

Yohan Wadia

AWS Administration - The Definitive Guide

Second Edition

Design, build, and manage your infrastructure on
Amazon Web Services



Packt>

AWS Administration – The Definitive Guide *Second Edition*

Design, build, and manage your infrastructure on Amazon
Web Services

Yohan Wadia



BIRMINGHAM - MUMBAI

AWS Administration – The Definitive Guide

Second Edition

Copyright © 2018 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 or its dealers and distributors, will be held liable for any damages caused or alleged to have been caused directly or indirectly by this book.

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

Commissioning Editor: Vijin Boricha
Acquisition Editor: Heramb Bhavsar
Content Development Editor: Sharon Raj
Technical Editor: Vishal Kamal Mewada
Copy Editor: Safis Editing
Project Coordinator: Virginia Dias
Proofreader: Safis Editing
Indexer: Aishwarya Gangawane
Graphics: Tom Scaria
Production Coordinator: Nilesh Mohite

First published: February 2016

Second edition: March 2018

Production reference: 1220318

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

ISBN 978-1-78847-879-3

www.packtpub.com



`mapt.io`

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

Why subscribe?

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

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.

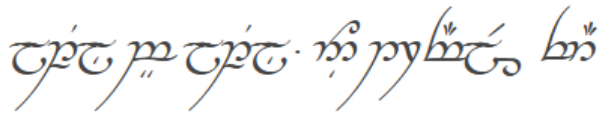
Contributors

About the author

Yohan Wadia is a client-focused evangelist and technologist with more than 8 years of experience in the cloud industry, focused on helping customers succeed with cloud adoption.

As a technical consultant, he provides guidance and implementation services to customers looking to leverage cloud computing through either Amazon Web Services, Windows Azure, or Google Cloud Platform by helping them come up with pragmatic solutions that make practical as well as business sense.

I wish to dedicate this book to my family: mom, dad, sister, and Fred! Thank you for all your love, support, and encouragement. Also a big shout out to my fellow mates who have helped me along the way, in many ways! Mitesh, Murali, Mahesh and Sam. Thank you! Last but not the least, a special thanks to a dear friend and family, Rohi. Happy Birthday!



Little by little, one travels far

- J. R. R. Tolkien

About the reviewer

Naveenkumar Vijayakumar (Naveen Vijay) is currently a cloud and automation architect at Digital Intelligence Systems, LLC (DISYS) and focuses on presales, solutions engineering, architecture, and delivery.

Naveen's portfolio includes experience in Azure, AWS, Serverless, containers, Robotic Process Automation, and chatbots (Alexa).

He earned his master's degree in IT from International Institute of Information Technology - Bangalore (IIIT-B) and currently lives in Dallas, Texas. He can be found on Twitter at [@navcode](#).

Packt is searching for authors like you

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

Table of Contents

Preface	1
Chapter 1: What's New in AWS?	7
Improvements in existing services	8
Elastic Compute Cloud	8
Availability of FPGAs and GPUs	9
Simple Storage Service	9
Virtual Private Cloud	11
CloudWatch	12
Elastic Load Balancer	13
Introduction of newer services	14
Plan of attack!	17
Summary	18
Chapter 2: Managing EC2 with Systems Manager	19
Introducing EC2 Systems Manager	20
Getting started with the SSM agent	20
Configuring IAM Roles and policies for SSM	20
Installing the SSM agent	27
Configuring the SSM agent to stream logs to CloudWatch	29
Introducing Run Command	31
Working with State Manager	35
Simplifying instance maintenance using System Manager Automation	39
Working with automation documents	39
Patching instances using automation	41
Triggering automation using CloudWatch schedules and events	43
Managing instance patches using patch baseline and compliance	46
Getting started with Inventory Management	50
Planning your next steps	51
Summary	52
Chapter 3: Introducing Elastic Beanstalk and Elastic File System	53

