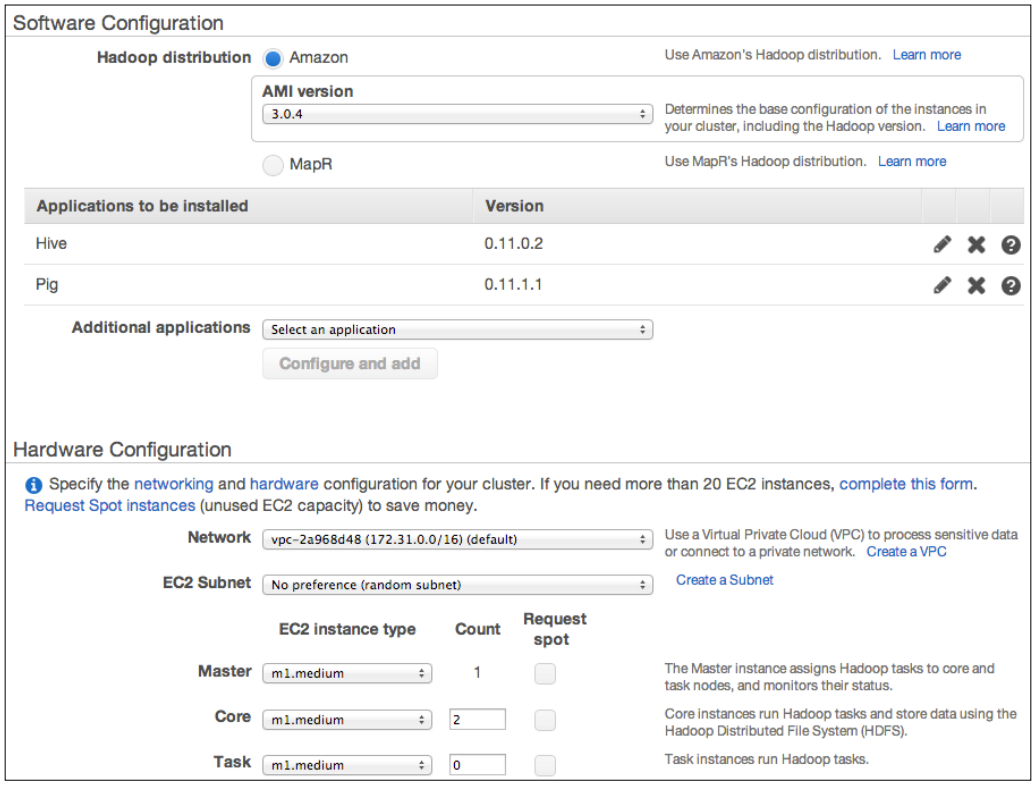
   - **Hadoop Distribution**: The Hadoop distribution to be used in the cluster can be set here. Amazon has its own Hadoop distribution. It also supports the Hadoop distribution of MapReduce. Hadoop versions 2.4.0, 2.2.0, 1.0.3, and 0.20.205 are present. Each version of Hadoop has different AMIs corresponding to it. In our example, we will go through Hadoop 2.2.0 that is installed on AMI version 3.0.4 to be consistent throughout the chapter.
   - **Applications to be installed**: By default, Hive and Pig are installed. HBase, Impala, and Ganglia are three other applications that are accessible with this sort of Hadoop.

   In the Hardware Configuration segment:

   - **Network**: A VPC can be used to connect to a private Cloud for processing subtle data. In the example, we will select a default VPC.
   - **EC2 Subnet**: You can choose from all the subnets available within your region and there is an option to choose a random subnet.

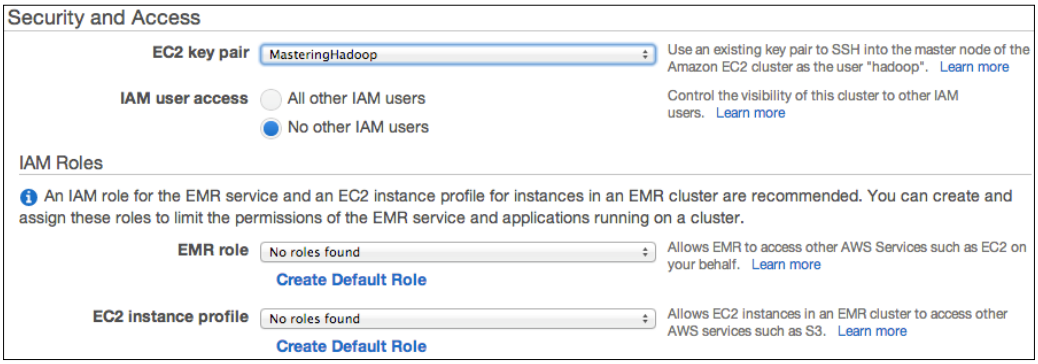
5. There are three types of EC2 instances:
   - **Master**: This EC2 instance is answerable for handover tasks to the diverse core and task nodes. At least one master instance should be there.
   - **Core**: These nodes execute tasks as well as the ones that act as data nodes.

- **Task**: These nodes can only implement tasks. They do not have the data node module and are not fragment of the HDFS.



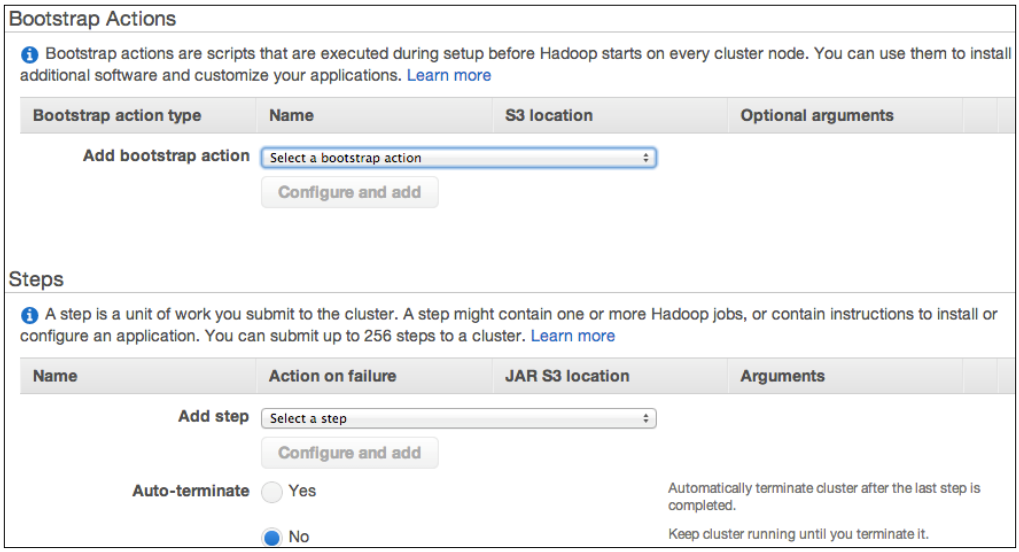
6. The next segment is the **Security and Access** segment. It permits the user to set access control on the cluster and specify the keys for access. If you need to SSH into any of the EC2 instances, you need an Amazon EC2 key pair.

7. Setting a role in the **EMR role** dropdown permits the application using that role to access other AWS services. Correspondingly, setting a role in the **EC2 instance profile** allows EC2 instances within EMR to access other AWS services:



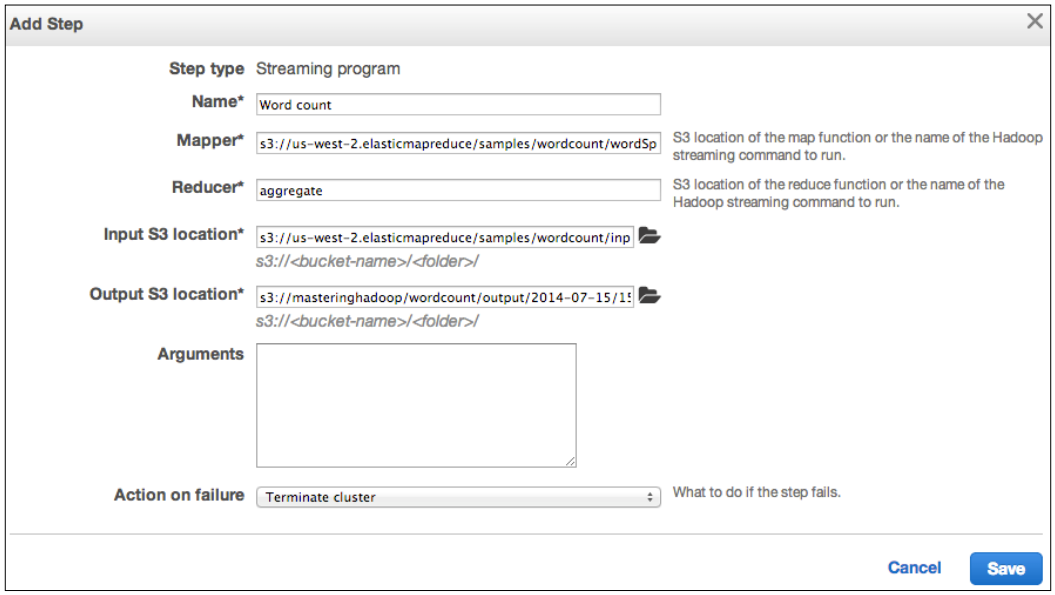
8. The next segment is about setting the **Bootstrap Actions** for the cluster. Setup scripts can be listed here to instruct any distinct configuration that would be essential before beginning the cluster. The **Add bootstrap actions** dropdown has the following options:
    - Configure Hadoop
    - Configure daemons
    - Run if
    - Custom action
9. The final subdivision is the **Steps** subdivision. This is the division where jobs can be acquiesced to the Hadoop cluster. In the example, the dropdown has options to execute following programs:
    - A Hive program
    - A Pig program
    - A Streaming program
    - An Impala program
    - A custom MapReduce Java JAR

We will understand how we can supplement a streaming program from the AWS EMR samples that are at present obtainable. This subdivision also has an auto terminate action that terminates the cluster before the last step has been executed:

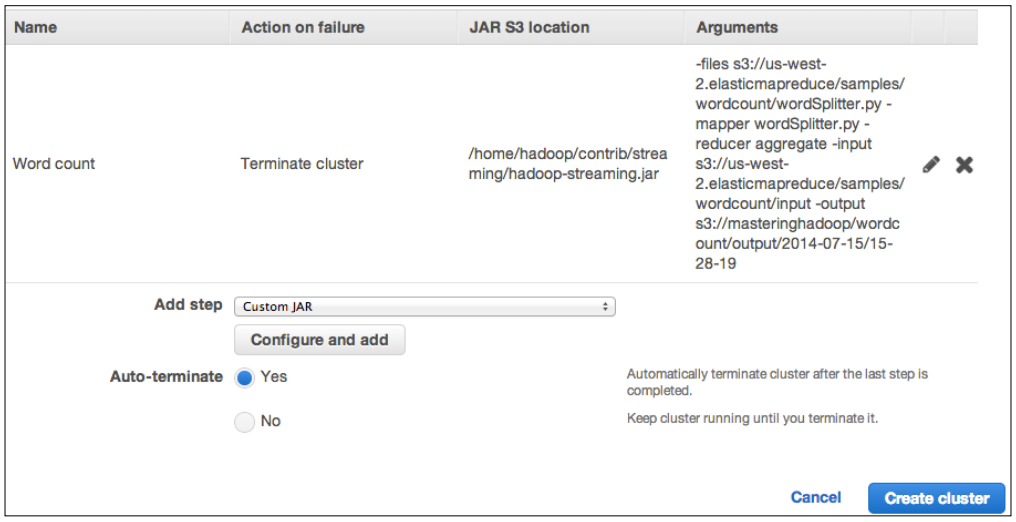


10. In the given example, let's choose a Hadoop **Streaming program** step. Clicking on the **Configure and add** button starts the **Add step** wizard, as shown in the next screenshot.
11. We will give a user-friendly name for the given step. The **Mapper** task is set to the Python program given by the S3 path, `s3://us-west-2.elasticmapreduce/samples/wordcount/wordSplitter.py`. The **Reducer** is set to `aggregate`. This is a built-in reducer that sums the values corresponding to every key. The folder holding the files on which we run word count is given by the S3 path, `s3://us-west-2.elasticmapreduce/samples/wordcount/input`. This is stated in the **Input S3 location**. The **Output S3 location** is indicated by the path, `s3://masteringhadoop/wordcount/output/2014-07-15/15-28-19`. The word count output is put in this folder. Any further arguments can be stated in the **Arguments** box.

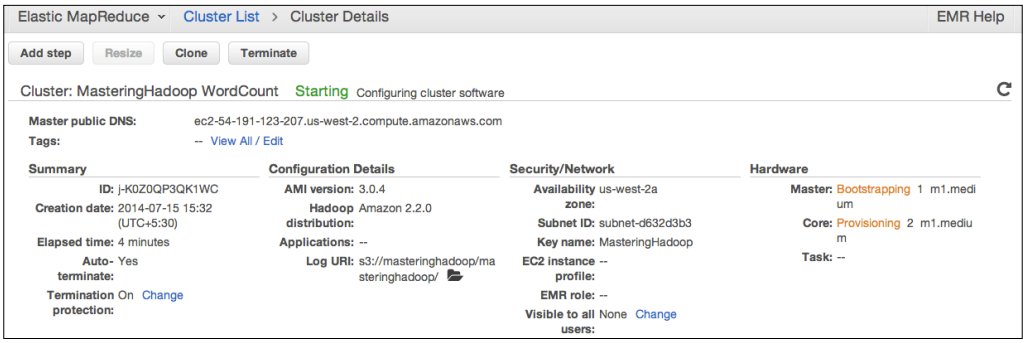
12. We can also postulate the steps to be undertaken if a failure is faced. Click on the **Save** button and you can review the earlier running of the cluster:



13. The **Steps** segment now looks like the following screenshot. Click on the **Create cluster** button to start provisioning the cluster, as shown here:

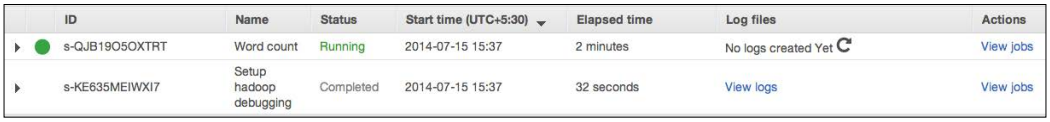


14. Clicking on a cluster shows the details of the cluster:

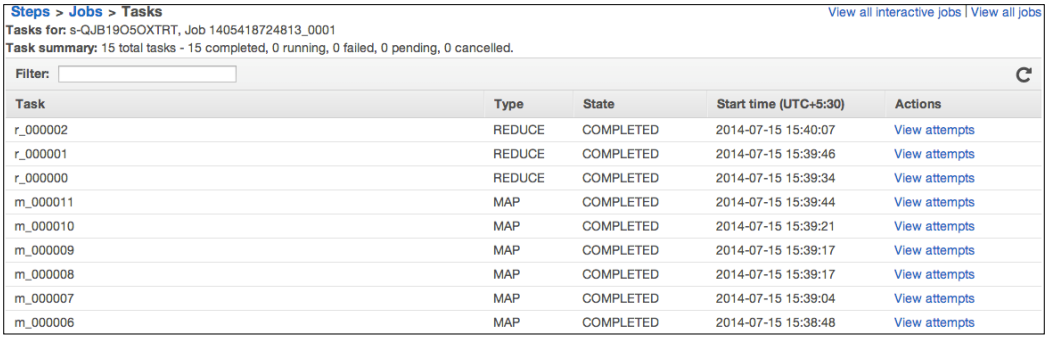


Cluster details

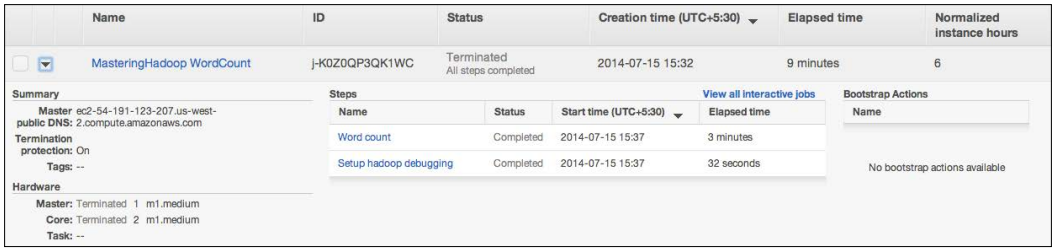
15. When we expand the **Steps** section, the details are as follows:



16. The link in the right-hand side corner gives details of the jobs being executed in the step. Clicking on the **View jobs** link of the current step shows the particulars about the job and its tasks as shown in the following screenshot. Three reduce tasks were executed along with 12 map tasks to complete the program. The attempts by the tasks can be analyze by clicking on the given **View attempts** link:

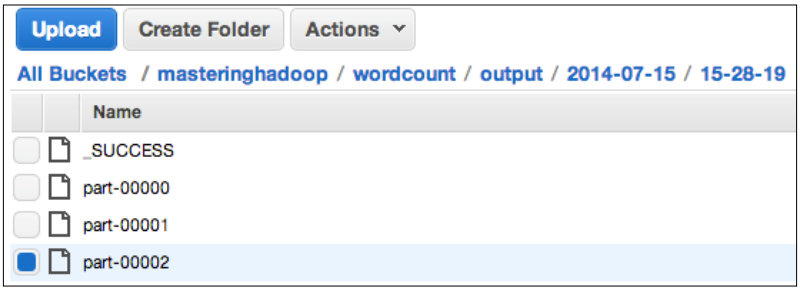


17. Expanding cluster details on the EMR dashboard can also display a swift summary of the cluster. The screenshot of this view is shown in the following image:

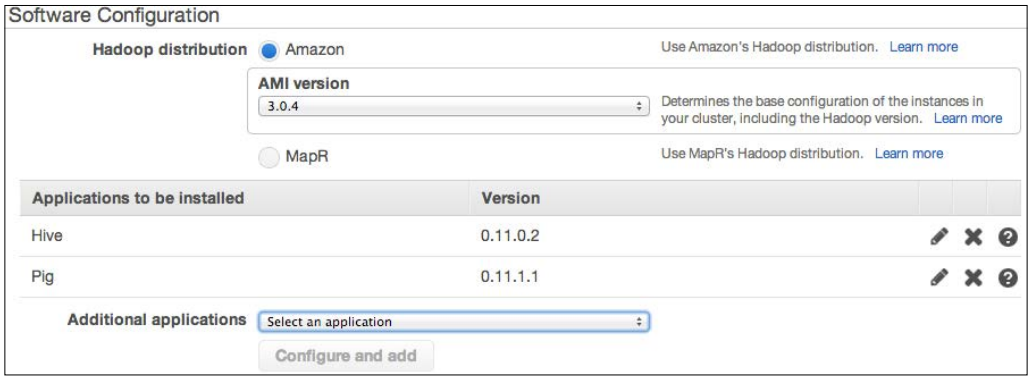


Cluster summary

18. Lastly, when the job is done, the output files can be realized in S3. The output directory was stated when initiating the job. The following image of the output directory in S3 is given as follows:



During the software configuration phase, we need to specify the install of Hive and/or Pig based on the necessity. The subsequent screenshot shows the software configuration segment of a cluster with Hive and Pig installed. Based on the AMI selection, a suitable version of Hive and Pig are accessible:



Once the cluster is provisioned with Hive and Pig installed, we can SSH into the master node. It is essential to get a key-value pair from Amazon and give it to the cluster. If this is not done, it is not possible to SSH into the cluster instances.