Introducing Amazon Elastic Beanstalk	54
Concepts and terminologies	55
Getting started with Elastic Beanstalk	58
Creating the Dev environment	58
Working with the Elastic Beanstalk CLI	64
Understanding the environment dashboard	67
Cloning environments	70
Configuring the production environment	71
Introducing Amazon Elastic File System	75
How does it work?	75
Creating an Elastic File System	77
Extending EFS to Elastic Beanstalk	79
Planning your next steps	81
Summary	83
Chapter 4: Securing Workloads Using AWS WAF	84
Introducing AWS Web Application Firewall	85
Concepts and terminologies	86
Getting started with WAF	89
Creating the web ACL	90
Creating the conditions	92
Creating rules	94
Assigning a WAF Web ACL to CloudFront distributions	99
Working with SQL injection and cross-site scripting conditions	100
Automating WAF Web ACL deployments using CloudFormation	105
Monitoring WAF using CloudWatch	111
Planning your next steps	113
Introduction to AWS Shield	113
Summary	115
Chapter 5: Governing Your Environments Using AWS CloudTrail and AWS Config	116
Introducing AWS CloudTrail	117
Working with AWS CloudTrail	119
Creating your first CloudTrail Trail	119
Viewing and filtering captured CloudTrail Logs and Events	122
Modifying a CloudTrail Trail using the AWS CLI	124

Monitoring CloudTrail Logs using CloudWatch	126
Creating custom metric filters and alarms for monitoring CloudTrail Logs	130
Automating deployment of CloudWatch alarms for AWS CloudTrail	132
Analyzing CloudTrail Logs using Amazon Elasticsearch	135
Introducing AWS Config	139
Concepts and terminologies	140
Getting started with AWS Config	141
Creating custom config rules	145
Tips and best practices	148
Summary	149
Chapter 6: Access Control Using AWS IAM and AWS Organizations	150
What's new with AWS IAM	151
Using the visual editor to create IAM policies	152
Testing IAM policies using the IAM Policy Simulator	156
Introducing AWS Organizations	159
Getting started with AWS Organizations	161
Planning your next steps	169
Summary	170
Chapter 7: Transforming Application Development Using the AWS Code Suite	171
Understanding the AWS Code Suite	172
Getting Started with AWS CodeCommit	175
Working with branches, commits, and triggers	180
Introducing AWS CodeDeploy	184
Concepts and terminologies	185
Installing and configuring the CodeDeploy agent	187
Setting up the AppSpec file	190
Creating a CodeDeploy application and deployment group	193
Introducing AWS CodePipeline	195
Creating your own continuous delivery pipeline	196
Putting it all together	198
Planning your next steps	202
Summary	203

Chapter 8: Messaging in the Cloud Using Amazon SNS and Amazon SQS	204
Understanding the AWS messaging services	205
Getting started with Amazon Simple Notification Service	208
Sending text messages using SNS	211
Using Amazon SNS as triggers	215
Monitoring Amazon SNS using Amazon CloudWatch metrics	219
Introducing Amazon Simple Queue Service	221
Creating your first queue	222
Creating a FIFO queue using the AWS CLI	227
Integrating Amazon SNS and Amazon SQS	229
Planning your next steps	232
Summary	234
Chapter 9: Powering Analytics Using Amazon EMR and Amazon Redshift	235
Understanding the AWS analytics suite of services	236
Introducing Amazon EMR	238
Concepts and terminologies	239
Getting started with Amazon EMR	241
Connecting to your EMR cluster	249
Running a job on the cluster	251
Monitoring EMR clusters	254
Introducing Amazon Redshift	256
Getting started with Amazon Redshift	257
Connecting to your Redshift cluster	261
Working with Redshift databases and tables	263
Planning your next steps	267
Summary	268
Chapter 10: Orchestrating Data using AWS Data Pipeline	269
Introducing AWS Data Pipeline	270
Getting started with AWS Data Pipeline	273
Working with data pipeline definition Files	280
Executing remote commands using AWS Data Pipeline	284
Backing up data using AWS Data Pipeline	288
Planning your next steps	296