All Hadoop services run under the Hadoop user. By inputting the respective application names on the CLI of the master node, we can start the Hive (grunt) shell, as shown in following screenshot:

Hive (grunt) shell

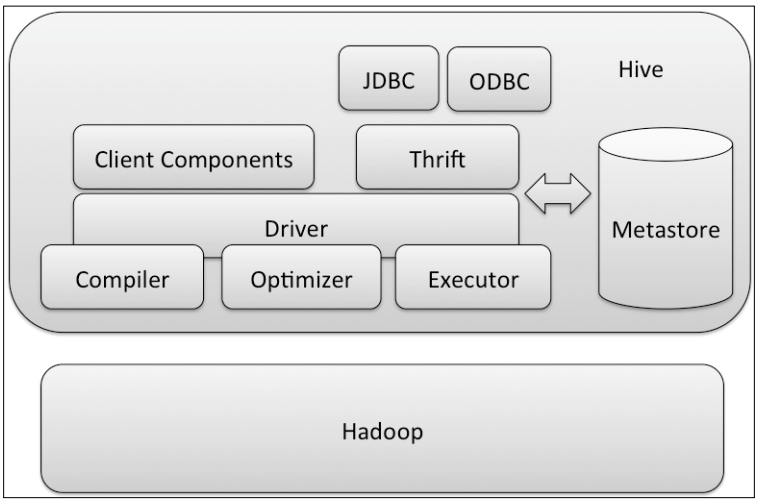
Interactive commands can now be executed using the preceding prompts. File locations can be identified with `s3://<bucket name>/<folder name>` paths to read them off S3.

Hive presents relational and SQL notions into Hadoop via MapReduce. A natural step in bringing further customers onto Hadoop is to flatten the knowledge curve by adopting notions they are well acquainted with.

Apache Hive is frequently labeled as a data warehouse infrastructure. Conventionally, business intelligence is assembled from a data warehouse, a database that stores data from many foundations contained by an enterprise. This data store is mainly queried for reporting and analytics. Usually, infrastructure that makes data warehouses involves RDBMS and the query language (SQL) which is used to achieve analysis and produce reports. Data warehouse infrastructure consists of relational data stores and is queried using SQL. Distinct star or snowflake schemas were used to model these data stores. Apache Hive continues this tradition of SQL, but changes the essential data store to HDFS. The queries are decoded into MapReduce jobs. The alternative to SQL used in Hive queries is called **HiveQL**.

Hive structural design

The following image shows the Hive structural design:



Metastore

The metastore is a Hive catalog for system allied metadata. It provisions facts around the tables, partitions, schemas, and table locations. It can be retrieved via the Thrift interface making it conceivable to read this data using clients transcribed in various different programming languages. The data is stored in a relational database system and uses an **Object-Relational Mapping** (**ORM**) layer to read and write data into the store. The reason behind using an RDBMS for the metastore was to condense the latency when serving this material to the Hive query compiler.

The ORM layer of the metastore permits a pluggable model, where any RDBMS can be plugged into Hive. The default RDBMS used is Apache Derby, an open source relational data store. In practice, establishments use MySQL and other widespread RDBMS suites to host the metastore. The data in the metastore imposes structure on otherwise raw HDFS files. This makes it dangerous to defend the metastore from smashes by consistent backups or replication. The metastore is solitary retrieved throughout compilation and never when MapReduce jobs are running.

Compiler

The compiler leads a HiveQL query and renders it into MapReduce jobs.
A parser parses the query and constructs an **Abstract Syntax Tree** (**AST**). The AST is tested for types and semantic stabilities. Metadata from the metastore is used to accomplish this phase. The output of the checks is a **Directed Acyclic Graphs** (**DAG**) operator. A sequence of optimization transformations are then applied on the DAG. The transformations are immobilized and the output is an optimized operator tree. Users are permitted to add their transformations by implementing the `Transform` interface. The optimized DAG is then translated into a physical plan. The physical plan is a set of MapReduce and HDFS jobs. A HDFS job is used to read and write data from HDFS.

The execution engine

The execution engine precedes the plan produced by the compiler and implements the job firmly in order of their dependencies. The plan is transferred to each task in the Hadoop cluster via a `plan.xml` file. This file is dispersed in the cluster using a side channel like the DistributedCache. Job outputs are deposited in temporary locations. On the accomplishment of the whole query, if a store place is indicated, these files are moved to suitable locations as indicated by the **Data Manipulation Language** (**DML**). If a query is deprived of a store position, the outcomes are served unswervingly from the momentary place.

Supporting apparatuses

The Hive infrastructure has a number of supporting apparatuses:

- The **driver** is the element that levers query submissions. It is accountable for organizing the lifecycle of a query by appealing the resources in the accurate order to finish it. The driver also spawns sessions and keeps track of session information.
- There are numerous client apparatuses that are used to submit queries to Hive. The famous ones are the CLI, a web interface, and JDBC/ODBC connectors.
- Extensibility apparatuses such as the **SerDe** and **ObjectInspector** interfaces are there to assist users assimilate with diverse data types and legacy data. **User-defined Functions** (**UDFs**) and **User-defined Aggregate Functions** (**UDAFs**) are convention utilities that can be transcribed by the user to deploy Hive's proficiencies.

Data types

Hive provides all the primary numeric data types such as `TINYINT`, `SMALLINT`, `INT`, `BIGINT`, `FLOAT`, `DOUBLE`, and `DECIMAL`. Along with these primitive data types, Hive also provisions string data types such as `CHAR`, `VARCHAR`, and `STRING`. Like SQL, the time pointer data types such as `TIMESTAMP` and `DATE` are contemporaneous. `BOOLEAN` and `BINARY` assorted types also exist.

A number of composite types are also available. Composite types can be composed from other primitive or composite types. The composite types are as follows:

- `Structs`: These are alliances of data foundations alike to a C-struct. The dot representation is used to dereference essentials surrounded by a struct. A field in the column C defined as a `STRUCT {x INT, y STRING}` can be retrieved as `U.x` or `U.y`. The syntax is as follows:

  ```
  STRUCT<field_name : data_type>
  ```

- `Maps`: These are key-value data types. Given that the key (surrounded by square braces) can contact a value, then a value of a map column M that maps from key `x` to `y` can be retrieved by `M[x]`. The syntax is as follows:

  ```
  MAP<primitive_type, data_type>
  ```

- `Arrays`: These are lists that can be randomly accessed through their position. The syntax is as follows:

  ```
  ARRAY<data_type>
  ```

- `Unions`: There is a union data type available in Hive. It can hold a component of one of the data types stated in the union. The syntax is as follows:

  ```
  UNIONTYPE<data_type1, data_type2…>
  ```

Hive functions and data types are case insensitive and available based on Hive versions.

Data model

Hive data is organized as databases. A **database** is a logical collection of Hive tables. A database surrounded by Hive, consigns a namespace for its tables. If no namespace is assigned to Hive tables, they go to the default database. Creating a database results in the creation of a HDFS directory for the files in the database. This directory serves as the namespace for the tables. The `CREATE DATABASE hadoop;` command constructs a Hadoop database. When we list the HDFS directory structure, we can check a directory created for this database with the following command:

```
drwxr-xr-x   - uchit supergroup 0 2015-02-28 18:55 /user/hive/warehouse/
hadoop.db
```

Now let's discuss the terminology behind Hive and its use cases:

- A **table** is the elementary unit of data storage that is similar to RDBMS. It groups records of the same type. Records are rows equivalent to typed columns. A table maps to a single directory within HDFS. Hive also permits impressive data structures on present data locations via **external tables**. Metadata stored inside Hive for each table holds the column types and list of columns. It also contains other information such as the owner of the table, serialization and deserialization evidence (Serde) for the columns, data storage formats, and bucketing related metadata. Databases and tables are stored in the HDFS location stated by `hive.metastore.warehouse.dir`.

While postulating the `LOCATION` for an `EXTERNAL` table in HDFS, Hive presumes the data files to be in a directory.

- **Partitions** are separations of the table based on separate column values. When partition columns are quantified, all the records consistent to discrete values or value recipes of the columns are stored in a subdirectory within the table directory. Partitions are used as filters to stop excessive records from being processed, decreasing query latency and I/O time. It is true that partitions also increase the number of files in HDFS and, therefore, the number of map tasks and in-between outputs. A `DROP` command on an `EXTERNAL` table does not delete the data in HDFS.
- **Buckets** or **clusters** are files in the leaf-level directories that are connected to records that have the equivalent column value hash. A Hive user can stipulate the number of buckets per partition or per table.

Let's take the file `worldcitiespop.txt` and create a table out of it. The following **Data Definition Language** (**DDL**) query shows how a schema can be executed on an external table:

```
CREATE EXTERNAL TABLE hadoop.worldcities_external(code
VARCHAR(15), name STRING, fullName STRING, region INT, population
BIGINT, lat FLOAT, long FLOAT)
COMMENT 'This is the population table'
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION '/user/uchit/worldcitiespop';
```

For the code, check `3632EN_11_01.txt` from the code bundle. The keyword `EXTERNAL` is used to specify a present table. There are keywords accompanying every bit of metadata that is deposited in the Metastore. `ROW FORMAT` is used to postulate the serialization and deserialization semantics of a table. If `ROW FORMAT` is not indicated or `ROW FORMATDELIMITED` is designated, a Hive native SerDe is used to create the table rows. The `STORED AS` clause postulates the underlying file format used by the table and `LOCATION` indicates the place where the table data is stored within HDFS. When the `EXTERNAL` keyword is used, no additional HDFS directories are produced.

It is compulsory in Hive that every table maps to an HDFS directory containing an `EXTERNAL` table.

Let's try to create a table that is not external using the DDL query as follows (for the code, check `3632EN_11_02.txt` from the code bundle):

```
CREATE TABLE hadoop.worldcities(code VARCHAR(15), name STRING,
fullName STRING, region INT, population BIGINT, lat FLOAT, long
FLOAT)
COMMENT 'This is the population table'
PARTITIONED BY (region_p INT)
CLUSTERED BY (code) SORTED BY (code) INTO 2 BUCKETS
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS SEQUENCEFILE;
```

This table definition states a partition column inside the table. It also specifies the number of buckets, two, within every partition. The fundamental file format of the table is the `SEQUENCEFILE` format. A partition column cannot have an identical name to any other column. When loading data into the table, the partition column must be preserved as a separate column.

Using the following DML query, we will colonize the above table using the external table we fashioned:

```
set hive.enforce.bucketing = true;
set hive.enforce.sorting = true;
set hive.exec.dynamic.partition = true;
set hive.exec.dynamic.partition.mode=nonstrict;
set hive.exec.max.dynamic.partitions.pernode=1000;
FROM MasteringHadoop.worldcities_external
INSERT OVERWRITE TABLE MasteringHadoop.worldcities
PARTITION(region_p)
SELECT code, name, fullName, region, population, lat, long, region
WHERE region IS NOT NULL;
```

You'll find the code snippet in `3632EN_11_03.txt` in the code bundle. All partitions have two buckets that will be organized on the country code.

Indexing on Hive tables

In an RDBMS, indexing is used for faster lookup of data, which in turn transmits to quicker queries. Hive indexes are alike to non-clustered indexes from old-fashioned databases. They keep track of mapping amid the data and the relevant HDFS blocks they exist in. This permits a MapReduce job to check out only the related blocks to practice queries.

In our example, an index is fashioned using two diverse handlers, the compact and the bitmap handler. A Hive index is nothing but a table in HDFS. The `DEFERRED REBUILD` directive is used to initiate Hive to fill the index at an advanced stage. An `ALTER INDEX` command can be supplied to construct the index at a later argument of time. The code is as follows:

```
USE hadoop;
CREATE INDEX worldcities_idx_compact ON TABLE worldcities (name)
AS 'COMPACT' WITH DEFERRED REBUILD;
CREATE INDEX worldcities_idx_bitmap ON TABLE worldcities (name) AS
'BITMAP' WITH DEFERRED REBUILD;
DESCRIBE hadoop__worldcities_worldcities_idx_compact__;
```

You'll also find the following code snippet in `3632EN_11_04.txt` in the code bundle. The output of the `DESCRIBE` operation on the squashed index tables is presented next. For every partition and bucket, the index table grasps an array of offsets. These equalizers can be used to unswervingly acquire the block of data. This can be seen in the following code snippet:

```
hive> DESCRIBE hadoop__worldcities_worldcities_idx_compact__;
OK
name                    string
_bucketname             string
_offsets                array<bigint>
region_p                int

# Partition Information
# col_name              data_type               comment

region_p                int
Time taken: 0.071 seconds, Fetched: 9 row(s)
```

Bitmap indexes are used when the number of conceivable values taken by the indexed column is the reduced amount. The index table structure for a Bitmap index is also related. Nonetheless, the information encoded is diverse. The `_bitmaps` field provisions a bit for every record in the table. If the presented value is contemporaneous in the record, that individual bit is turned on or off. The index table structure of a bitmap index is as follows:

```
hive> DESCRIBE hadoop__worldcities_worldcities_idx_bitmap__;
OK
name                    string
_bucketname             string
_offset                 bigint
_bitmaps                array<bigint>
region_p                int

# Partition Information
# col_name              data_type               comment

region_p                int
Time taken: 0.087 seconds, Fetched: 10 row(s)
```

Amazon Kinesis

As designated by Amazon, **Kinesis** is a *fully managed service for real-time processing of streaming data at immense scale*.

Kinesis can ingest data from countless sources extending from cell phones to large servers or, in general, "any scheme that is capable of making a `put` call can be a foundation of data for Kinesis". Kinesis is supplementary around rapid calculations and this rapidly computed data can be later deposited into other AWS Services such as Redshift, DynamoDB, and so on for additional investigation.

Visualize the method in which lumberjacks move lumber. It is pushed in to a rapidly moving stream, where it floats all the way to the end and is picked up and taken away to factories for subsequent treatment. In the same way, AWS Kinesis can be measured as a constantly rolling stream. Cell phone devices, servers who push the server logs, Facebook comments, and stock market data are fed into the stream and AWS Kinesis, executing the **Kinesis Client Library** (**KCL**) will be at the end the stream prepared to process the data, as and when it arrives.

Kinesis terminology

The following terms are used most frequently when you are working with Amazon Kinesis. To start with Kinesis, Amazon uses their pre-build terms. So we will go though those terms and their definitions.

Streams

An Amazon Kinesis stream is a methodical order of data records. Each record in the stream has a classification (sequence) number that is allocated by Amazon Kinesis. The data records in the stream are dispersed into shards.

Data records

A data record is the component of data deposited in an Amazon Kinesis stream. Data records are composed of a sequence number, partition key, and data blob, which is an absolute sequence of bytes. Amazon Kinesis does not examine, construe, or alter the data in the blob in whichever way. A data blob can be up and about to 50 KB.

Producers

Producers place records into Amazon Kinesis streams.

Consumers

Consumers acquire records from Amazon Kinesis streams and process them.

Shards

A shard is an exclusively recognized group of data records in an Amazon Kinesis stream. A stream is made up of multiple shards, each of which delivers a stationary piece of capability. Every shard can provision up to five read transactions per second, up to an extreme total of 2 MB of data read per second and up to 1000 records inscribed per second, up to a concentrated total of 1 MB data transcribed per second. The data aptitude of your stream is a utility of the number of shards that you stipulate for the stream. The total capability of the stream is the summation of the capabilities of the aforementioned shards.

Partition keys

A partition key is used to assemble data by shard surrounded by a stream. Amazon Kinesis isolates the data records that are in the right abode in a stream that is into manifold shards, by means of the partition key accompanying with every data record to govern which shard a specified data record fits to. Partition keys are Unicode strings with a supreme extent boundary of 256 bytes. An `MD5` hash function is used to map partition keys to 128-bit integer values and to map linked data records to shards.

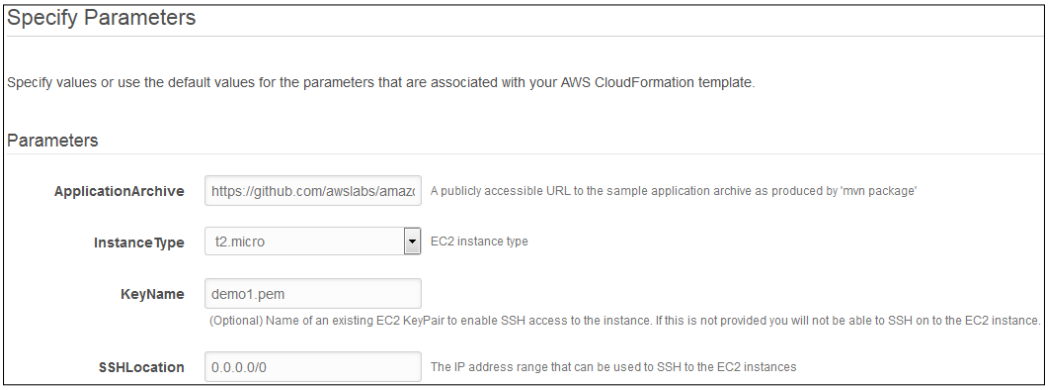
Amazon Kinesis Client Library

The Amazon Kinesis Client Library is accumulated into your application to empower fault-tolerant ingestion of data from the Amazon Kinesis stream.

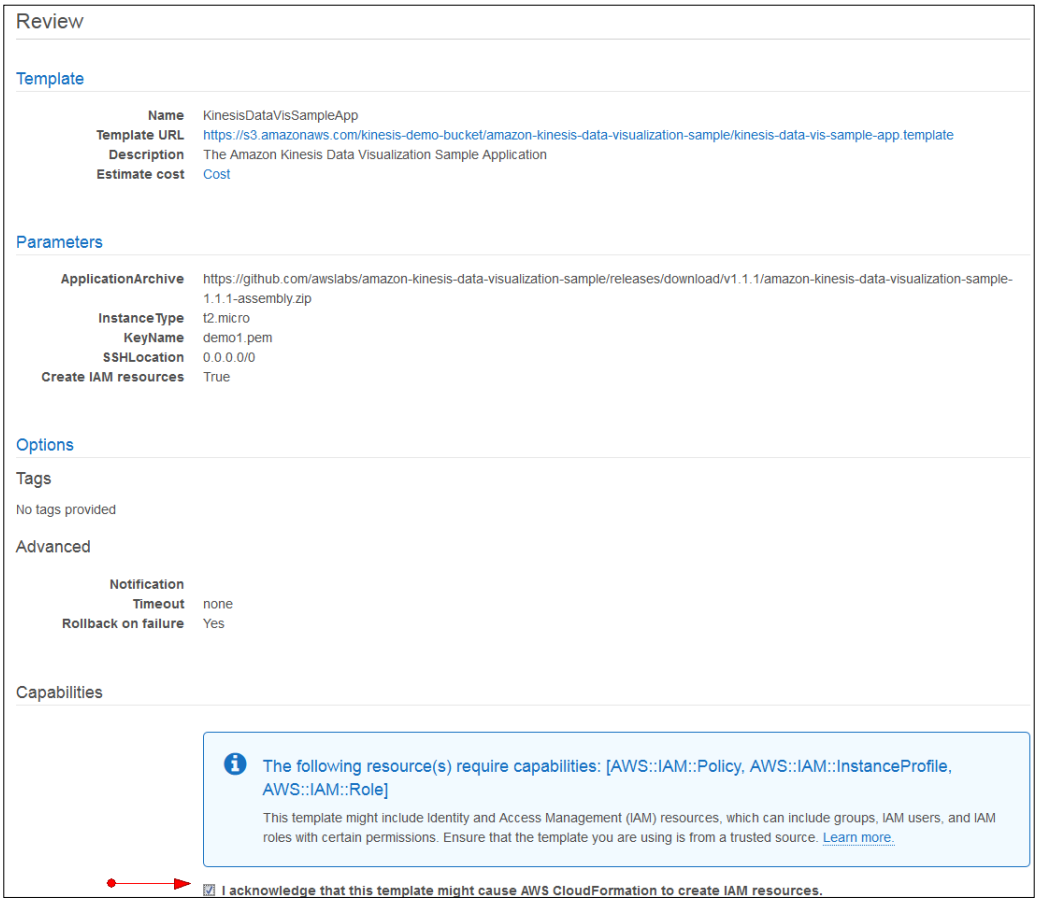
For example, we will form a data visualization model application that establishes how to practice Amazon Kinesis for real-time data ingestion and analysis. The model application fashions a data producer that positions pretend visitor counts from several URLs hooked on an Amazon Kinesis stream. The stream strongly provisions these data records in the order acknowledged. A data consumer gets these records from the stream, and then estimates how many visitors initiated from a precise URL. To conclude, a simple web application samples the outcomes in real time to deliver a conception of the designs.