Summary	297
Chapter 11: Connecting the World with AWS IoT and AWS Greengrass	298
IoT – what is it?	299
Introducing the AWS IoT suite of services	300
Getting started with AWS IoT Core	301
Connecting a device to AWS IoT Core	303
Getting started with AWS IoT Device SDK	308
Working with IoT rules	314
Introducing AWS Greengrass	318
Connecting a device to Greengrass Core	320
Running Lambda functions on AWS Greengrass	324
Monitoring AWS IoT devices and services	327
Summary	328
Other Books You May Enjoy	330
Leave a review - let other readers know what you think	332
Index	333

Preface

Amazon Web Services has been the go-to cloud for customers and enterprises for a long time now. The cloud provider has evolved from just an Infrastructure as a Service provider to everything and anything as a service that helps in the development of applications, game development, IoT, big data analysis, customer engagement services, AR-VR, and much more! However, with so many services and products coming up each year, it tends to get difficult for beginners to know where and how exactly to start using these services.

This book is a one-stop shop where you can find all there is to getting started with AWS services, which includes EC2 Systems Manager, Elastic Beanstalk, EFS, CloudTrail, EMR, IoT, and a whole lot more! If you are a sysadmin or an architect or someone who just wants to learn and explore various aspects of administering AWS services, then this book is the right choice for you! Each chapter of this book is designed to help you understand the individual services' concepts and gain hands-on experience by practicing simple and easy-to-follow steps. The book also highlights some key best practices and recommendations that you ought to keep in mind when working with AWS.

Who this book is for

This book is intended for any and all IT professionals who wish to learn and implement AWS for their own environment and application hosting. Although no prior experience or knowledge is required, it will be beneficial for you to have basic Linux knowledge and some understanding of networking concepts and server virtualization.

What this book covers

Chapter 1, *What's New in AWS?*, contains a brief introduction to some of the key enhancements and announcements made to the existing line of AWS services and products.

Chapter 2, *Managing EC2 with Systems Manager*, provides a brief introduction to using EC2 Systems Manager to manage your fleet of EC2 instances. It also covers an in-depth look at how to work with SSM agents, Run Command, as well as other systems manager features, such as automation, patching, and inventory management.

Chapter 3, *Introducing Elastic Beanstalk and Elastic File System*, explains how to leverage both Elastic Beanstalk and the Elastic File Systems services to build and scale out web applications and deploy them with absolute ease.

Chapter 4, *Securing Workloads Using AWS WAF*, discusses some of the key aspects that you can leverage to provide added security for your web applications using AWS WAF and AWS Shield. The chapter also provides some keen insights into how you can protect your web applications against commonly occurring attacks such as cross-site scripting and SQL injections.

Chapter 5, *Governing Your Environments Using AWS CloudTrail and AWS Config*, introduces you to the concept and benefits provided by leveraging AWS CloudTrail and AWS Config. The chapter covers in-depth scenarios using which you can standardize governance and security for your AWS environments.

Chapter 6, *Access Control Using AWS IAM and AWS Organizations*, takes a look at some of the latest enhancements made to the AWS IAM service. It also walks you through how you can manage your AWS accounts with better efficiency and control using AWS organizations as a Service.

Chapter 7, *Transforming Application Development Using the AWS Code Suite*, covers an in-depth look at how you can leverage CodeCommit, CodeDeploy, and CodePipeline to design and build complete CICD pipelines for your applications.

Chapter 8, *Messaging in the Cloud Using Amazon SNS and Amazon SQS*, provides an in-depth look at how you can effectively develop modern cloud-ready, decoupled applications, and perform general housekeeping of your AWS accounts.

Chapter 9, *Powering Analytics Using Amazon EMR and Amazon Redshift*, provides practical knowledge and hands-on approach to process and a run large-scale analytics and data warehousing in the AWS Cloud.

Chapter 10, *Orchestrating Data Using AWS Data Pipeline*, covers how you can effectively orchestrate the movement of data from one AWS service to another using simple, reusable pipeline definitions.

Chapter 11, *Connecting the World with AWS IoT and AWS Greengrass*, provides a quick introduction to the AWS IoT Suite of services, along with hands-on guides on how you can connect, test, and monitor IoT devices with utmost ease.

To get the most out of this book

To start using this book, you will need the following software installed on your local desktop:

- An SSH client such as PuTTY, a key generator such as PuTTYgen, and a file transferring tool such as WinSCP
- Any modern web browser, preferably Mozilla Firefox

Download the example code files

You can download the example code files for this book from your account at www.packtpub.com. If you purchased this book elsewhere, you can visit www.packtpub.com/support and register to have the files emailed directly to you.

You can download the code files by following these steps:

1. Log in or register at www.packtpub.com.
2. Select the **SUPPORT** tab.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the book in the **Search** box and follow the onscreen instructions.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR/7-Zip for Windows
- Zipeg/iZip/UnRarX for Mac
- 7-Zip/PeaZip for Linux

The code bundle for the book is also hosted on GitHub at <https://github.com/PacktPublishing/AWS-Administration-The-Definitive-Guide-Second-Edition>. In case there's an update to the code, it will be updated on the existing GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at <https://github.com/PacktPublishing/>. Check them out!

Conventions used

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

`CodeInText`: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "The document comprises of two primary sections: a `Parameters` section, which contains a list of actions to be performed by the document, followed by a `mainSteps` section that specifies the action, which in this case is the `aws:configurePackage` to be performed by the document. In this case, the document when invoked will ask the user to select either `apache2`, `mysql-server`, or `php` from the dropdown list followed by an optional version number of the software you select."

A block of code is set as follows:

```
{
  "Effect": "Allow",
  "Action": [
    "ec2messages:AcknowledgeMessage",
    "ec2messages:DeleteMessage",
    "ec2messages:FailMessage",
    "ec2messages:GetEndpoint",
    "ec2messages:GetMessages",
    "ec2messages:SendReply"
  ],
  "Resource": "*"
},
```

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

```
{
  "Effect": "Allow",
  "Action": [
    "ec2messages:AcknowledgeMessage",
    "ec2messages:DeleteMessage",
    "ec2messages:FailMessage",
    "ec2messages:GetEndpoint",
    "ec2messages:GetMessages",
    "ec2messages:SendReply"
  ],
  "Resource": "*"
},
```

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

```
# wget
https://s3.amazonaws.com/ec2-downloads-windows/SSMAgent/latest/debian_amd64/
amazon-ssm-agent.deb
```

Bold: Indicates a new term, an important word, or words that you see onscreen. For example, words in menus or dialog boxes appear in the text like this. Here is an example: "In the **Create Role** wizard, select the **EC2** option from the **AWS service** role type, as shown in the following screenshot. Next, select the **EC2** option as the *use case* for this activity and click on **Next: Permissions** button to continue."



Warnings or important notes appear like this.



Tips and tricks appear like this.

Get in touch

Feedback from our readers is always welcome.

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

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

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

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

Reviews

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

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

1

What's New in AWS?

Having spent many years in the IT industry, you get to see a lot of new technologies, products, and platforms that start to evolve, gradually mature, and eventually be replaced by something that's faster and better! I guess in some ways, this concept applies to this book as well. When I first started out writing the first edition of this series in 2016, I felt that this would be the pinnacle of technology and nothing could be created to replace it! And now, here I am doing precisely the opposite! Writing this second edition is a just small testament that everything evolves with time! Just look at AWS, and you will see how much the platform has changed and grown, especially in the last couple of years!

I still remember the time when I first started exploring AWS way back in 2009, when it was the early days for the likes of EC2 and CloudFront, still adding new features to them, SimpleDB and VPC just starting to take shape, and so on; the thing that really amazes me is how far the platform has come today! With more than 50 different solutions and service offerings ranging from big data analytics, to serverless computing, to data warehousing and ETL solutions, digital workspaces and code development services, AWS has got it all! Which is one of the reasons why I have always been a huge fan of it! It's not only about revenue and the number of customers, but how well do you adapt and evolve to changing times and demands.

So here we are, back at it again! A new book with a lot of new things to learn and explore! But before we begin with the deep dives into some really interesting and powerful services, let's take this time to traverse a little way back in time and understand what has been happening in AWS over this past year, and how the services that we explored in the first edition are shaping up today!

In this chapter, we will be covering the following topics:

- Improvements in existing AWS services.
- A brief introduction to newer AWS services and what they are used for.
- Plan of attack! How we will progress through the book.