To get you started quickly, the sample application uses AWS CloudFormation. AWS CloudFormation sanctions you to produce sample templates to designate the AWS resources and any supplementary dependencies or runtime parameters essential to run your application. The sample application practices the sample template to produce all the essential resources swiftly, containing producer and consumer applications running on Amazon EC2 and a table in Amazon DynamoDB to stock the combined record amounts:

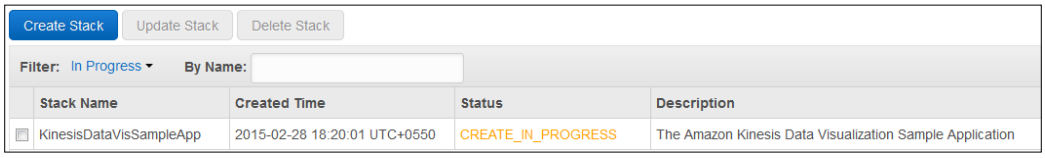
1. Go to AWS CloudFormation service dashboard and give the template URL as given here (please try with m3.large instance): `https://s3.amazonaws.com/kinesis-demo-bucket/amazon-kinesis-data-visualization-sample/kinesis-data-vis-sample-app.template`.
2. Once you provide the above URL and click **Next**, you will get the subsequent screen:



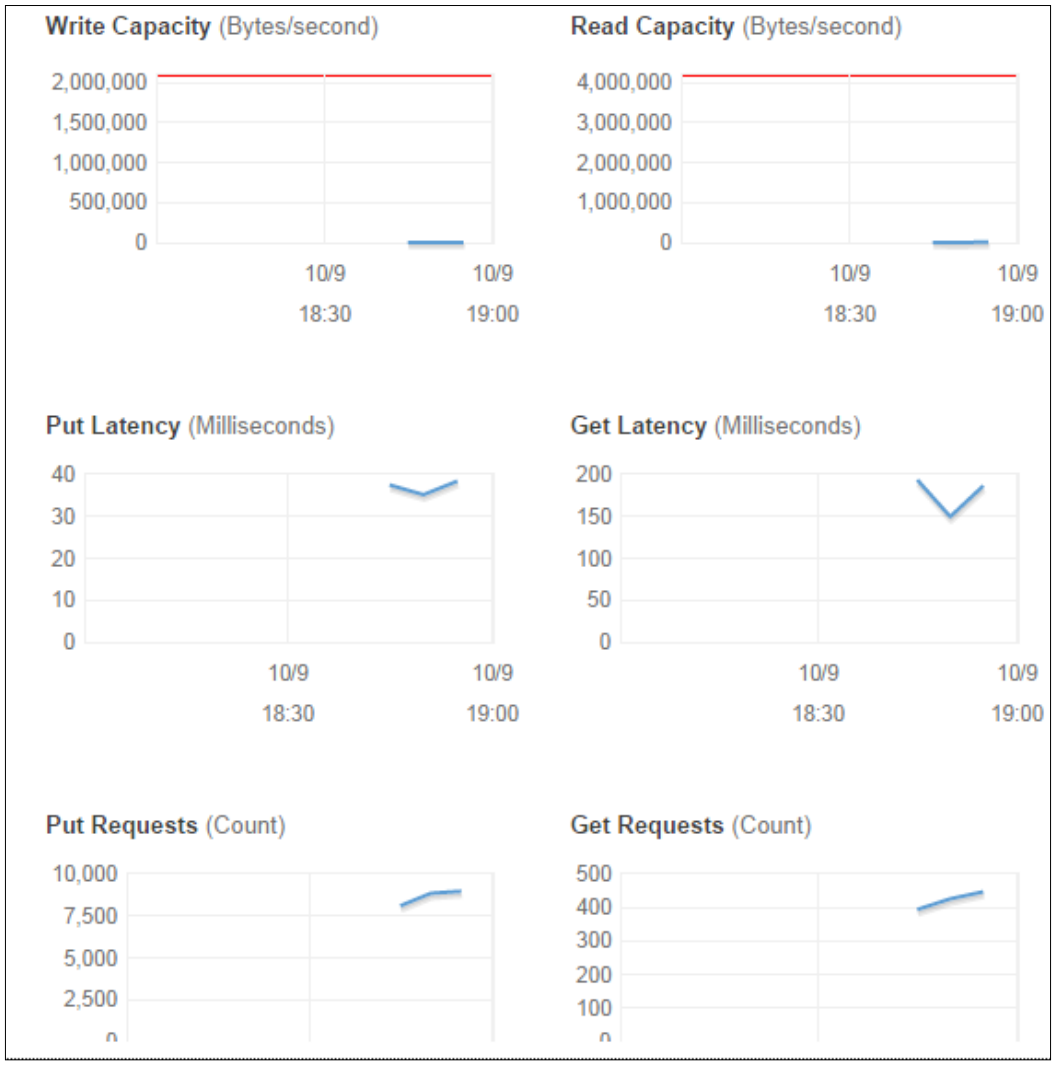
3. Click on **Next** after specifying the necessary parameters. You will see the following review screen where you have to click on the check box to acknowledge that IAM resources will be created on behalf of you. Finally, click on the **Create** button:



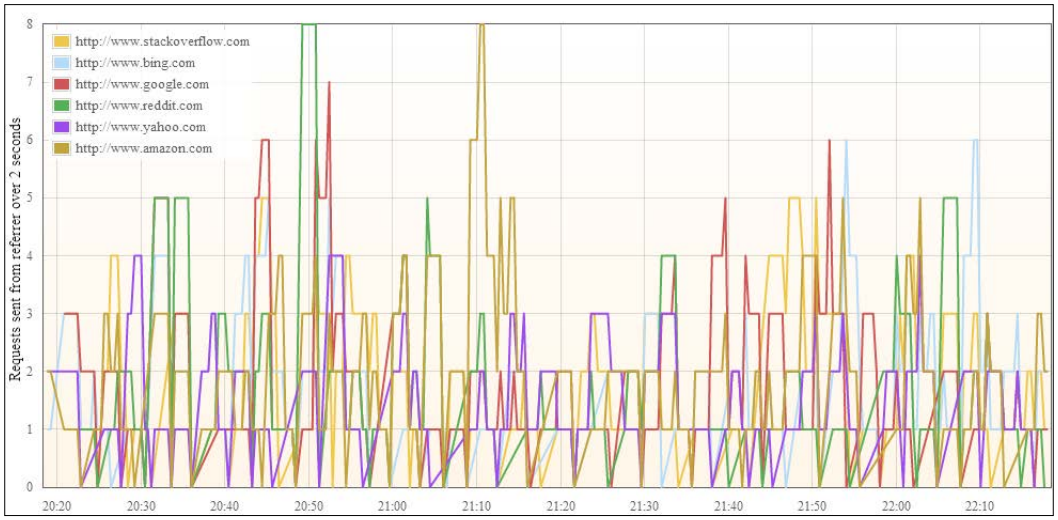
4. After clicking on the **Create** button, you should see a stack with the status **CREATE_IN_PROGRESS**. The stack may take several minutes to create, but when the status is marked **CREATE_COMPLETE**, go on to the next step:



5. Once this is done, on the **Stream List** page, you should see a stream with two shards and a status of **ACTIVE**. The stream name initiates with `KinesisDataVisSampleApp-KinesisStream-[RandomString]`. Click on the link to the stream to open the stream details page.
6. Review the stream statistics in the following graphs:



7. To view the real-time data analysis graph, go to AWS CloudFormation service dashboard and select your template. After selecting, go to the **Output** tab to see the graph like the following one:



This model application establishes the communal stream processing use case of carrying out a sliding window investigation over a 10-second gap. The data shown in the overhead visualization imitates the fallouts of the sliding window analysis of the stream as a constantly modernized graph. To sum up, the data consumer accomplishes Top-K analysis in excess of the data stream to work out the top three referrers by amount, which is presented in the table instantaneously underneath the graph and modernized every 2 seconds.

Summary

We started the chapter by introducing Big Data and Hadoop. After getting an overview of Hadoop, we created an EMR cluster with example. Then, we learned about Hive. Hive brings in SQL and RDBMS concepts to Hadoop through its query language, HiveQL. Finally, you learned about the basics of Kinesis and how to fashion a sample application with the help of the CloudFormation template on Amazon Kinesis.

In the next chapter, we will be looking at advanced services administration and programming with Amazon CloudSearch and Amazon Mechanical Turk. Also, we will see what kind of security AWS is providing and how to use those security features at the infrastructure and application level.

12
Miscellaneous Features, AWS Security, and Troubleshooting

In this chapter, we will discuss advanced services for administration and programming with CloudSearch and Mechanical Turk. Also, we will try to understand what kind of security AWS is providing and how to use those security features in the infrastructure and application level. At the end of this chapter, we will discuss some common troubleshooting practices. The list of topics that will be covered in this chapter is as follows:

- Leveraging Amazon CloudSearch and Mechanical Turk
- AWS security features
- Troubleshooting practices

So, let's start our journey in this chapter with the Amazon CloudSearch service.

Amazon CloudSearch

Search is the most perilous unit of many online verticals, such as travel, e-commerce, classifieds, and so on. If users cannot search applications/products proficiently, they will not make their buying verdicts appropriately, which, in turn, immensely touches the revenues of those organizations. Most of the search functionalities are generally provided by some of the open source or licensed products such as Apache Solr, FAST, Autonomy, ElasticSearch, and so on.

Amazon CloudSearch is a completely managed service from AWS that is easy to start and manage and makes it easy to scale your searching result for an application. Amazon CloudSearch has the capabilities to search a large set of data, such as web pages, documents, forum articles, and product details. You can set up search capabilities without having prior expertise of search or worrying about hardware setup and maintenance. As your size of data and traffic varies, Amazon CloudSearch scales to meet your requirements dynamically. To start with Amazon CloudSearch, simply execute the following steps:

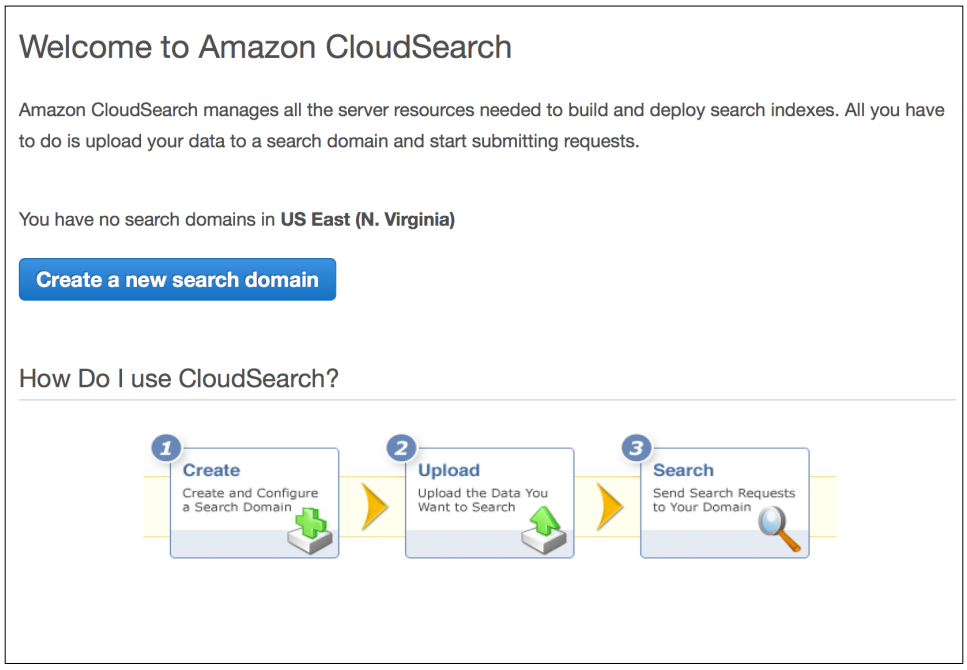
1. Create and configure a search domain.
2. Upload and index the data for search.
3. Send search requests to the domain.

Let's start with Amazon CloudSearch step by step. So, to start with Amazon CloudSearch, the steps are as follows.

Creating and configuring a search domain

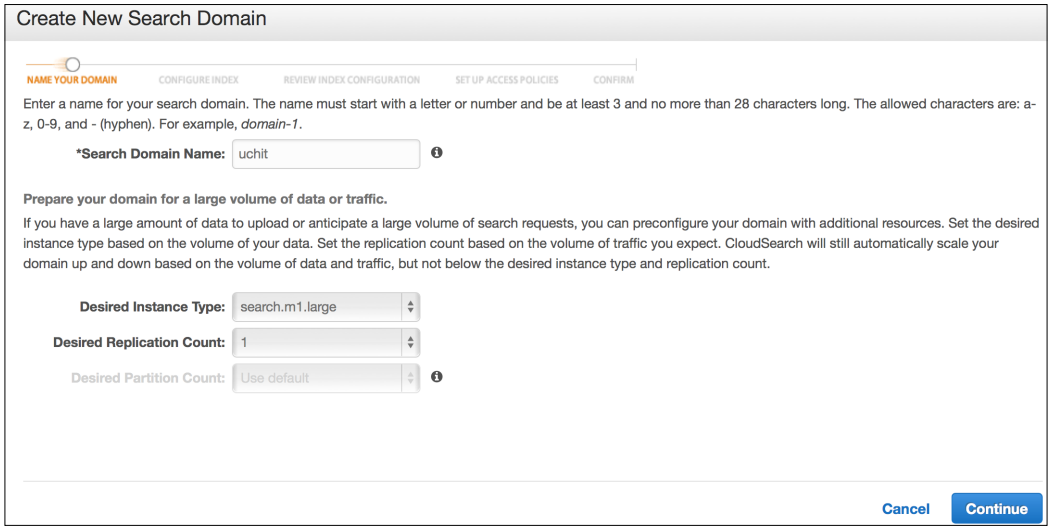
To create and configure a search domain, follow the subsequent steps:

1. Log in with AWS and select the Amazon CloudSearch service. You will see the following screen:

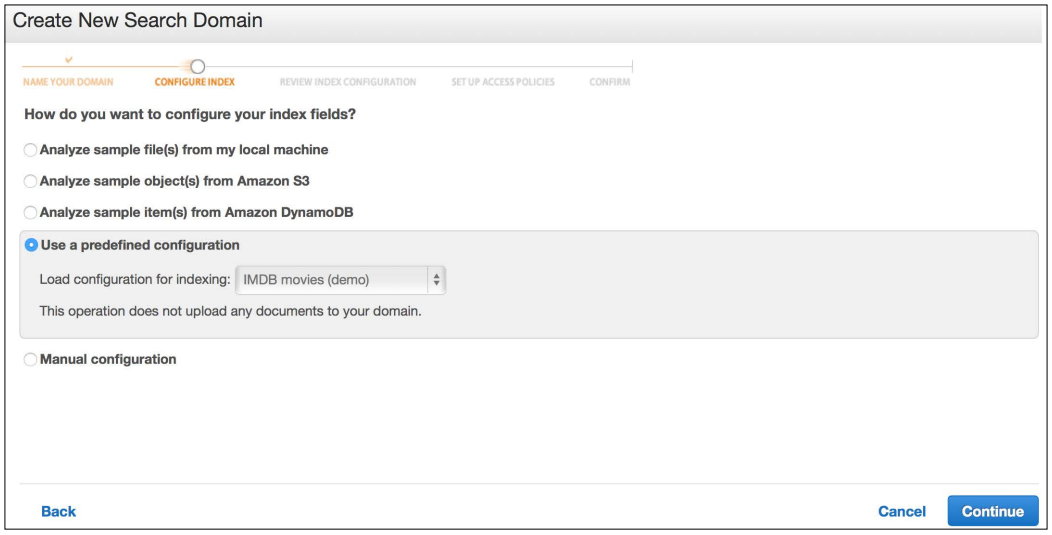


2. Click on the **Create a new search domain** button and you will get a domain creation popup, as shown in the following screenshot. Here, provide a meaningful name, instance type, and replication count as required.

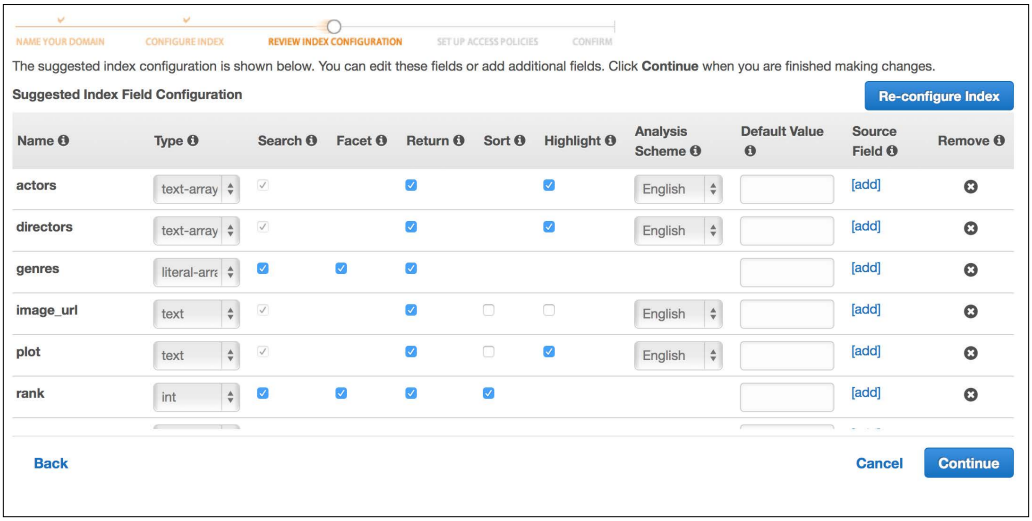
A domain name has to start with a letter or number and should be at least three and not more than 28 characters long. A domain name can have the subsequent characters: a-z (lower case), 0-9, and - (hyphen). Upper case characters and underscores are not acceptable.



3. In the **CONFIGURE INDEX** tab, choose **Use a predefined configuration**, select **IMDB movies (demo)**, and click on **Continue**.

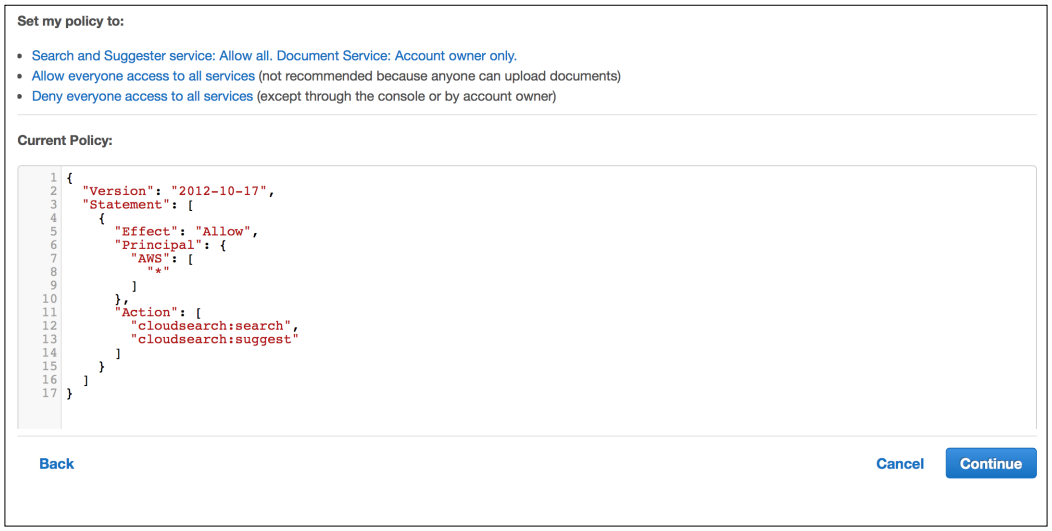


4. In the **REVIEW INDEX CONFIGURATION** tab, check that the index fields are configured properly or not. Automatically, 11 fields will be configured automatically for the IMDB movie data with the following fields: **actors**, **directors**, **genres**, **image_url**, **plot**, **rank**, **rating**, **release_date**, **running_time_secs**, **title**, and **year**. This is the default example provided by AWS, so you will definitely get these fields.

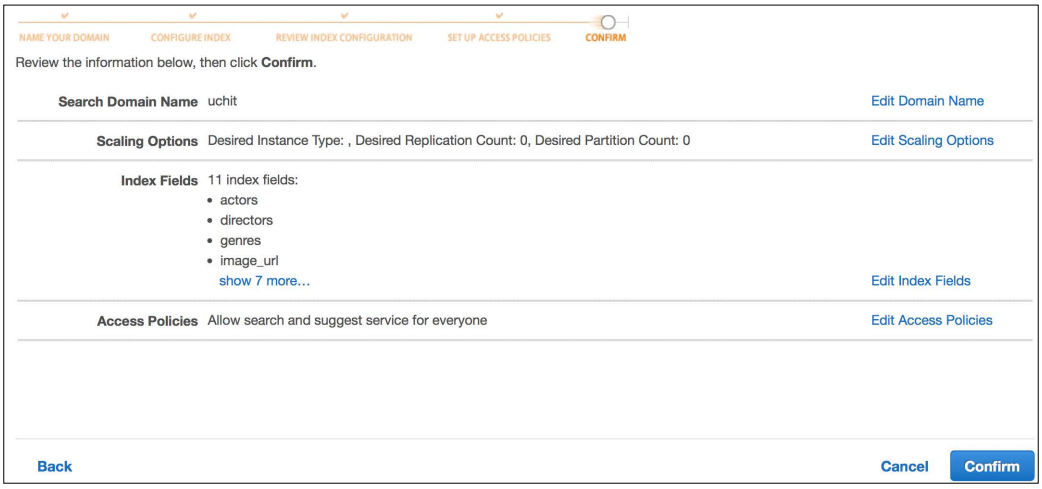


5. In the **SET UP ACCESS POLICIES** step, click on **Recommended rules**, and then on **Continue**.

   The recommended rules will allow access to the search endpoint from all IP addresses given in the policy as of now, and restrict access to the document service for the IP address you specify (not set right now in the given policy).



6. In the **CONFIRM** tab, review the configuration and click on **Confirm** to start the process of domain creation.



7. Once the domain has been created, click on the **OK** button to exit and go to the domain's dashboard to check the domain. The domain creation process will usually take a few minutes, around 10 to 12. Once you get the status **ACTIVE**, you can upload your data and start using the Amazon CloudSearch functionality.

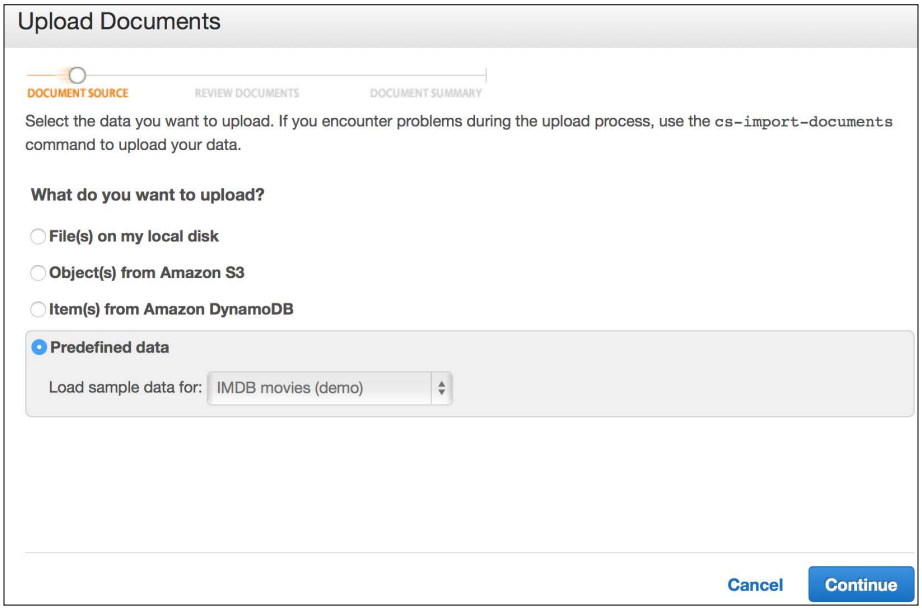
Uploading and indexing the data for search

Data has to be indexed by Amazon CloudSearch. The given data must be formatted in either JSON or XML. If the given data is in a different format than JSON or XML, it will be automatically converted to that format. The following data types are supported in Amazon CloudSearch: `.csv`, `.pdf`, `.htm`, `.html`, `.xls`, `.xlsx`, `.ppt`, `.pptx`, `.doc`, `.docx`, and `.txt`.

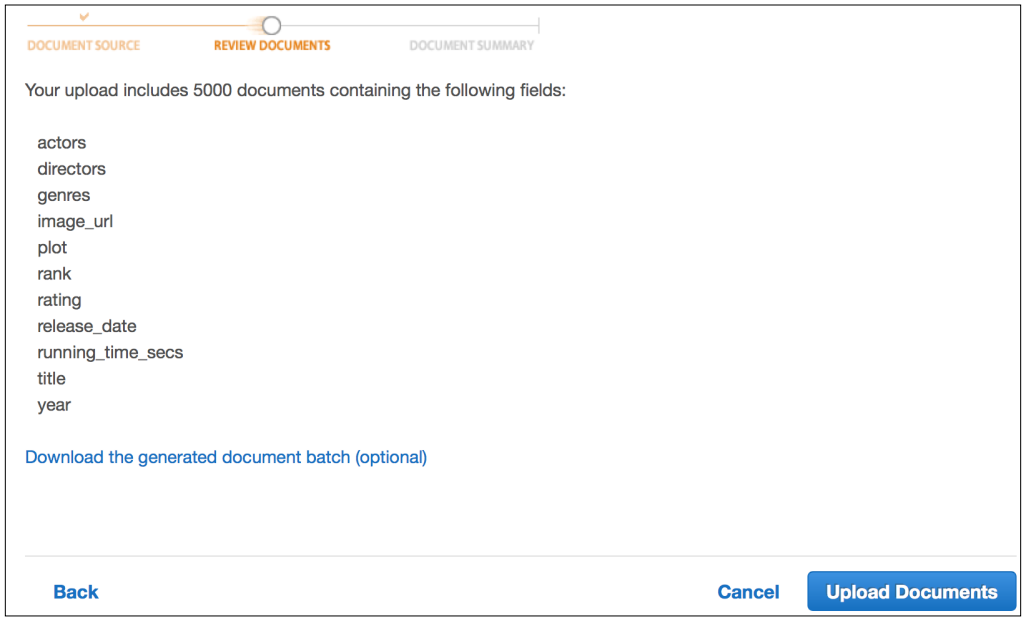
In the previous example, the sample IMDB movies data is already formatted in the JSON type.

To upload data, perform the given steps:

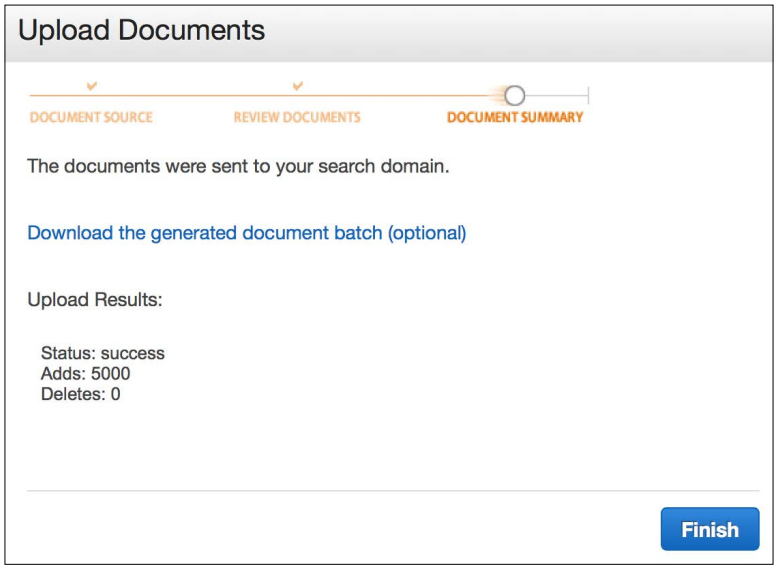
1. In the **Navigation** pane, click on your newly created domain, and at the top of the page, you will find the **Upload Documents** button. Click on it and you are good to go.
2. In the **DOCUMENT SOURCE** tab, choose **Predefined data**, select **IMDB movies (demo)**, and click on **Continue**:



3. In the **REVIEW DOCUMENTS** tab, check the upload summary and proceed further:



4. Finally, on the **DOCUMENT SUMMARY** page, click on the **Finish** button to complete the process:



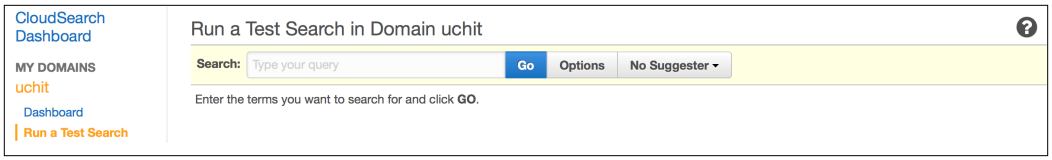
So, here you configured the search domain, which is fully functional and ready to search. Now, it's time to search your domain using Amazon CloudSearch.

Searching your Amazon CloudSearch domain

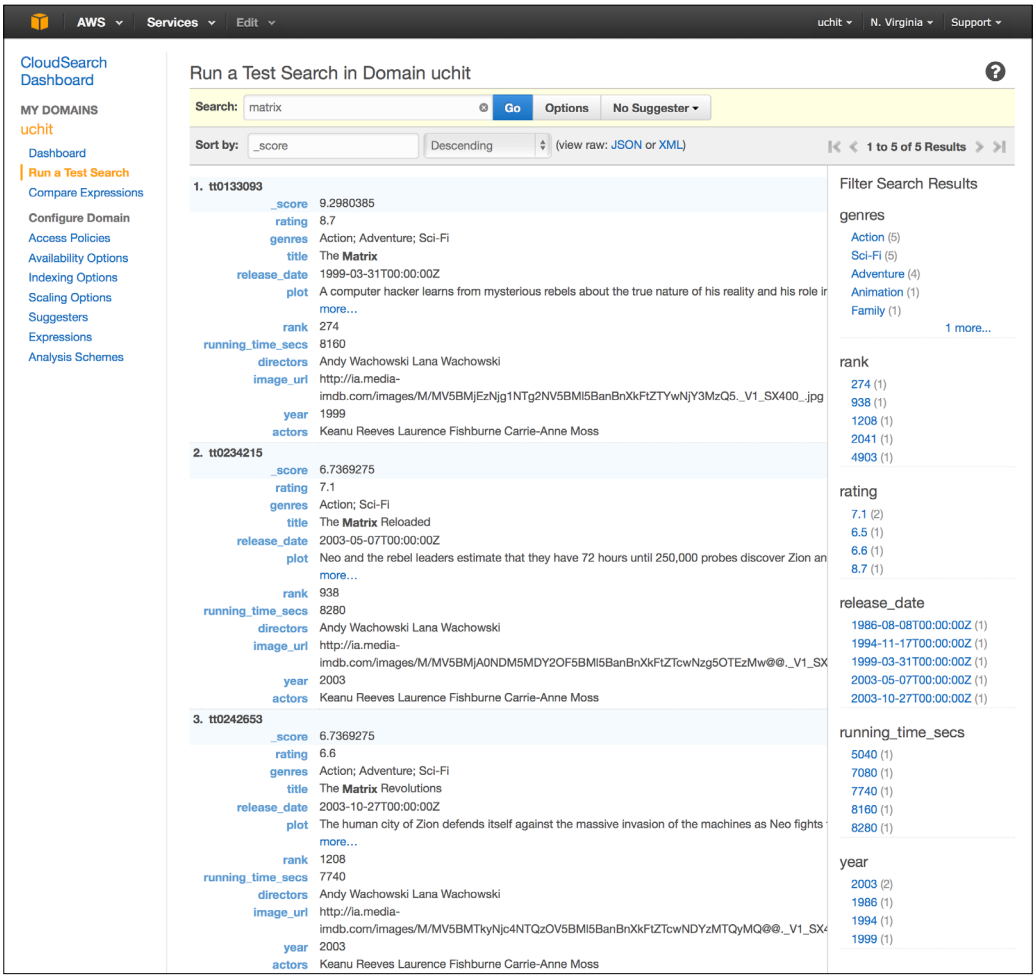
You can use the search tester on the Amazon CloudSearch console to start with a sample search. You can also do the search process through a web browser or using cURL.

Let's try to search with the tester here. By default, search requests are handled with a simple query parser. You can identify options for the selected parser, filter, sort the results, and browse the configured facets. To start with the tester, execute the following steps:

1. Go to the **Navigation** pane and click on the **Run a Test Search** link:

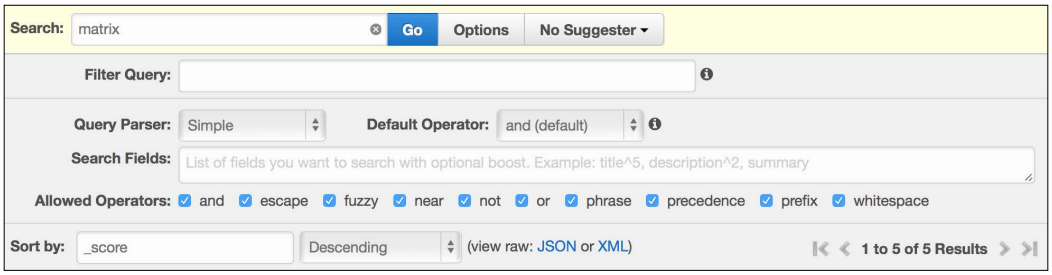


2. In the search box, add the term that you want to search and click on the **Go** button. For example, enter a word `matrix` and you will get the following results:



Domain search result

3. You can edit your search option by clicking on **Options** button and can get specific results as required:

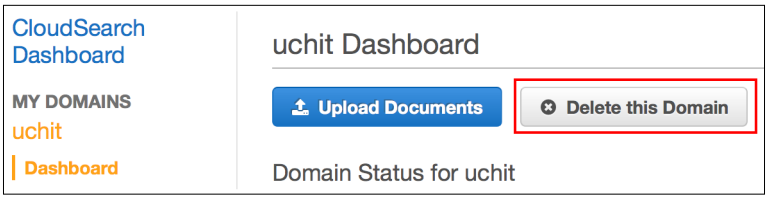


4. By selecting the **Query Parser**, you can select the type for your query to search. By default, you will get the following options:
    - **Simple**
    - **Structured**
    - **Lucene**
    - **DisMax**

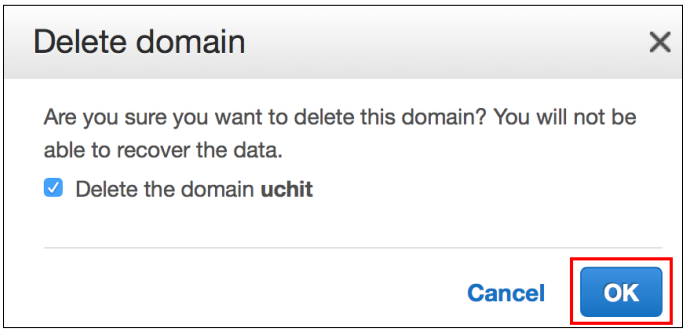
The preceding example is the simplest example to start with Amazon CloudSearch. To learn more examples, you can check `http://docs.aws.amazon.com/cloudsearch/latest/developerguide/getting-started-search.html`.

Finally, to remove your domain from Amazon CloudSearch, execute the following steps:

1. Go to the top of the domain dashboard and click on the **Delete this Domain** button:



2. You will be prompted with a dialog box for confirmation of deletion. Click on the **OK** button and the deletion process will start:



It can take around 10 to 15 minutes to delete a domain and its related resources from Amazon CloudSearch completely.

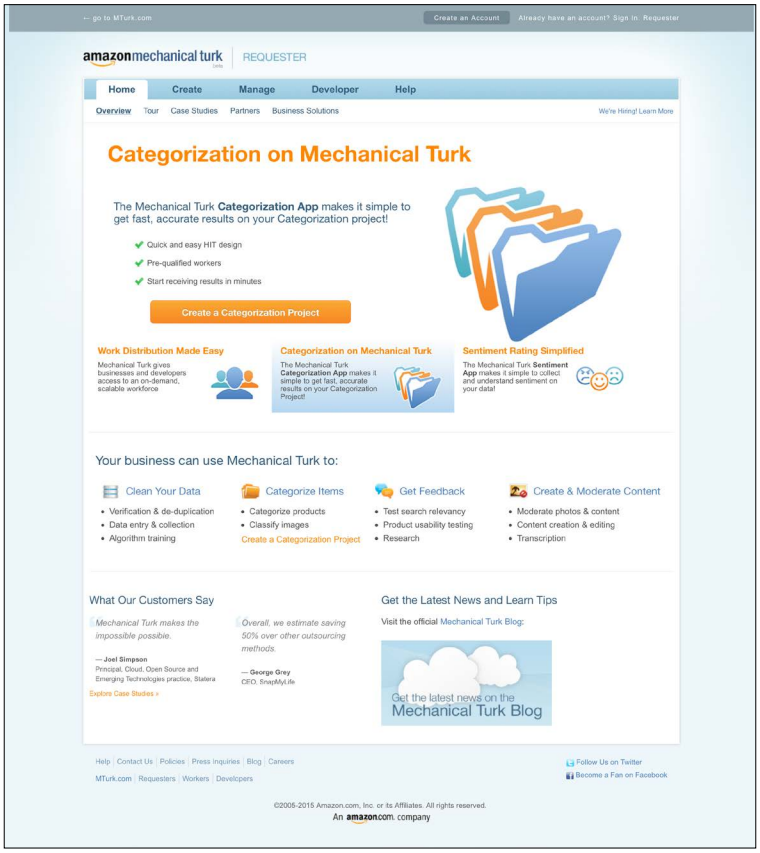
Amazon Mechanical Turk

Mechanical Turk is an Amazon service where individuals can publish tasks that are then executed by other people. The distinct tasks are called HITs. The Requester publishes a task such us filling in a survey. The Workers can view and accept the task, and upon finishing the task, will get paid the extent specified by the HIT explanation. So, digging from the origins, what you require is this:

- **Boto library (2.0b4)**: You can download this package from `https://github.com/boto/boto`
- **AWS keys**: You will need the access key and secret key for this

To start with Amazon Mechanical Turk, follow the ensuing steps:

1. Log in with an AWS account on `https://requestersandbox.mturk.com/` to create the MTurk sandbox account:



The MTurk sandbox account page

2. To connect the MTurk engine, follow the ensuing code snippet (you'll also find the snippet in the `3623EN_12_01.txt` file in the code bundle):

```
from boto.mturk.connection import MTurkConnection
ACCESS_ID ='put access key here'
SECRET_KEY = 'put secret key here'
HOST = 'mechanicalturk.sandbox.amazonaws.com'
umtc = MTurkConnection(aws_access_key_id=ACCESS_ID,
                       aws_secret_access_key=SECRET_KEY,
                       host=HOST)
print umtc.get_account_balance()
```

3. Now, once you run the preceding code snippet, you will get the result with a value such as $ 10,000.

4. Once you have completed the previous steps, it means you are all set to create a HIT. Fundamentally, a HIT is a question or an assortment of questions. Let's try to create a HIT with the initial two questions in which one will be mandatory and the other will be optional. Execute the following code snippet for it, which you'll also find in the `3623EN_12_02.txt` file:

```
title = 'Offer your view roughly about this website'
description = ('Call a website and give your thoughts
about the design layout and also some personal comments')
keywords = 'website, rating, thoughts, views'

ratings =[('Very Bad','-2'),
         ('Bad','-1'),
         ('Not bad','0'),
         ('Good','1'),
         ('Very Good','1')]
#---------------BUILD OVERVIEW ------------------
overview = Overview()
overview.append_field('Title', 'Give your opinion on this
  website')
overview.append(FormattedContent('<a target="_blank"'
                                 ' href="cloudbyuchit.
  weebly.com ">'
                                 ' Uchit Vyas private
  site</a>'))
```

The overview is a free content of the question form, preceding processes (for example, binary texts, HTML texts, and so on). Consider that it isn't a "question object", it is just an arbitrary content. Take a look at `3623EN_12_03.txt` for the entire code:

```
#---------------CONSTRUCT QUESTION ONE--------------
mqc1 = QuestionContent()
mqc1.append_field('Title','How this design looks like to you?')
wfta1 = SelectionAnswer(min=1, max=1,style='dropdown',
                      selections=ratings,
                      type='text')
qs1 = Question(identifier='design',
               content=mqc1,
               answer_spec=AnswerSpecification(wfta1),
               is_required=True)

#-------------CONSTRUCT QUESTION TWO---------------
mqc2 = QuestionContent()
mqc2.append_field('Title','Your particular inputs)
wfta2 = FreeTextAnswer()
qs2 = Question(identifier="comments",
               content=mqc2,
               answer_spec=AnswerSpecification(wfta2))
```

The question is mostly prepared by the `AnswerSpecification` object, which stipulates the type of answer to be extracted for the query by the `QuestionContent` object. The content can be the same for the overview type, text type, binary type content, and HTML type content. Check out `3623EN_12_04.txt` for a detailed version of the code:

```
#--------------CREATE THE QUESTION FORM--------
question_form = QuestionForm()
question_form.append(overview)
question_form.append(qs1)
question_form.append(qs2)
```

This is the container for all the listed questions and overviews, and ultimately, you can say an addition of a Python standard catalog. You'll find the following snippet in `3623EN_12_05.txt`:

```
#---------CREATE THE HIT AS FINAL STEP------------
umtc.create_hit(questions=question_form,
                max_assignments=1,
                title=title,
```

```
description=description,
keywords=keywords,
duration = 60*
```

This is the method to create HITs. After running this code all together, you will be able to create a HIT on MTurk. To practice MTurk on a deep level, you can try the PDF by AWS available at `http://s3.amazonaws.com/awsdocs/MechTurk/latest/amt-gsg.pdf`.