Improvements in existing services

There have been quite a few improvements in the services that were covered back in the first edition of *AWS Administration - The Definitive Guide*. In this section, we will highlight a few of these essential improvements and understand their uses. To start off, let's look at some of the key enhancements made in EC2 over the past year or two.

Elastic Compute Cloud

Elastic Compute Cloud (EC2) is by far one of the oldest running services in AWS, and yet it still continues to evolve and add new features as the years progress. Some of the notable feature improvements and additions are mentioned here:

- **Introduction of the t2.xlarge and t2.2xlarge instances:** The **t2** workloads are a special type of workload, as they offer a low-cost burstable compute that is ideal for running general purpose applications that don't require the use of CPU all the time, such as web servers, application servers, LOB applications, development, to name a few. The *t2.xlarge* and *t2.2xlarge* instance types provide 16 GB of memory and 4 vCPU, and 32 GB of memory and 8 vCPU respectively.
- **Introduction of the I3 instance family:** Although EC2 provides a comprehensive set of instance families, there was a growing demand for a specialized storage-optimized instance family that was ideal for running workloads such as relational or NoSQL databases, analytical workloads, data warehousing, Elasticsearch applications, and so on. Enter I3 instances! I3 instances are run using non-volatile memory express (NVMe) based SSDs that are suited to provide extremely optimized high I/O operations. The maximum resource capacity provided is up to 64 vCPUs with 488 GB of memory, and 15.2 TB of locally attached SSD storage.



This is not an exhaustive list in any way. If you would like to know more about the changes brought about in AWS, check this out, at <https://aws.amazon.com/about-aws/whats-new/2016/>.

Availability of FPGAs and GPUs

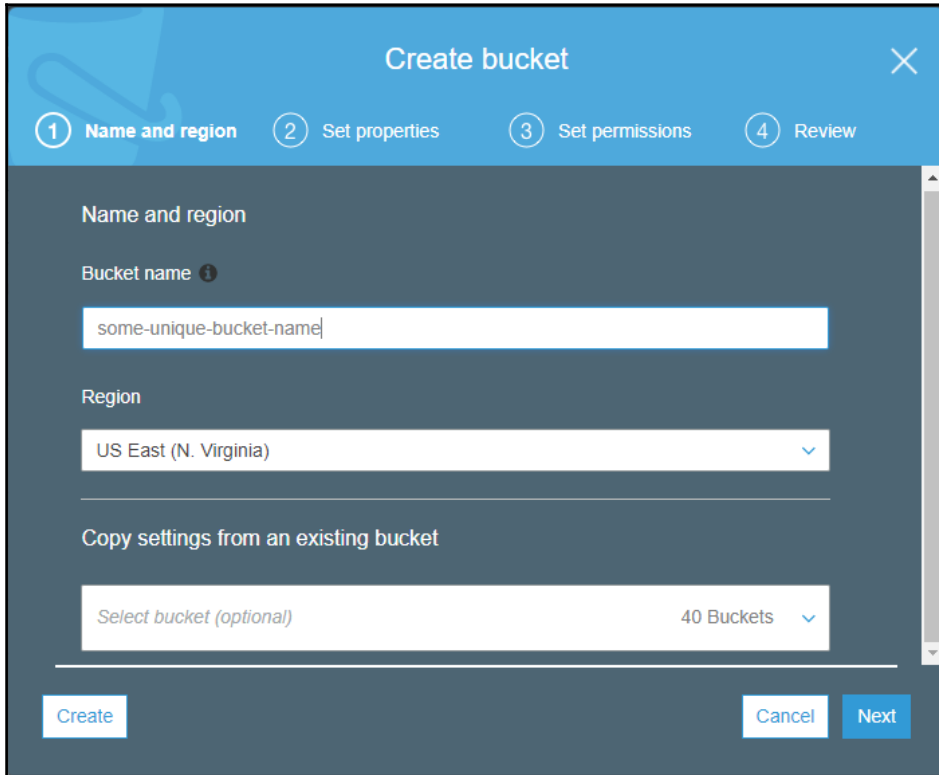
One of the key use cases for customers adopting the public cloud has been the availability of high-end processing units that are required to run HPC applications. One such new instance type added last year was the F1 instance, which comes equipped with field programmable gate arrays (FPGAs) that you can program to create custom hardware accelerations for your applications. Another awesome feature to be added to the EC2 instance family was the introduction of the Elastic GPUs concept. This allows you to easily provide graphics acceleration support to your applications at significantly lower costs but with greater performance levels. Elastic GPUs are ideal if you need a small amount of GPU for graphics acceleration, or have applications that could benefit from some GPU, but also require high amounts of compute, memory, or storage.