AWS Security best practices

Information security (**IS**) holds dominant significance to **Amazon Web Services** (**AWS**) clienteles. Security is a fundamental efficient constraint that defends perilous intelligence from unintentional or considered theft, escape, reliability compromise, and removal. AWS gives a variety of security functions and topographies that customers can use to be assured of their resources. AWS consumers are liable for keeping the privacy, integrity, and availability of their data in the AWS Cloud, as well as for meeting certain industry needs for information safety.

AWS offers secure global infrastructure and services in the cloud. You can build your systems using AWS as the foundation, and architect an ISMS (Information Security Management System, that is, a group of IS policies and practices for an association's resources on AWS) that takes advantage of AWS qualities. To plan an ISMS in AWS, you should be familiar with the AWS shared responsibility model, which engages AWS and customers to join efforts concerning security aims. To guarantee secure services, AWS provides a shared responsibility model for services:

- **Infrastructure services**: This type consists of AWS services, such as Amazon EC2, and related services, such as Amazon EBS, Auto Scaling, and Amazon VPC.
- **Container services**: The services in this category type normally run on individual AWS EC2 but sometimes you don't have to manage the OS or platform layer. AWS offers a managed service for application "containers". Examples of this type of services are Amazon RDS, Amazon Elastic Map Reduce (Amazon EMR), and AWS Elastic Beanstalk.
- **Abstracted services**: These types of services are Amazon S3, Amazon Glacier, Amazon DynamoDB, Amazon SQS, and Amazon SES. These services abstract the platform on which you can construct and manage your applications.

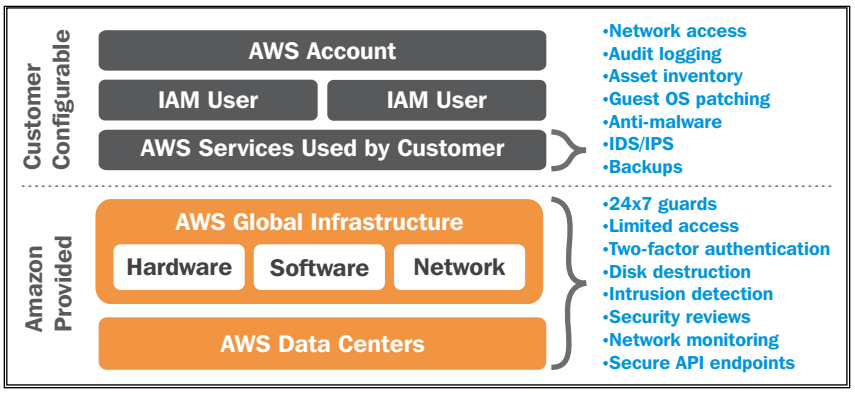
The shared responsibility model for infrastructure services, for example, AWS EC2, states that AWS will administer the security of the subsequent resources:

- Facilities
- Physical security of hardware devices
- Network infrastructure
- Virtualization infrastructure

To start with AWS Cloud security, you have to learn AWS Secure Global Infrastructure thoroughly.

Understanding AWS Secure Global Infrastructure

The global infrastructure and services are managed by AWS itself securely and offer a truthful base for business systems and distinct applications. Amazon establishes high standards for information security, for example, from physical security throughout software purchase and improvement to operative life cycle management and security of organization. Also, the secure global infrastructure is subject to systematic third-party compliance audits.



In this secure global infrastructure, IAM is also a vital part as it centrally directs users, security credentials, and policies that control AWS services and resources from end users and internal users within organization.

In the AWS global infrastructure, they give the following checks for physical security:

- Controlled, need-based access
    - All access is logged and reviewed
    - Multifactor authentication
- Separation of duties
    - Employees have physical access do not have logical access (access to internal data)
- 24x7 security guards for each building of datacenter

On the other hand, at the networking end, Amazon provides the subsequent securities:

- Standard mitigation techniques for DDOS (distributed denial-of-service)
- API points are secured by SSL
- IP spoofing is prohibited at host OS level
- Port scanning is prohibited
- Packet sniffing protection at hypervisor level and in promiscuous mode becomes ineffective

In the storage area, AWS provides the following features to ensure the level of security:

- All storage devices are degaussed and physically destroyed
- All the physical disks are wiped out after creation

For further reading on AWS security, third-party certifications, and compliance, you can refer to `https://aws.amazon.com/compliance/` and `https://aws.amazon.com/security`.

Regions, Availability Zones, and service endpoints

You already learned about regions, AZs, and endpoints as they are also key components in Global Secure Infrastructure. So, we will only glance through these terms:

- **Regions**: AWS regions work to manage the network latency and regulatory compliance. When you supply data in a particular region, it won't be replicated externally from that region automatically. You are the only person who is responsible for picking the appropriate region to store data with your compliance and network latency requirements. Every region consists of at least two Availability Zones, often more.
- **Availability Zones** (**AZs**): AZs are planned for fault segregation. They are associated with manifold **Internet Service Providers** (**ISPs**) and different power grids. You are the only person who is responsible for choosing the AZs where your applications will exist. Apps can extend in multiple AZs in the case of a disaster.
- **Service endpoints**: AWS provides web access as well as programmatic access to each service by service endpoint, which is managed by AWS.

Managing keys in the Cloud

You can practice on-premises **hardware security modules** (**HSMs**) or CloudHSM to provide a diversity of use cases, such as DB encryption, **digital rights management** (**DRM**), and **public key infrastructure** (**PKI**) involving authentication and authorization, and transaction handling. CloudHSM presently practices Luna SA HSMs from SafeNet. The Luna SA is intended to convene **Federal Information Processing Standard** (**FIPS**) 140-2 and Common Criteria EAL4+ standards, and also provisions a diversity of industry-standard cryptographic processes. Your apps can practice the regular APIs sustained by CloudHSM, for example, PKCS#11, MS CAPI, and Java JCA or JCE (Java Cryptography Architecture or Java Cryptography Extension). The CloudHSM client specifies the APIs for cloud apps and gears each API call by concerning the CloudHSM appliance with the use of a communally authenticated SSL assembly.

Managing patches

You are the only person who is responsible for patch management for AWS AMIs and running EC2 instances. I would like to recommend the freeze patch management process rigidly once you are ready and can maintain a written document.

Although you can use third-party vendor patch management systems for OS and other critical apps, it is a good habit to have an inventory of all software and apparatuses with patch versioning and details of the origin of source. Also, it is important to compare the catalog of security patches installed on every system with the most recent vendor security patch list to validate the latest patches that are getting installed. Moreover, AWS has occasionally held scheduled maintenance to upgrade EC2 and requires users (EC2 instances) to reboot their OS.

Mitigating compromise and abuse

On AWS Secure Global Infrastructure, abuse activities are superficially perceived activities of AWS customers' resources that are nasty, invasive, illegal, or could harm other Internet resources.

AWS works with you to identify and address mistrustful and malicious happenings from AWS resources in your account. Unanticipated or doubtful behaviors from your account resources can signify that your AWS assets have been compromised, which indicates potential risks to your commerce. AWS practices the subsequent means to discover abuse activities from consumer assets:

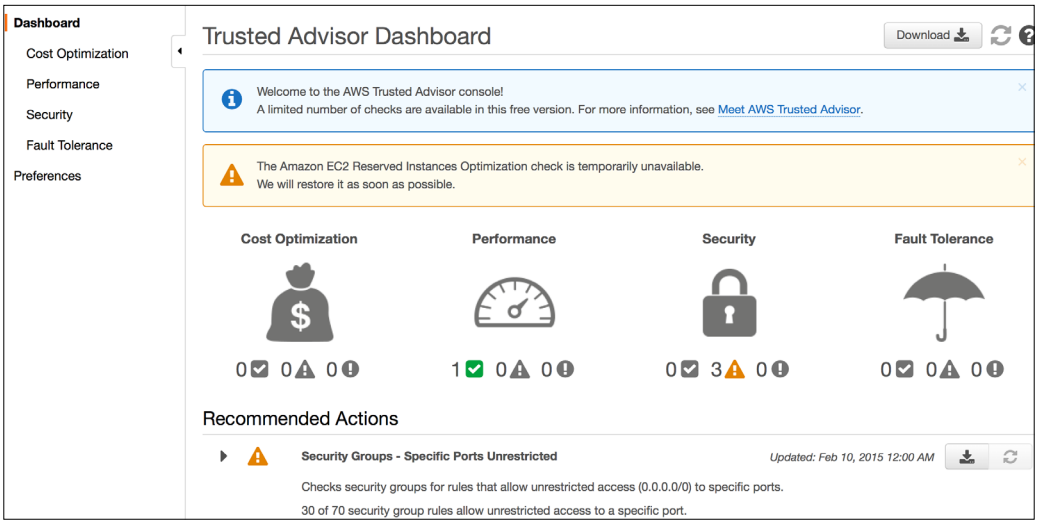
- Interior event monitoring
- Exterior security intelligence in contradiction of the AWS network space
- Internet abuse grievances in contradiction to AWS assets

Malicious, illegal, or harmful actions that make your AWS assets infringe the AWS standard use policy and can be an indication of account suspension. If the consumer disappoints to report abuse activities, AWS will suspend the account to defend the integrity of the AWS Global Secure Platform.

The Trusted Advisor tool

AWS support plans contain access to the Trusted Advisor tool as a service as of now, which suggests a single window snapshot of your AWS services and assists to recognize usual security misconfigurations, recommendations for cultivating system performance, and underutilized resources.

The following is the screenshot for the Trusted Advisor service for your reference:



So, from the Trusted Advisor service, you can dig your resources configuration and the entire loop for security breach.

Troubleshooting practices

Until now, we have seen AWS services, their workings, and, mostly, the integration of these services with each other. Now it's time to see some of the basic errors and how to resolve them in a simple manner. So, let's start our journey with this.

Ephemeral disk corruption

When working an instance with ephemeral disks, your ephemeral disks can become corrupt. The filesystem will become read-only, and you will be incapable to write to disk. Messages will also display up in `dmesg` allied to filesystem corruption. A possible solution is to try to detach your root EBS volume and attach it to a new instance or launch an instance again to get your vital data back. Even you can follow the AWS blogs; to check the EBS volume status, check at `https://aws.amazon.com/blogs/aws/ebs-volume-status-checks/`.

DNS concerns

It is conceivable for the internal DNS resolver on the hypervisor to crash, which produces DNS lookups to collapse from within the instance. A possible solution is to alter your DNS resolver, and try to re-launch your instance storage-backed instance and start/stop an EBS backed instance.

Resizing or emptying disks

While one of the cases with AWS is that you can scale up as needed, it is a calamitous truth that ephemeral disks don't do scaling automatically. You can upsize EBS but not automatically. You can check how filled your disk is by inspecting disk usage metrics. A possible solution is to change the size of your disk or delete idle files.

Host dispute

At the time of allocating server space, Amazon will frequently oversubscribe a server under the postulation that all consumers won't use the full ability. Exercise a command such as follows:

```
iostat 1
```

You can extend the volume of CPU steal your instance is facing. High CPU steal is commonly a sign of piercing neighbors. A possible solution is to try to re-launch your instance storage-backed instance and start/stop an EBS backed instance.

Security group misconfiguration

Misconfiguration of security groups can produce glitches, such as timeouts, 50x errors, or application unreachability. You should remember to expose the local firewall and port internally on an instance also, but forget to open it in the EC2 security group. A possible solution is to open an appropriate port on the security group to reach an application smoothly.

AWS can be tough to administer, but when done accurately, it isn't difficult to understand why it's a dynamic tool for so many different industries.

Summary

In this chapter, we discussed some advanced services for better administration and programming, such as AWS CloudSearch and Mechanical Turk (MTurk). Also, we have gone through the security features provided by AWS and how to use those security features at the infrastructure and application level within your Amazon account. At the end of this chapter, we have seen some common AWS services problems and troubleshooting practices.

13
Building Applications and AWS Best Practices

We discussed most of the interesting and beneficial AWS offerings in the previous chapters. Now, it is time to integrate all of those services in a single application as a use case and the best practices. Here, we will create a web app that will be a mixture of AWS services. In this chapter, we will focus on the following topics:

- Overview of the application
- Software and tools requirement elicitation
- Implementation and management of application
- AWS best practices and design solutions

Application impression

We will call the application that we will develop and maintain using AWS as EduCloud. The app will grant AWS EC2 instances to its registered consumers. We will use **Java Server Faces** (**JSF**) to create the web app. So, the user registering page, login page, dashboard, and every web page will be coded using JSF. In this app, a user will catalogue himself to the EduCloud with his cell phone number and e-mail address. During the registration, an SNS topic will be created with his/her username as the topic name. Both the cell phone number and e-mail address will be subscribed to this topic. Once the admin will approve the request of a new requester, the user can request for an EC2 instance (or our custom AMIs).

Even if the user requests the instances in VPC, we will provision the same. As a database, MySQL (RDS) will be used to store this information. Once the instance request has been approved by the administrator, an SMS and e-mail will be sent to the respective endpoints about the connection details to the instance via SES and SNS. As the number of users and instance requests exceeds, there is a possibility that some SNS messages are lost, so we will add those to the SQS queues. We will store the instance key pairs in S3 once the user downloads it; we will delete the same from S3. Lastly, we will see how this application can be deployed on the Elastic Beanstalk container service.

Tool mixture

We don't need any precise software tools to build our app. We will practice on the following freeware (an open source software) tools:

- Eclipse IDE (as the development environment) with the AWS plugin setup
- JDK 7 and Tomcat 7.0 (to run the web application locally)
- MySQL connector (to connect to our RDS instance)
- Puttygen (to create the `.ppk` file from the `.pem` file)
- Putty (to connect to the EC2 instances)

All the preceding software is open source and available over the Internet.
Since we have already discussed most of this software in the previous chapters, we will move to the core part, which is obviously the "development".

Development phase

It's time to get our hands dirty by creating the first AWS app, which will hold many AWS services. It will be a JSF app, so rather than talking about the adornment of the JSF page and how to mark the connection with the MySQL database, we will have to concentrate only on the code where we will perform the AWS operations. And, of course, yes, this app can be downloaded from the official Packt website.

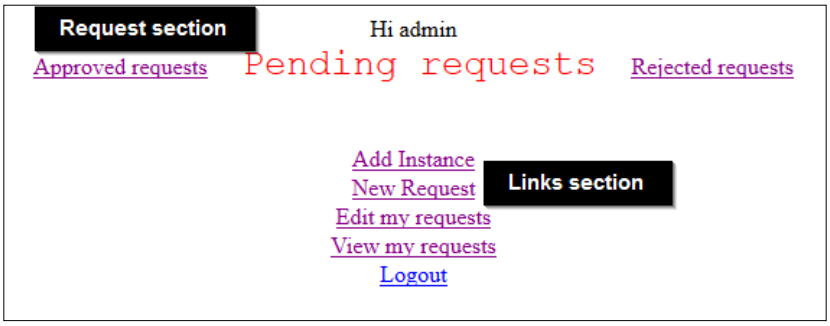
Conventions

For the appropriate working of the app, the subsequent conventions must be made. We can even call the following points as prerequisites. Even if one of the following points is missing, the whole app will fail:

- We should have an S3 bucket with the name `my-keypair` to save the EC2 keys.
- An RDS instance must have an `education_cloud` database with admin privileges.
- The AWS credentials file must be configured with the admin's root AWS account.
- The method `publishInstanceCreation` of the `SNSoperations` class should have an accurate subscriber. In the app, it will be hardcoded as `016883241246`. This must be replaced with a proper value. AWS SNS will notify the users and admin if an event occurs.

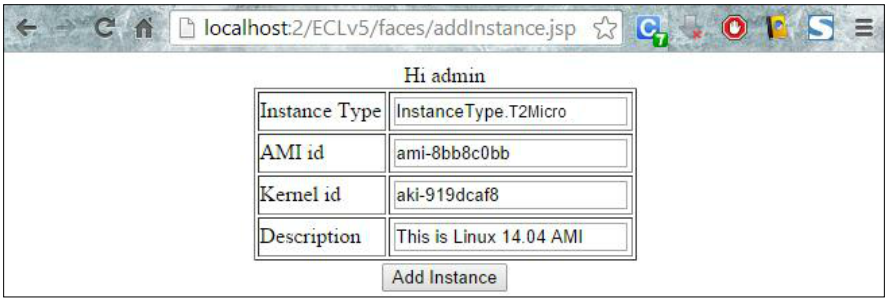
Handlers

This app has two kinds (roles) of handlers, namely the admin and end consumer (normal user). The admin operator can achieve instance requests. His/her dashboard is shown as follows. It consists of two segments. The first segment is the **Request section** in which he/she can approve, reject, or waitlist a group of instance requests. This unit consists of three links, each to navigate to the **Approved requests**, **Pending requests**, and **Rejected requests** pages. The second sector is the **Links section**, which has links to add new instances to be made available for the consumer. Other links are used to test whether the app is running, by requesting the instance and approving it; thereby checking the unabridged workflow.



Our app has another role, which is no-admin (can be apprentice or educator). We will call this handler customer. A non-admin user's homepage will have links to request a new instance and view and edit the present requests.

An instance is a collection of instance type, AMI ID, and kernel ID. These possessions are appealing much perilous, as choosing improper mishmash will result in an exception. In our app, it will be the concern of the admin to adopt instance configuration combinations because it requires proficiency. This is a crucial operation of our app. Without adding an instance, the consumer cannot request for any instance. The following screenshot will give you the hint for the **Add Instance** page. The first parameter will be the instance type, which starts from the lower end (of course, with free tier eligibility) t2 micro up to i2 8xlarge. If we want to add information such as RAM disk size, we can specify it in portrayal. Then, we can click on the **Add Instance** button, which will augment this info to the MySQL database.



Starting with EduCloud

On the login page, there will be a link to sign up. The sign up page will ask for subsequent information:

1. Enter the essential personal information (first five fields). The other two fields are used to create login credentials:

2. Once we click on the **Create Account** button, a user's personal details will be added to the `Customer` table and login details will be added to the `Login` table.
3. In chorus, an SNS topic will be created by our app with the username as the topic name. In the preceding screenshot, we registered a user with the name `uchit86`. So, an SNS topic will be created as topic name so that the user will be notified about his request update.
4. The code for topic creation is shown as follows:

```
public void createInstanceTopic(String topicName, String
  mailId,
    String mobileNumber) {
  CreateTopicRequest topic = new CreateTopicRequest();
    topic.setName(topicName);
    topic.setRequestCredentials(credential);
  String topicArn = client.createTopic
    (topic).getTopicArn();
  createInstanceSubscription(topicArn, mailId,
    mobileNumber);
  }
```

EduCloud uses four tables in total. The `Customer` table stores the personal information quantified for the duration of the signup process. The `Login` table stores the EduCloud login information. The `Instance` table stores information about instance parameters such as AMI ID, kernel ID, instance type, and so on. The fourth table, `Request`, has information about the instance requested by the user and its status.

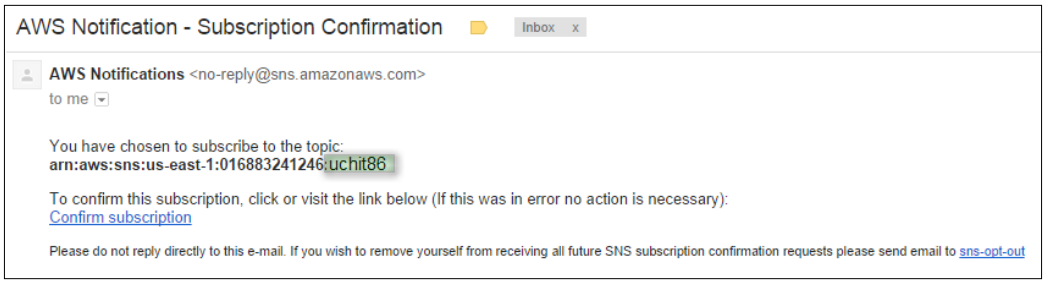
The function `createInstanceTopic` takes three parameters. The first parameter is `topicName`, which is the same as that of the username, and the rest is `mailId` and `mobileNumber`, which is obtained from the personal information entered during the signup process. The preceding code generates a topic, and `topicArn` along with `mailId` and `mobileNumber` are passed to the `createInstanceSubscription` method (whose implementation is shown as follows):

```
private void createInstanceSubscription(String topicArn, String
mailId, String mobileNumber) {
  SubscribeRequest subscribeRequest = new
    SubscribeRequest();
    subscribeRequest.setTopicArn(topicArn);
    subscribeRequest.setEndpoint(mailId);
    subscribeRequest.setProtocol("email");
  client.subscribe(subscribeRequest);
  if (mobileNumber.contains("+1")) {
    SubscribeRequest request = new SubscribeRequest();
      request.setTopicArn(topicArn);
      request.setEndpoint(mobileNumber);
      request.setProtocol("SMS");
      client.subscribe(request);
  }
}
```

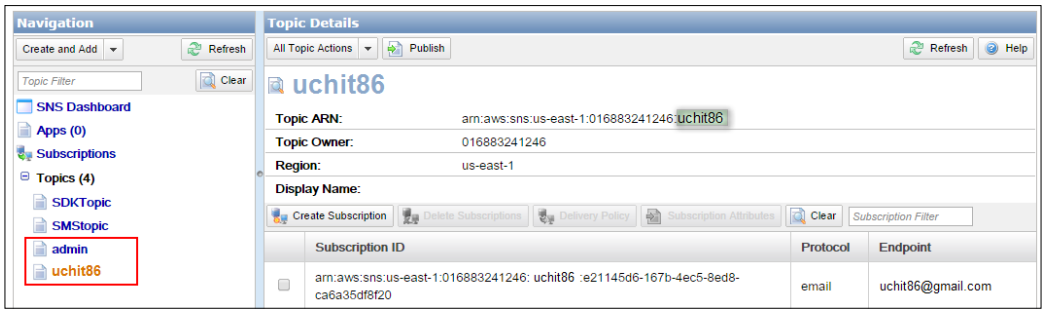
The preceding code subscribes the registered e-mail ID and phone number to the topic. SNS supports all the e-mail IDs, but *SMS notification is currently supported only for the US*.

5. As soon as we click on the **Create Account** button, the user will get a subscription mail for the topic.

6. Once the user clicks on the **Confirm subscription** link, the mail ID will be subscribed. The same is the case with the mobile number. If the mobile number has +1 (US), then a text message to confirm the subscription will be sent via an SMS.



7. We can validate the subscription by accessing the management console. Until now, there were two users in our app (`admin` and `uchit86`). The last two topics will serve as the determination for instance notifications.

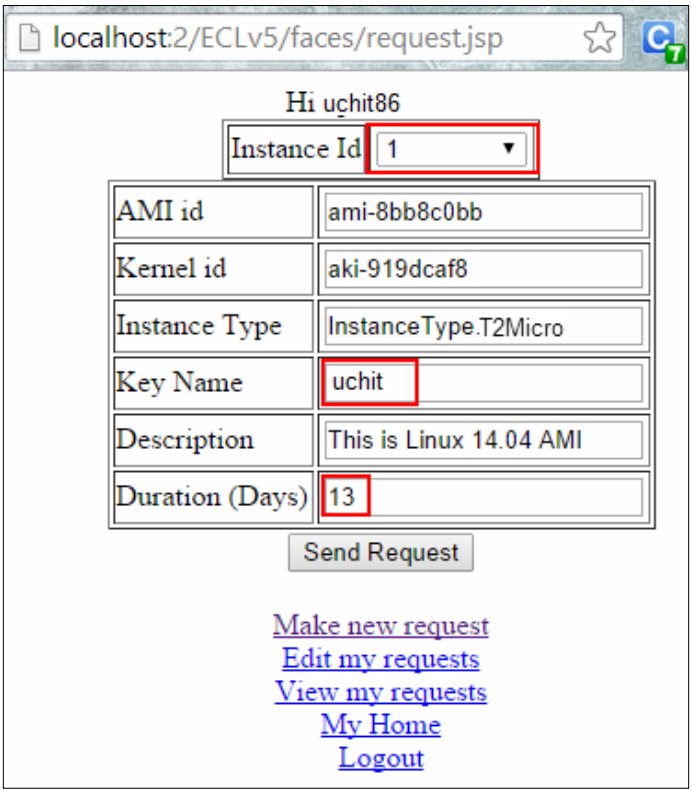


Handling instance entreaty

As of now, we configured parameters for an instance and registered a customer to our app; the user then has to subscribe to the notification.

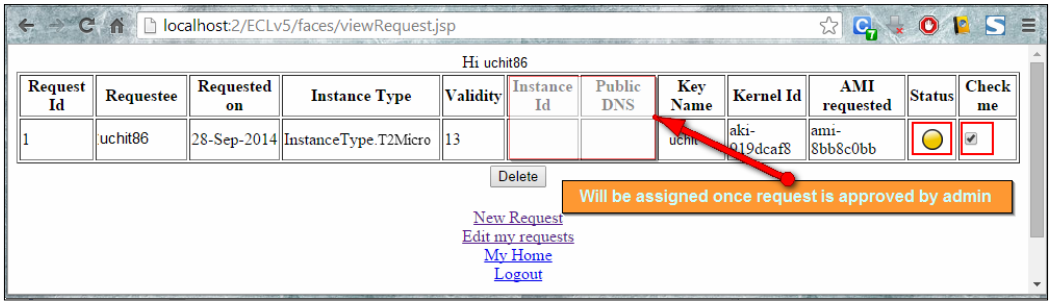
1. Once the user is done with the subscription, they can request for the available instances. The user can select one of the instances added by the admin using the **Instance Id** dropdown.
2. Once the user selects the instance ID, **AMI id**, **Kernel id**, **Instance Type**, and **Description** will be populated automatically.

3. In addition to this, the user can specify the private key pair **Key Name** to be created and the **Duration** value for which the instance is needed for practice.
4. Once we are done, we can click on **Send Request**. This will not produce any EC2 instance. The instance will be created and the connection details will be mailed (and the SMS will be sent) once the request is approved by the admin.



5. The same information will be available in the View Request page (shown in the next screenshot). Since the request is not yet approved, the **Status** column (of the following table) will show a yellow picture and the checkbox next to this field will be enabled.

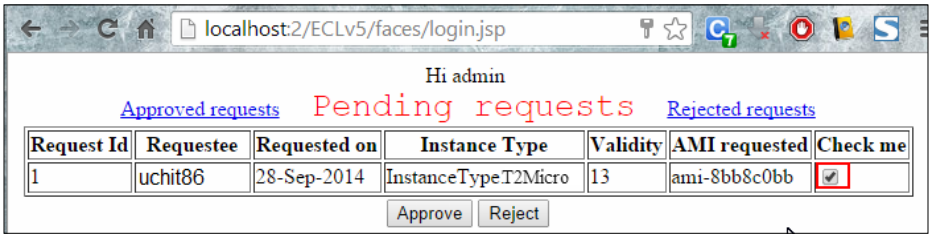
This will be enabled only if the request is in the pending state. Since the instance is not yet created, instance ID and public DNS will be empty as long as it is not approved by the admin.



Instance entreaty sanction

The next steps show how to approve or reject the instance request. The admin home page is the same as that of the pending requests page. It will show all the pending instance requests. The last field (column) of the table is a checkbox.

1. The admin can check this button and click on the **Approve** button. Moreover, in case of multiple requests, the admin can select multiple checkboxes.



2. Clicking on the **Approve** button will change the status of the request and request for the EC2 instance creation. The `createInstance` method will be invoked with six parameters, as shown in the subsequent code.

The subsequent code invokes three local functions `createKeyPair()`, `getInstancePublicDndName()`, and `getInstanceState()`. Another two functions `updateRequest()` and `publishInstanceCreation()` will be invoked to change the status of the request and send a notification to the registered e-mail address and phone number. In a nutshell, the code does the following functions.

- Create a key pair and store it in Amazon S3.
- Submit the EC2 instance request.
- Track whether the instance got created.
- Email the connection details along with the key pair location by e-mail and SMS.
- Update the same details in the database, and in case the user missed the notification, he can get it from his dashboard.

The code is as follows:

```
public void createInstance(Integer requestId, String instanceType,
  String imageId, String kernelId, String keyName, String
    topicName) throws Exception {
  String keyPairName = createKeyPair(keyName);
  String keyPairLoc = "https://s3.amazonaws.com/my-
    keypair/"
    + keyPairName + ".pem";
  RunInstancesRequest request = new RunInstancesRequest();
  request.setInstanceType(InstanceType.T1Micro);
  request.setImageId(imageId);  request.setMinCount(1);
  request.setMaxCount(1);  request.setKernelId(kernelId);
  request.setKeyName(keyPairName);
  RunInstancesResult rs = client.runInstances(request);
  List<Instance> instances = rs.getReservation().
    getInstances();
  for (Instance instance : instances) {
    String awsInstanceId = instance.getInstanceId();
    String publicDNS = getInstancePublicDnsName
      (awsInstanceId);
    String State = getInstanceState(awsInstanceId);
    if (State.equalsIgnoreCase("running")) {
      String emailMsg = "Hi " + topicName
      + ",\nYour instance's public DNS is " + publicDNS
```

```
      + ".\nKey pair can be downloaded from " + keyPairLoc;
      RequestEntity e = new RequestEntity();
        e.setRequestId(requestId);
        e.setAwsInstanceId(awsInstanceId);
        e.setKeyName(keyPairName);   e.setDns(publicDNS);
      new RequestService().updateRequest(e);
      new SNSoperations().publishInstanceCreation(
        topicName, emailMsg);
} } }
```

The following code is used to produce a key pair with the name stated by the user. It will also add this key pair to the instance request. In order to store this file at an S3 location, the `saveKeyPair` method of the `S3Operations` class will be invoked with the `keyName` and `keypair` content as a byte array:

```
public String createKeyPair(String keyName) {
  keyName += System.currentTimeMillis();
  CreateKeyPairRequest request = new
    CreateKeyPairRequest();
  request.setKeyName(keyName);
  CreateKeyPairResult keyPair = client.
    createKeyPair(request);
  String key = keyPair.getKeyPair().getKeyMaterial();
  return new S3Operations().saveKeyPair(keyName,
  new ByteArrayInputStream(key.getBytes()));
}
```

The `saveKeyPair` method of the `S3Operations` class is shown as follows. In order to make this file available for public download, we set the ACL rule as shown here. The following code adds the key pair to the `my-keypair` S3 bucket:

```
public String saveKeyPair(String keyName, InputStream key)
  {
  AccessControlList acl = new AccessControlList();
  acl.grantPermission(GroupGrantee.AllUsers,
    Permission.Read);
  PutObjectRequest request = new PutObjectRequest("my-
    keypair", keyName + ".pem", key, null).
    withAccessControlList(acl);
  client.putObject(request);
  return keyName;
}
```

3. The following code iterates over all the instances in our account and returns the instance state of the passed `instanceId`. The following method will be invoked until the state becomes running:

```
public String getInstanceState(String instanceId) {
  DescribeInstancesResult dir = client.describeInstances();
  List<Reservation> reservations = dir.getReservations();
  for (Reservation reservation : reservations) {
    for (Instance instance : reservation.getInstances()) {
      if (instance.getInstanceId().equals(instanceId)) {
        InstanceState instanceState = instance.getState();
        return instanceState.getName();
      } } }
return null; }
```

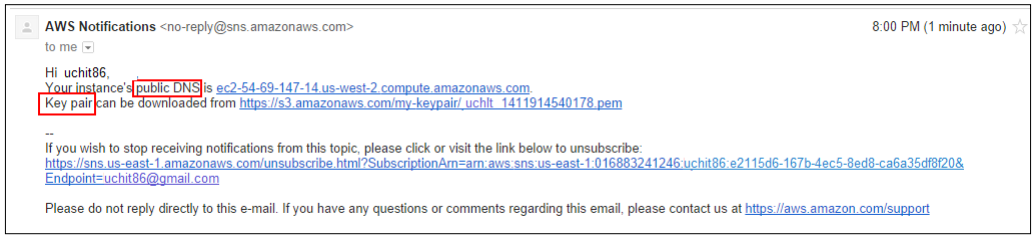
4. Once the instance state becomes running, the following method will be invoked. The following code is used to get the public DNS of the instance, without which the user cannot connect to the EC2 instance. The following code will describe all the EC2 instances for the account. Inside the `for-each` loop, we will check whether the passed `instanceId` is the same as that of the current instance. If that is the case, the public DNS will be returned.

```
public String getInstancePublicDnsName(String instanceId) {
  DescribeInstancesResult dir = client.describeInstances();
  List<Reservation> reservations = dir.getReservations();
  for (Reservation reservation : reservations) {
    for (Instance instance : reservation.getInstances()) {
      if (instance.getInstanceId().equals(instanceId)) {
        InstanceState instanceState = instance.getState();
        return instance.getPublicDnsName();
      } } }
return null;  }
```

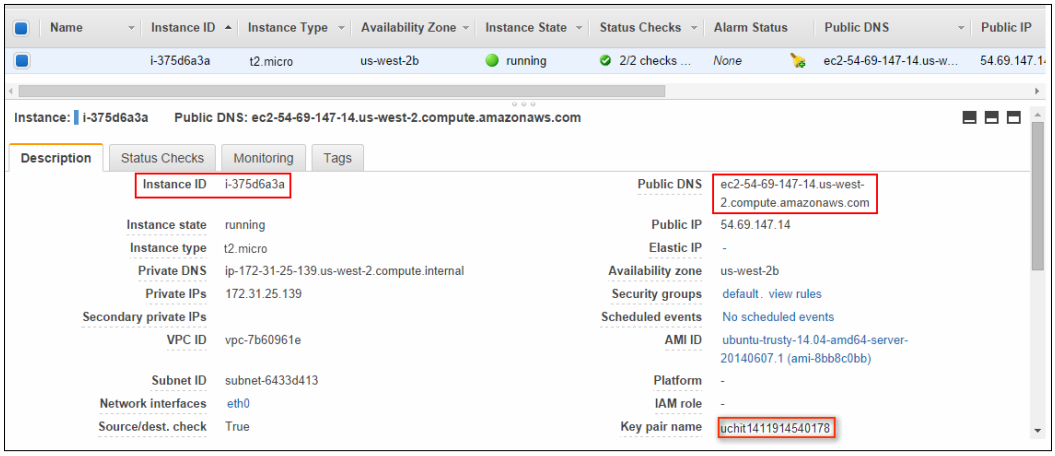
5. Once the key pair is created, stored, and the instance becomes available, the DNS and key pair location will be published as e-mail and SMS to the registered endpoints of the topic, as shown in the following code:

```
public void publishInstanceCreation(String topicName,
  String emailMsg) {
  PublishRequest publishRequest = new PublishRequest();
  publishRequest.setSubject("Education cloud- details");
  publishRequest.setMessage(emailMsg);
  publishRequest.setTopicArn("arn:aws:sns:us-east-
                1:016883241246:"+ topicName);
  publishRequest.setRequestCredentials(credential);
  client.publish(publishRequest);
}
```

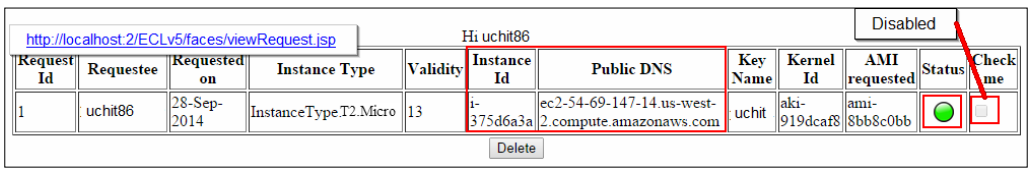
6. The mail for the user request is shown in the following screenshot. It will have information about public DNS and the S3 location, where key pairs corresponding to this instance is stored.



7. We can validate the identical details in the management console, as shown in the following screenshot. The three most important parameters (**Instance ID**, **Public DNS**, and **Key pair name**) are highlighted in the following screenshot:

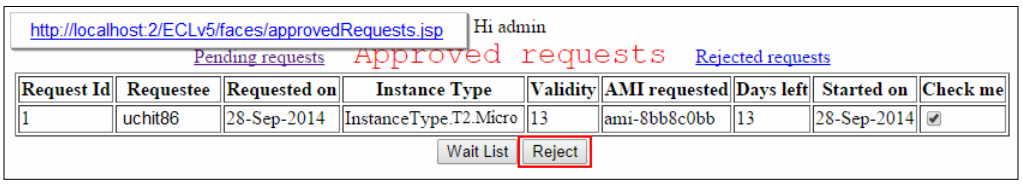


8. If the user logs into the app, the same information is shown in the dashboard.
9. Earlier, the status icon was in yellow; now, it has transformed to green (since the request is approved and the instance is provisioned). The checkbox to delete the request is disabled. This will be enabled only if the status is pending; otherwise (approved or rejected), it will be disabled.



Rejecting an instance entreaty

In the admin dashboard, if the admin has to reject the request and terminate the instance, we can check the corresponding checkbox and click on the **Reject** button. This will terminate the instance.