Simple Storage Service

Similar to EC2, **Simple Storage Service (S3)** has had its own share of new features and support added to it. Some of these are explained here:

- **S3 Object Tagging:** S3 Object Tagging is like any other tagging mechanism provided by AWS, used commonly for managing and controlling access to your S3 resources. The tags are simple key-value pairs that you can use for creating and associating IAM policies for your S3 resources, to set up S3 life cycle policies, and to manage transitions of objects between various storage classes.
- **S3 Inventory:** S3 Inventory was a special feature provided with the sole purpose of cataloging the various objects and providing that as a useable CSV file for further analysis and inventorying. Using S3 Inventory, you can now extract a list of all objects present in your bucket, along with its metadata, on a daily or weekly basis.
- **S3 Analytics:** A lot of work and effort has been put into S3 so that it is not only used just as another infinitely scalable storage. S3 Analytics provides end users with a medium for analyzing storage access patterns and defines the right set of storage class based on these analytical results. You can enable this feature by simply setting a storage class analysis policy, either on an object, prefix, or the entire bucket as well. Once enabled, the policy monitors the storage access patterns and provides daily visualizations of your storage usage in the AWS Management Console. You can even export these results to an S3 bucket for analyzing them using other business intelligence tools of your choice, such as Amazon QuickSight.

- **S3 CloudWatch metrics:** It has been a long time coming, but it is finally here! You can now leverage 13 new CloudWatch metrics specifically designed to work with your S3 buckets objects. You can receive one minute CloudWatch metrics, set CloudWatch alarms, and access CloudWatch dashboards to view real-time operations and the performance of your S3 resources, such as total bytes downloaded, number of 4xx HTTP response counts, and so on.
- **Brand new dashboard:** Although the dashboards and structures of the AWS Management Console change from time to time, it is the new S3 dashboard that I'm really fond of. The object tagging and the storage analysis policy features are all now provided using the new S3 dashboard, along with other impressive and long-awaited features, such as searching for buckets using keywords and the ability to copy bucket properties from an existing bucket while creating new buckets, as depicted in the following screenshot:



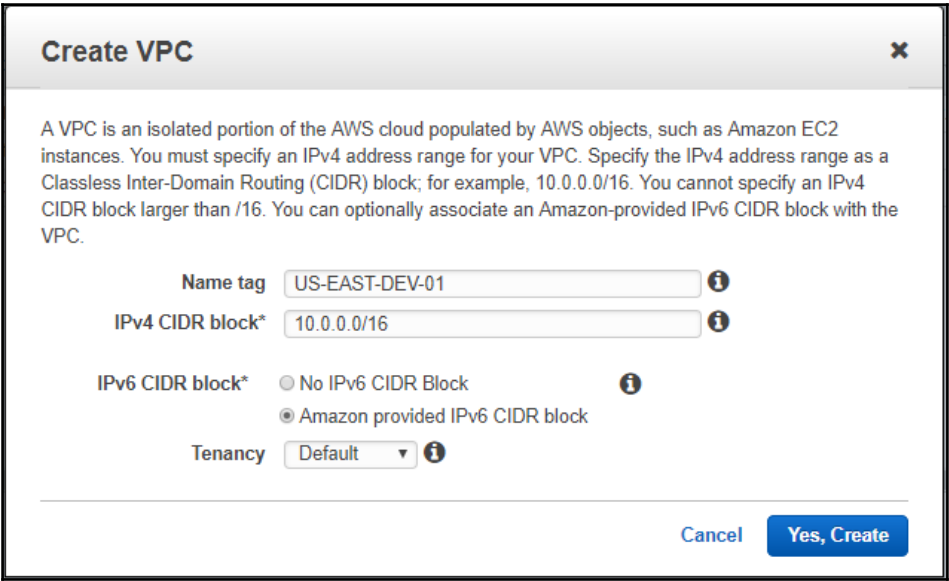
The screenshot shows the 'Create bucket' wizard in the AWS Management Console. The wizard has four steps: 1. Name and region, 2. Set properties, 3. Set permissions, and 4. Review. The first step is active. It contains a 'Bucket name' field with the text 'some-unique-bucket-name', a 'Region' dropdown menu set to 'US East (N. Virginia)', and a 'Copy settings from an existing bucket' section with a dropdown menu showing 'Select bucket (optional)' and a count of '40 Buckets'. At the bottom, there are three buttons: 'Create', 'Cancel', and 'Next'.