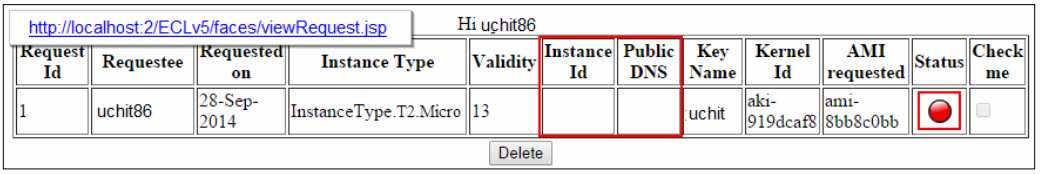


1. The code to terminate the instance request is shown as follows. It can accept a list of `instanceId`. Each `instanceId` will be iterated and the instances are terminated one by one. The implementation will stay with the `while` loop until it is terminated.

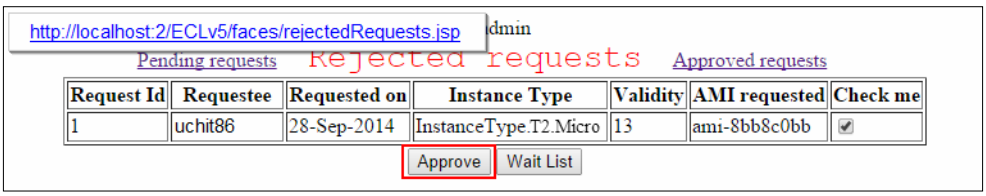
```
public void terminateInstance(List<String> instanceIds) {
  TerminateInstancesRequest tir = new
    TerminateInstancesRequest();
  tir.setRequestCredentials(credentials);
  tir.setInstanceIds(instanceIds);
  TerminateInstancesResult result = client.
    terminateInstances(tir);
  List<InstanceStateChange> resultList = result.
    getTerminatingInstances();
  for (InstanceStateChange instanceStateChange :
    resultList) {
  while (!getInstanceState(instanceStateChange.
    getInstanceId())
    .toString().equalsIgnoreCase("terminated"));
  }
}
```

2. If we access the management console, we can infer the same information about the requested instance. We can check the status as **shutting-down**:

3. The equivalent information will be made available in the user dashboard. Since the EC2 instance is terminated, instance ID and public DNS is also cleared:

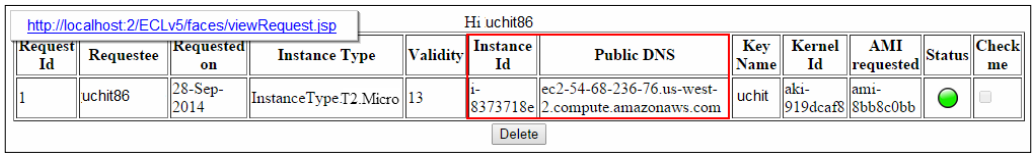


4. Now, this request will be available in the admin's rejected request page. We can check this request and click on **Approve** to provision the instance once more to the user.



5. Even though the instance (for the same request) is rejected and approved again, public DNS, key pair, and instance ID will be different. The following figure shows the mail:

6. In the user dashboard, instance ID, public DNS, and status will be updated.



Using RDS and Elastic Beanstalk

Since the app we created is the JSF app, we can see a file named `persistence.xml` in the `src/META-INF` folder. Copy the subsequent content to the XML and fill up `RDS-endpoint IP`, `port-number`, `RDS-instance-username`, and `RDS-instance-password` with the RDS instance details and save the file. Before using the app, make sure that a database with the name `education_cloud` is available in the RDS instance.

```
<persistence version="1.0"...>
<persistence-unit name="ECL">
  <class>education.cloud.entity.InstanceEntity</class>
  <class>education.cloud.entity.RequestEntity</class>
  <class>education.cloud.entity.LoginEntity</class>
  <class>education.cloud.entity.CustomerEntity</class>
<properties>
  <property name="toplink.jdbc.url" value="jdbc:mysql://<RDS-
    endpoint IP>:<port-number>/education_cloud" />
  <property name="toplink.jdbc.user" value="<RDS-instance-
    username>" />
  <property name="toplink.jdbc.driver"
    value="com.mysql.jdbc.Driver"/>
  <property name="toplink.jdbc.password" value="<RDS-instance-
    password>" />
<!-- Some more properties removed -->
</properties>
</persistence-unit>
</persistence>
```

Deploying this app on Elastic Beanstalk will be done in dozens of clicks if we use the Eclipse IDE. We can right-click on the JSF application and go to **Run as** | **Run on Server**, which opens a window asking us to choose the proper server. To run the app in our local system, we use Tomcat7. In order to deploy it in Elastic Beanstalk, we need to choose AWS Elastic Beanstalk for the Tomcat 7 server. A few more windows will pop up, in which we need not do anything (except click on the **Next** button). If the AWS plugin is properly configured, then this application will be deployed on AWS Elastic Beanstalk.

The application of superlative AWS exercises

There are a few security and performance apprehensions, which we might have come across in the EduCloud app. Let's review these one by one.

- We save the instance key pair in S3 and the file has read permission for everyone, so this is a major security breach. This should not be the case. We must store this somewhere secured; at least the read permission should be allowed only for the instance requester.
- Irrespective of whether the instance request is rejected or waitlisted, the EC2 instance gets terminated. So, we can make the application more efficient by stopping the instance (when waitlisting the request) and starting it again (when approving the request).
- There should be a provision for a non-admin user to start/stop an instance, which will reduce the billing amount.
- We cannot sell this app to a third party since it is against the AWS agreement.

Best practices with AWS

In this section, we will discuss complex application development and deployment problems, and how to develop AWS centric applications based on the underlying AWS infrastructure. We will go through some of the generic problem statements and try to solve them using AWS services as best possible solutions.

Problem 1: The scaling problem of the infrastructure for the in-house project or in the data center.

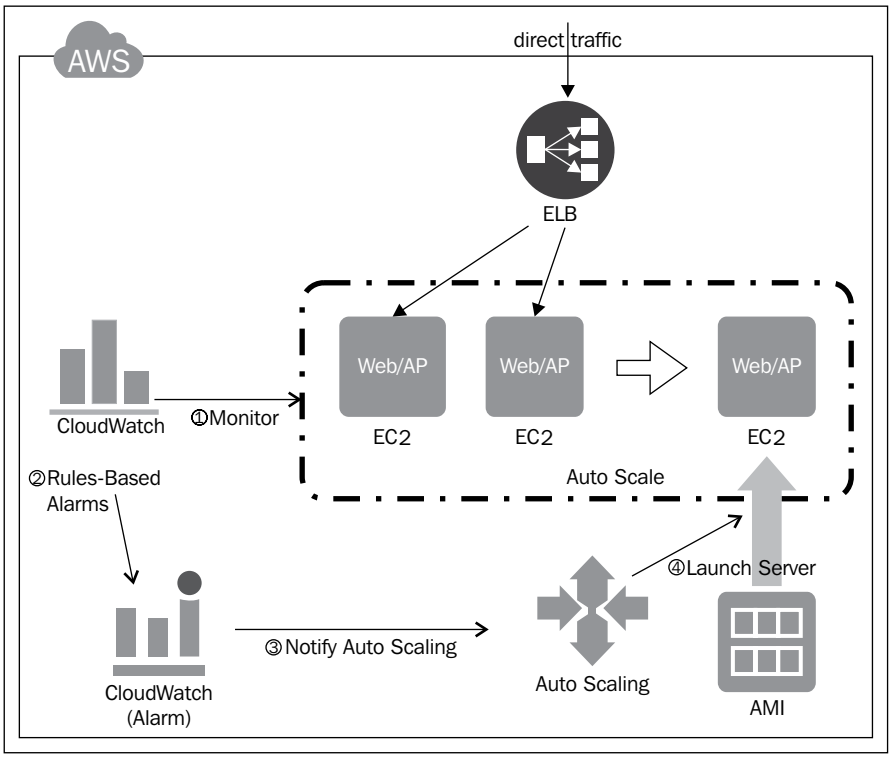
Solution: In this scenario, you have to scale up your infrastructure based on the load and demand. So, the following can be the best approach using Auto Scaling:

1. Set up EC2 instances under the ELB.
2. Create Base AMI for instances.
3. Define the policies (metrics) to trigger events such as add or remove instances.
4. Use CloudWatch to monitor these metrics.
5. Set up Auto Scaling, which will increase or decrease the number of instances based on policies.

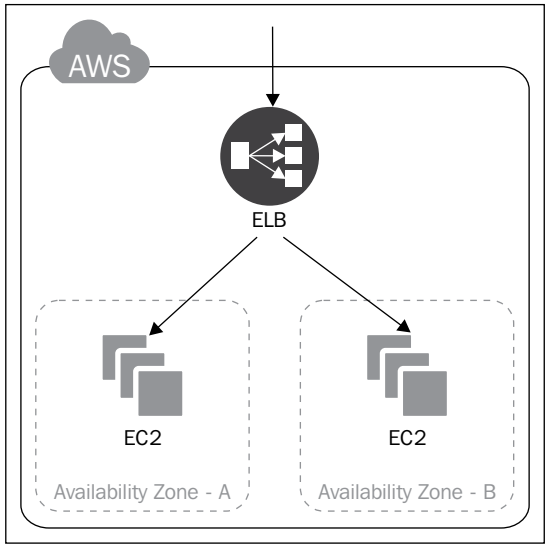
One should consider the following points as well:

- All of the data is removed in the instances.
- The data that the user wants to store or remain should be uploaded into S3 or inserted into DynamoDB.
- The application deployment method should be carefully considered because the following architecture depends on AMI.

You can refer to the following sample architecture as a design solution—how to implement these services together that we have seen throughout the book:



So, this can be the best suitable solution for scale out problem. Also, this can be a solution for high availability requirements. Similarly, for the multi datacenter pattern, you can create the following:

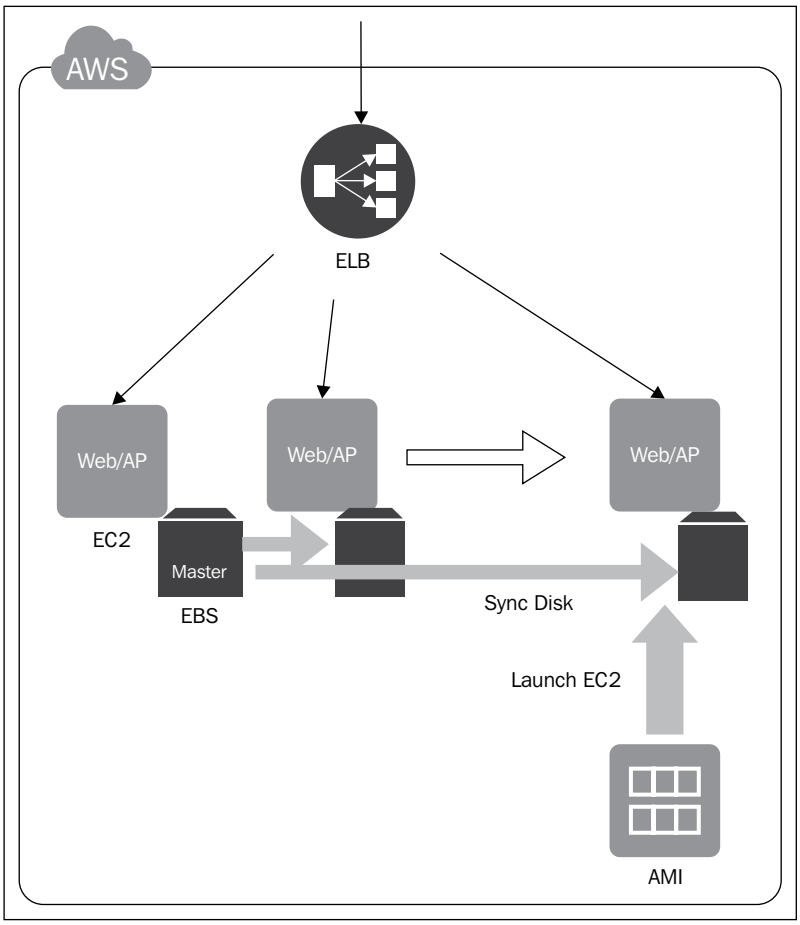


Problem 2: A scale-out construction is a communal method, but in systems that start less frequently, the construction is not one where it is likely to deliver multiserver services using numerous servers at all. In such a case, establishing procedures to handle the increased load may be time-consuming.

Solution: In this case, as a solution, you can create a combination of services, such as EC2, ELB, EBS, and AMI. The following are the steps to proceed further in such a case:

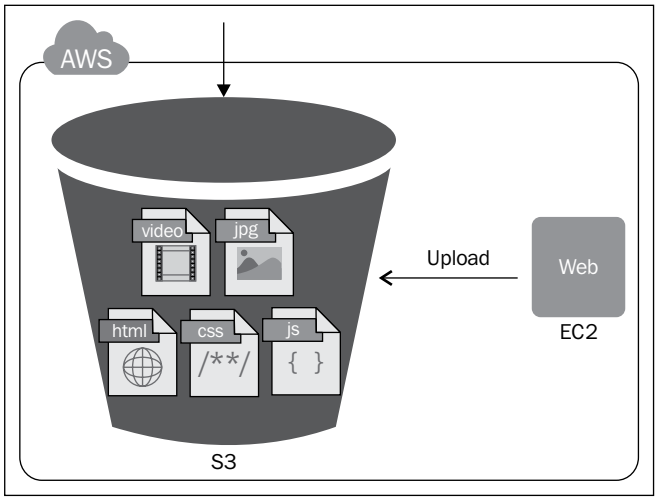
1. Set up your instances under ELB.
2. Create instances from running instances, or, let's say, clone them.
3. Use rsync or scheduled EBS volumes, which means rotationally, EBS volumes will attach and detach to instances and serve you the data.

The following diagram can be useful for you to understand this:

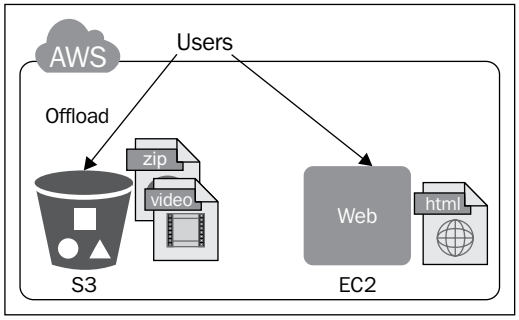


Problem 3: You will not be able to get benefit up by increasing the number of machines when there is an unexpected increase in the number of admissions in a short period of time. While you might deliberate handling this by increasing the number of servers by predicting the growth in admissions, growing the number of servers wastefully is difficult in terms of the price.

Solution: In this case, you can use Amazon S3 along with EC2 instances to store static content or even static websites. The following diagram can be helpful to understand more about the combination of AWS EC2 and AWS S3:



If you have a combination of static and dynamic content, you can follow this diagram:



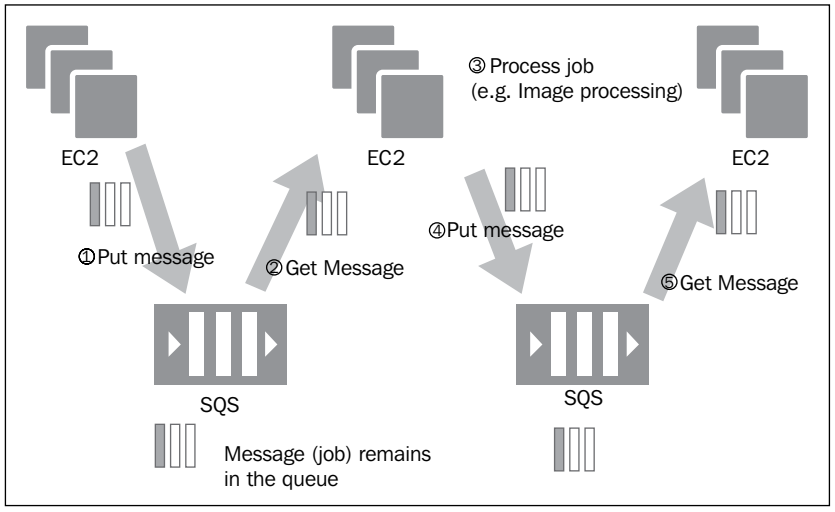
Problem 4: If you are carrying out successive processing, where processes running on numerous systems accompany each other, there will be a propensity for performance blockages resulting from having systems strongly linked to one another. This tight association also thwarts the retrieval operations when there is a catastrophe.

Solution: You can accomplish this loose coupling of systems among systems using queues and then exchanging messages that transfer jobs. You can use the Amazon SQS service to transfer a process from an EC2 instance for one process to the EC2 instance for the next process.

To start with this, you can create the SQS flow with EC2 as follows:

1. Receive a message.
2. Trigger a job.
3. Transmit the message.
4. Repeat steps 1 to 3.

Just go through the following diagram as per the preceding steps to understand the message flow between SQS and EC2:

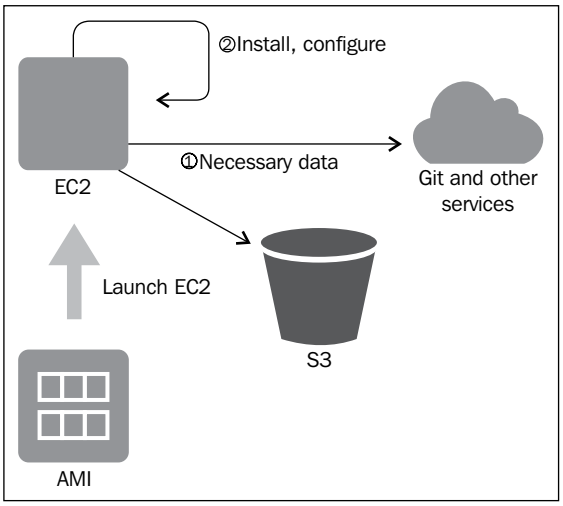


Problem 5: In the stamp pattern, you can get the machine image after all the setup has been accomplished, with the middleware and apps up and running. Whereas this lets you start up the VM tremendously fast, if you want to upgrade one of the middleware things, or you want to alter a locale in an app, you will have to rebuild the AMI.

Solution: While creating an AMI, you can specifically place the numerous parameter files required to initialize the EC2 instance in Amazon S3, and then, at the time of starting up an EC2 instance, have the EC2 instances read out the parameter files to shape themselves. You can store the parameter files in a repository such as Git or SVN. The following steps describe how to do this:

1. Place the required data for bootstrapping into S3 or some repository called Git or SVN.
2. Spin up an EC2 instance from a specific AMI, which includes the bootstrap.
3. At the time of startup, the instance will get the bootstrap data and configure itself accordingly.

Take a look at the following diagram to understand the flow and fundamentals of this:



So, based on the requirements, your architecture will be different with different AWS services. You can also check out the sample design patterns at `http://en.clouddesignpattern.org/index.php/Main_Page`, and the best possible solutions given by AWS for commonly required architecture are available at `http://aws.amazon.com/architecture`.

Summary

We started our journey by discussing the overview of the EduCloud app. Then, we saw the homepage and roles of the admin user and non-admin user. After this, we logged in as the admin and configured an instance so that it could be requested by the consumer. Then, the user signed up and requested for the instance. Then, we saw how a mail will be sent with connection particulars and key pair, when the request is approved by the administrator. In conclusion, we terminated the instance by rejecting the instance.

This chapter doesn't stop there; we also conferred about how to integrate RDS and AWS Elastic Beanstalk to this app. In the end, we deliberated four points, which will improve the app both recital wise and trade wise. The learning doesn't stop here. Since the app is downloadable, we can accomplish the earlier modifications and call ourselves AWS developers.

Finally, we saw generic problem statements of various real-time applications types and best solutions based on AWS services to fulfill the requirements.

Index

A

abstract syntax tree (AST) 324
Access Control List (ACL) 139
access keys
  adding, in Eclipse Toolkit 163, 164
  configuring, in Eclipse Toolkit 220
Amazon CloudFormation
  defining 117-121
  starting with 250
Amazon CloudFront
  about 85
  modifications 86
Amazon CloudFront Distribution
  creating 86-90
Amazon CloudSearch
  data, indexing for search 342-344
  data, uploading for search 342-344
  defining 337, 338
  search domain, configuring 338-341
  search domain, creating 338-341
  searching 344-346
Amazon CloudWatch
  defining 98-101
  EC2 instance, creating 101-103
  metrics, defining 98
Amazon EC2 key pairs
  observing, in Eclipse Toolkit 164-174
Amazon Elastic Block Storage (Amazon EBS) 10
Amazon Elastic MapReduce
  defining 310, 311
  Hadoop cluster, provisioning on 311-322
Amazon instances
  bootstrapping, Chef used 252-263
Amazon Kinesis
  Amazon Kinesis Client Library 331-335
  consumers 331
  data record 330
  defining 330
  partition keys 331
  producers 330
  shards 331
  streams 330
Amazon Machine Image (AMI) 8
Amazon Mechanical Turk
  defining 347-349
Amazon RDS
  about 91
  advantages 91
  managing, with CLI 91-94
Amazon Redshift
  cluster configurations 230, 231
  configuration options 228, 229
  defining 227, 228
Amazon SimpleDB 161
Amazon Simple Notification Service 161
Amazon Simple Queue Service 161
Amazon Simple Storage Service 161
Amazon SWF
  activity workers 266
  deciders 266
  workflow execution 266-275
  workflow starters 266
Amazon VPC
  architecting with 139-143
  components 139
  database instance, spinning in private subnet 148-150

instance, launching 143-147
OpenVPN client, downloading 154
OpenVPN instance, launching 151-153
OpenVPN server, configuring 155, 156
private subnet, creating 148
Remote Access Software VPN, creating 151
Amazon Web Services. *See* **AWS**
API
endpoints 197
eventual consistency model 197
libraries 197
API tools
defining 196
installing 197, 198
application
architecture 239
defining 359, 360
deploying, AWS Elastic Beanstalk used 217-227
migrating, to Cloud 238-240
application services
characteristics 6
app migration 240
architecture, Availability Zones (AZ)
advanced failover 10
intermediary failover 10
simple failover 10
as-create-launch-config command
arguments 43
authentication
about 31
factors 32
authorization
about 31, 32
services 31
automation, with Amazon SWF 265, 266
Auto Scaling
about 36
configuring 38
defining, AWS Management Console used 45-57
defining, CLI used 42-45
dynamic scaling 37
installing 38
performing, for AWS Elastic infrastructure 36
predictive scaling 37
prerequisites, installing 38-42
service, accessing 37
working with 37
Auto Scaling group
creating 44
Auto Scaling launch configuration
starting 43
Auto Scaling management
Auto Scaling API 38
AWS CLI 37
AWS Management Console 38
CLI 37
SDKs, for Auto Scaling 38
Availability Zones (AZ)
about 9
using 10
AWS
about 6, 97, 217, 350
accessing 162
Availability Zones (AZ), using 6, 9, 10
CDN service, defining from 293
defining 122
interfaces 157
regions 6-9
services 122
AWS architecture
URL 381
AWS Cloud
about 127
features 2
AWS CloudFormation
instances, bootstrapping with 250-252
AWS CloudTrail
benefits 232
features 232
interacting with 232
starting with 233-235
AWS EC2
about 11, 15
functionality 11
instance numbers and pricing 17
instance type 12
performance 15
pricing 12
working with 12

AWS EC2 endpoint 8
AWS EC2 instances
bootstrapping 244
AWS Elastic Beanstalk
starting 218, 237, 238
used, for application deployment 217-227
AWS Explorer 161
AWS global infrastructure 4-6
AWS keys
URL 347
AWS Management Console
used, for defining Auto Scaling 45-57
AWS OpsWorks
apps 275
database layer, creating 279
EC2 instances 275
instances, adding 280-286
layer 275
OpsWorks stack, creating 277
prerequisites 277
Rails App Server layer, creating 278
stack 275
working with 275, 276
AWS SDK for Java
URL 218
AWS SDKs
AWS Toolkit 161
documentation 159
lib folder 159
Samples folder 159
Third-party folder 160
working with 158-176
AWS secret keys and access ID
URL 40
AWS Secure Global Infrastructure
abuse activities, mitigating 354
Availability Zones (AZs) 353
defining 351, 352
keys, managing in Cloud 353
patches, managing 354
regions 353
service endpoints 353
Trusted Advisor tool 354
AWS security
URL 352
AWS services 2, 3, 17
AWS Simple Storage Service (S3)
about 79
object, storing in 80-85
AWS toolkit, for Eclipse
URL 61

B

batch processing flow
cluster, monitoring 116
defining 104
IAM role, creating 105, 106
results, viewing 115, 116
S3 bucket, creating 108
SQS tasks, creating 107
work, dispatching 115, 116
worker nodes, launching 109-115
best practices, with AWS 375-381
Big Data
defining 310
billing, AWS EC2
defining 17
URL 17
billing conventions
defining 289
Bitmap indexes 329
black belt booting
about 28
defining 244-249
bootstrapping 26, 27

C

case study
LAMP environment 122
CD
defining 264, 265
Chef
about 253
chef-client 255
chef-repo 255
cookbook 255
Knife 255
node 255
recipe 255
run_list 255

used, for bootstrapping Amazon instances 252-263
using 254
workstation 255
Chef servers
Enterprise Chef 255
Hosted Chef 255
Open Source Chef 255
URL 256
Chef Solo
URL 255
Chef Zero
URL 255
CI
defining 263
Gerrit 263
Jenkins 263
Nexus 264
Promotional workflow 263
Classless Inter-Domain Routing (CIDR) 141
CLI
Amazon RDS, managing with 91-94
defining 187-191
URL 188
used, for defining Auto Scaling 42-45
Cloud
applications, migrating to 238-240
leveraging 241
CloudFront
content, streaming 307
defining 293
using 297-306
working 293-295
CloudFront nomenclatures
distribution 294
edge locations 294
expiration time 294
objects 294
origin servers 294
Cloud Magic World
URL 36
CloudWatch metric
monitoring 98
cluster
monitoring 116
code libraries
working with 176
compiler 324
compliance, AWS
URL 352
components, of Cluster Configuration
defining 314
composite types
Arrays 325
Maps 325
Structs 325
Unions 325
computer services
characteristics 4
content delivery and networking services
characteristics 5
Content Delivery Network (CDN) 85, 293
Continuous delivery. *See* **CD**
Continuous integration. *See* **CI**
cost allocation reporting 290, 291
cost control architectures
about 292
access, controlling 292, 293
custom metric, in CloudWatch
monitoring 103

D

data
elementary approaches 138
replicating 137, 138
database layer
creating 279
databases
about 326
characteristics 5
Data Definition Language (DDL) 327
data format, for DynamoDB
defining 199
Data Manipulation Language (DML) 324
Data Migration 240
data model 326-328
data replication
asynchronous replication 138
considerations 137
synchronous replication 138
data storage scaling 60
data types 325

data types, DynamoDB
about 61-64
binary 200
binary set 200
number 200
number set 200
string 200
string set 200
deployment and management services
characteristics 6
design patterns
URL 381
development phase
AWS exercises, defining 375
best practices, with AWS 375-381
conventions 361
defining 360
defining, via Elastic Beanstalk 374
defining, via RDS 374
EduCloud, defining 362-365
handlers 361
instance, handling 365, 366
instance request, approving 367-371
instance request, rejecting 372, 373
devkit 158
digital rights management (DRM) 353
Directed Acyclic Graphs (DAG) 324
distribution
creating 298, 299
domain
removing, from Amazon CloudSearch 346
DR 128
driver 324
DR scenarios
backup and restore 129
defining 128
multisite solution 135
pilot light recovery, in AWS 130
warm standby solution 132-134
DR, with AWS
defining 128
RPO 128
RTO 128
DynamoDB
about 60, 170
data types 61-64
first SDK project, creating 65-69
Java SDK operations 70-76
DynamoDB local
about 76-78
URL, for downloading 76
DynamoDB Local
defining 185-187
URL 185

E

EBS 23
EBS-backed
versus instance store-backed 23
EBS (persistent storage)
attaching, to instance 24
EBS volume status
URL 355
EC2 API
example 198, 199
starting with 196
EC2 instances
Core 315
creating 101-103
Master 315
running 198
Task 316
Eclipse
URL 218
EduCloud
defining 362-365
Elastic Compute Cloud. *See* **AWS EC2**
elasticity 26, 27
Elastic Load Balancer (ELB) 35
Elastic MapReduce (EMR) 310
ELB
creating 53
ephemeral storage
about 18-23
versus persistent storage 18
ephemeral storage size
URL 22
example, Amazon CloudSearch
URL 346
execution engine 324
external tables 326

F

Federal Information Processing Standard (FIPS) 353

G

Gerrit 263
global infrastructure services
characteristics 4
Google File System (GFS) 310

H

Hadoop
defining 310
Hadoop Distributed File System (HDFS) 310
hardware security modules (HSMs) 353
Hive
about 310
buckets 326
clusters 326
partitions 326
table 326
HiveQL 322
Hive structural design
apparatuses, supporting 324
compiler 324
defining 323
execution engine 324
metastore 323
Hive tables
indexing on 328, 329
HTTP requests
defining 200-202
request body 203
request header 202, 203
response header 203, 204
HTTP standard components
HTTP method 194
request body 194
request headers 194
Universal Resource Identifier (URI) 194

I

IAM
about 11, 28
accessing 29-31
features 28
IAM role
creating 105-107, 235, 236
creating, for EC2 instance 99-101
IDE toolkit
working with 158-175
instance categories
compute-optimized instances 16
defining 16
general purpose instance 16
GPU instances 16
memory-optimized 16
storage-optimized 16
t1.micro instances 16
URL 16
use cases 16
instance numbers and pricing, AWS EC2
URL 17
instances
about 11
adding 280-286
bootstrapping 27
bootstrapping, with AWS CloudFormation 250-252
launching 244, 245
launching, in VPC 143-147
instances, booting
advantages 243
instances, bootstrapping
advantages 27, 28
instance storages 22
instance store-backed
versus EBS-backed 23
instance type
defining 16
on-demand instance 13
reserved instance 13, 14
selecting 12-15
spot instance 13, 15

Integrated Development Environment (IDE) 158
interfaces, with AWS
API tools 193
AWS Management Console 193
command-line interface 193
Eclipse plugin 193
Internet Service Providers (ISPs) 353

J

Java
URL 38
Java SDK operations
defining 180-184
Java SDK operations, DynamoDB 70-76
Java Server Faces (JSF) 359
Jenkins 263

K

Kinesis 330
Knife-ec2
about 260
installing 260

L

LAMP, on Amazon EC2
about 122
files permissions 124
LAMP server, installing 123
LAMP server, starting 123
LAMP web server, testing 124
prerequisites 122
launch configuration
determining 43

M

managed services bucket
application services 6
compute 4
content delivery and networking 5
databases 5
deployment and management 6
global infrastructure 4
security 4
storage 5
metastore 323
MFA device
about 32
defining 33
Hardware MFA 32
types 32
URL 33
Virtual MFA 32
migration
Cloud assessment 240
reasons for 239
MTurk
URL 350
MTurk sandbox account
URL 347
Multi-factor authentication device. *See* **MFA device**
multisite solution, phases
preparation phase 135
recovery phase 136

N

network access
authorizing 94-96
Network Address Translation (NAT) 139
Nexus 264
nodes 254
non-persistent storage 25

O

object
storing, in AWS Simple Storage Service (S3) 80-85
ObjectInspector 324
Object-Relational Mapping (ORM) 323
OpenVPN client
downloading 154
OpenVPN instance
launching 151-153

OpenVPN server
configuring 155, 156
URL 155
operations, in DynamoDB
BatchGetItem 212
BatchWriteItem 214
CreateTable 205
defining 204
DeleteItem 210
DeleteTable 212
DescribeTable 210
GetItem 207
ListTables 212
PutItem 205, 206
Query 208, 209
Scan 209
UpdateItem 206, 207
UpdateTable 211
OpsWorks stack
creating 277
options, scaling plans 50

P

paravirtual (PV) 25
persistent storage
about 25
using, with instance 24-26
versus ephemeral storage 18
pilot light method, phases
preparation phase 131
recovery phase 132
pip 40
Postman
using 201
pricing, AWS EC2
defining 17
private subnet
creating 148
database instance, spinning in 148-150
programmatic AWS billing
about 287
detailed billing report data, referencing 290
detailed billing reports, selecting 289
detailed billing reports, turning on 288
public key infrastructure (PKI) 353

Q

Query API 193
queue
creating, from AWS Management Console 234

R

Rails App Server layer
creating 278
RDP connections
enabling 148
RDS command-line toolkit
URL 92
recipes 253
Recovery point objective. *See* **RPO**
Recovery time objective. *See* **RTO**
Redundant Array of Independent Disks (RAID) 23
region, AWS
about 7-9
defining 3
URL 7
Relational Database Service (RDS) 138
Remote Access Software VPN
creating, to VPC 151
request body 203
request header
about 202
authorization 202
connection 202
Content-Length 202
Content-Type 202
host 202
x-amz-date 202
x-amz-target 202
requests
authenticating, REST APIs used 195, 196
response header
about 203
HTTP/1.1 203
x-amz-crc32 204
x-amzn-RequestId 203

REST APIs
API tools, defining 196
API tools, installing 197, 198
AWS access key ID 194
EC2 instance, running 198
signature 194
time stamp and date 194
used, for authenticating requests 194-196
REST-based APIs
defining 194
RPO 128
RTO 128

S

S3 bucket
creating 108
sample word count example
URL 313
scalability 26, 27
scalable application
characteristics 26
scaling types
horizontal scaling 35
vertical scaling 35
SDK, for Java
URL 161
SDK project
creating 177-179
SDKs and libraries, for PHP
URL 157
SDKs and libraries, for Python
URL 157
SDKs and libraries, for Ruby
URL 157
Secure Shell (SSH) 98
SerDe 324
server 254
services, AWS
abstracted services 350
container services 350
infrastructure services 350
setx command 40
Signature parameter
URL 203
Simple Notification Service (SNS) 52
Simple Storage Service (S3) 244
Simple Workflow (SWF) 265
SOAP API 193
Software Development Kit (SDK) 158
Solid State Disk (SSD) 60
SQS flow, with EC2
creating 380
SQS tasks
creating 107
storage system 59
streaming, content
defining 307
Live Streaming, for Amazon CloudFront 307
Live Streaming, with Wowza 307
on-demand Smooth Streaming 307

T

template, SugarCRM server
URL 118
TodoApp
URL 282
tools
using 360
working with 176
troubleshooting practices
about 355
DNS concerns 356
ephemeral disk corruption 355
High CPU 356
idle files, deleting 356
security group misconfiguration 356
Trusted Advisor tool 354

U

user data script, in EC2 instance launch
defining 246
User-defined Aggregate Functions (UDAFs) 324
User-defined Functions (UDFs) 324

V

virtual machine (VM) 252
VPC
about 42, 127
starting 140-143

W

warm standby solution, phases
preparation phase 133
recovery phase 134
web application
deploying 222
WordPress plugin
URL 90
worker nodes
launching 109-115
workstation 254

X

X.509 certificate 196

Thank you for buying

Mastering AWS Development

About Packt Publishing

Packt, pronounced 'packed', published its first book, *Mastering phpMyAdmin for Effective MySQL Management*, in April 2004, and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution-based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern yet unique publishing company that focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website at `www.packtpub.com`.

About Packt Enterprise

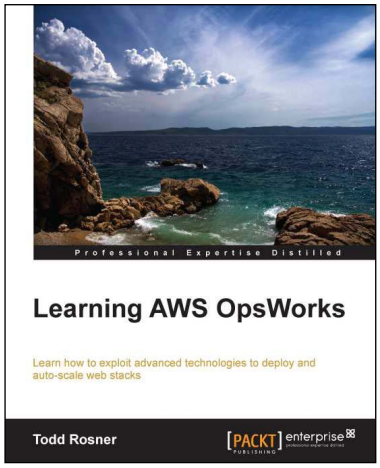
In 2010, Packt launched two new brands, Packt Enterprise and Packt Open Source, in order to continue its focus on specialization. This book is part of the Packt Enterprise brand, home to books published on enterprise software – software created by major vendors, including (but not limited to) IBM, Microsoft, and Oracle, often for use in other corporations. Its titles will offer information relevant to a range of users of this software, including administrators, developers, architects, and end users.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to `author@packtpub.com`. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, then please contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

[PACKT] enterprise
professional expertise distilled
PUBLISHING

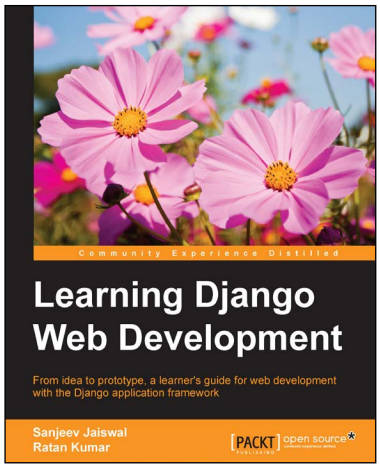


Learning AWS OpsWorks

ISBN: 978-1-78217-110-2 Paperback: 126 pages

Learn how to exploit advanced technologies to deploy and auto-scale web stacks

1. Discover how a DevOps cloud management solution can accelerate your path to delivering highly scalable infrastructure and applications.
2. Learn about infrastructure automation, auto-scaling, and distributed architecture using a Chef-based framework.
3. Includes illustrations, details, and practical examples for successful scaling in the cloud.



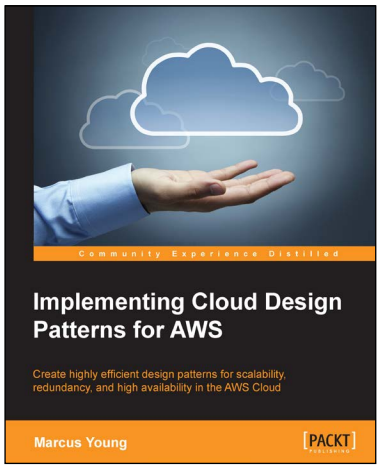
Learning Django Web Development

ISBN: 978-1-78398-440-4 Paperback: 336 pages

From idea to prototype, a learner's guide for web development with the Django application framework

1. Build two real-life based projects, one based on SQL and other based on NoSQL.
2. Best practices to code, debug, and deploy the Django web application.
3. Easy to follow instructions and real world examples to build highly effective Django web application.

[PACKT] enterprise
professional expertise distilled
PUBLISHING

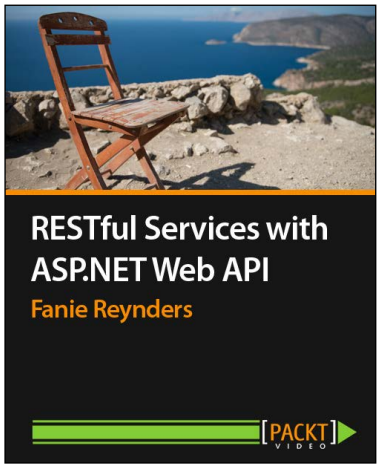


Implementing Cloud Design Patterns for AWS

ISBN: 978-1-78217-734-0 Paperback: 228 pages

Create highly efficient design patterns for scalability, redundancy, and high availability in the AWS Cloud

1. Create highly robust systems using cloud infrastructure.
2. Make web applications resilient against scheduled and accidental down-time.
3. Explore and apply Amazon-provided services in unique ways to solve common problems.



RESTful Services with ASP.NET Web API

ISBN: 978-1-78328-575-4 Duration: 02:04 hours

Get hands-on experience of building RESTful services for the modern Web using ASP.NET Web API

1. Apply your current ASP.NET knowledge to make your Web APIs more secure and comply to the global standard in order to make your service RESTful.
2. Explore the possibilities of extending your Web APIs by making use of message handlers, filters, and media formatters.
3. Comprehensive examples to help you build an end-to-end working solution for a real-use case.

Please check **www.PacktPub.com** for information on our titles