- **Amazon S3 transfer acceleration:** This feature allows you to move large workloads across geographies into S3 at really fast speeds. It leverages Amazon CloudFront endpoints in conjunction with S3 to enable up to 300 times faster data uploads without having to worry about any firewall rules or upfront fees to pay.

Virtual Private Cloud

Similar to other services, **Virtual Private Cloud (VPC)** has seen quite a few functionalities added to it over the past years; a few important ones are highlighted here:

- **Support for IPv6:** With the exponential growth of the IT industry as well as the internet, it was only a matter of time before VPC too started support for IPv6. Today, IPv6 is extended and available across all AWS regions. It even works with services such as EC2 and S3. Enabling IPv6 for your applications and instances is an extremely easy process. All you need to do is enable the **IPv6 CIDR block** option, as depicted in the VPC creation wizard:



The screenshot shows the 'Create VPC' wizard in the AWS Management Console. The title bar says 'Create VPC' with a close button. Below the title bar, there is a descriptive text: 'A VPC is an isolated portion of the AWS cloud populated by AWS objects, such as Amazon EC2 instances. You must specify an IPv4 address range for your VPC. Specify the IPv4 address range as a Classless Inter-Domain Routing (CIDR) block; for example, 10.0.0.0/16. You cannot specify an IPv4 CIDR block larger than /16. You can optionally associate an Amazon-provided IPv6 CIDR block with the VPC.'

The form contains the following fields and options:

- Name tag:** A text input field with the value 'US-EAST-DEV-01' and an information icon.
- IPv4 CIDR block*:** A text input field with the value '10.0.0.0/16' and an information icon.
- IPv6 CIDR block*:** Two radio button options: 'No IPv6 CIDR Block' (unselected) and 'Amazon provided IPv6 CIDR block' (selected). An information icon is next to the options.
- Tenancy:** A dropdown menu with the value 'Default' and an information icon.

At the bottom right, there are two buttons: 'Cancel' and 'Yes, Create'.

Each IPv6 enabled VPC comes with its own /56 address prefix, whereas the individual subnets created in this VPC support a /64 CIDR block.

- **DNS resolution for VPC Peering:** With DNS resolution enabled for your VPC peering, you can now resolve public DNS hostnames to private IP addresses when queried from any of your peered VPCs. This actually simplifies the DNS setup for your VPCs and enables the seamless extension of your network environments to the cloud.
- **VPC endpoints for DynamoDB:** Yet another amazing feature to be provided for VPCs later this year is the support for endpoints for your DynamoDB tables. Why is this so important all of a sudden? Well, for starters, you don't require internet gateways or NAT instances attached to your VPCs if you are leveraging the endpoints for DynamoDB. This essentially saves costs and makes the traffic between your application to the DB stay local to the AWS internal network, unlike previously where the traffic from your app would have to bypass the internet in order to reach your DynamoDB instance. Secondly, endpoints for DynamoDB virtually eliminate the need for maintaining complex firewall rules to secure your VPC. And thirdly, and most importantly, it's free!

CloudWatch

CloudWatch has undergone a lot of new and exciting changes and feature additions compared to what it originally provided as a service a few years back. Here's a quick look at some of its latest announcements:

- **CloudWatch events:** One of the most anticipated and useful features added to CloudWatch is CloudWatch events! Events are a way for you to respond to changes in your AWS environment in near real time. This is made possible with the use of event rules that you need to configure, along with a corresponding set of actionable steps that must be performed when that particular event is triggered. For example, designing a simple back-up or clean-up script to be invoked when an instance is powered off at the end of the day, and so on. You can, alternatively, schedule your event rules to be triggered at a particular interval of time during the day, week, month, or even year! Now that's really awesome!

- **High-resolution custom metrics:** We have all felt the need to monitor our applications and resources running on AWS at near real time, however, with the least amount of configurable monitoring interval set at 10 seconds, this was always going to be a challenge. But not now! With the introduction of the high-resolution custom metrics, you can now monitor your applications down to a 1-second resolution! The best part of all this is that there is no special difference between the configuration or use of a standard alarm and that of a high resolution one. Both alarms can perform the exact same functions, however, the latter is much faster than the other.
- **CloudWatch dashboard widgets:** A lot of users have had trouble adopting CloudWatch as their centralized monitoring solution due to its inability to create custom dashboards. But all that has now changed as CloudWatch today supports the creation of highly-customizable dashboards based on your application's needs. It also supports out-of-the box widgets in the form of the *number* widget, which provides a view of the latest data point of the monitored metric, such as the number of EC2 instances being monitored, or the *stacked graph*, which provides a handy visualization of individual metrics and their impact in totality.

Elastic Load Balancer

One of the most significant and useful additions to ELB over the past year has been the introduction of the Application Load Balancer. Unlike its predecessor, the ELB, the Application Load Balancer is a strict Layer 7 (application) load balancer designed to support content-based routing and applications that run on containers as well. The ALB is also designed to provide additional visibility of the health of the target EC2 instances as well as the containers. Ideally, such ALBs would be used to dynamically balance loads across a fleet of containers running scalable web and mobile applications.

This is just the tip of the iceberg compared to the vast plethora of services and functionality that AWS has added to its services in just a span of one year! Let's quickly glance through the various services that we will be covering in this book.

Introduction of newer services

The first edition of *AWS Administration - The Definitive Guide* covered a lot of the core AWS services, such as EC2, EBS, Auto Scaling, ELB, RDS, S3, and so on. In this edition, we will be exploring and learning things a bit differently by exploring a lot of the services and functionalities that work in conjunction with the core services:

- **EC2 Systems Manager:** EC2 Systems Manager is a service that basically provides a lot of add-on features for managing your compute infrastructure. Each compute entity that's managed by EC2 Systems Manager is called a *managed instance* and this can be either an EC2 instance or an on-premise machine! EC2 Systems Manager provides out-of-the-box capabilities to create and baseline patches for operating systems, automate the creation of AMIs, run configuration scripts, and much more!
- **Elastic Beanstalk:** Beanstalk is a powerful yet simple service designed for developers to easily deploy and scale their web applications. At the moment, Beanstalk supports web applications developed using Java, .NET, PHP, Node.js, Python, Ruby, and Go. Developers simply design and upload their code to Beanstalk, which automatically takes care of the application's load balancing, auto-scaling, monitoring, and so on. At the time of writing, Elastic Beanstalk supports the deployment of your apps using either Docker containers or even directly over EC2 instances, and the best part of using this service is that it's completely free! You only need to pay for the underlying AWS resources that you consume.
- **Elastic File System:** The simplest way to define **Elastic File System**, or **EFS**, is an NFS share on steroids! EFS provides simple and highly scalable file storage as a service designed to be used with your EC2 instances. You can have multiple EC2 instances attach themselves to a single EFS mount point which can provide a common data store for your applications and workloads.
- **WAF and Shield:** In this book, we will be exploring quite a few security and compliance providing services that provide an additional layer of security besides your standard VPC. Two such services we will learn about are WAF and Shield. **WAF**, or **Web Application Firewall**, is designed to safeguard your applications against web exploits that could potentially impact their availability and security maliciously. Using WAF you can create custom rules that safeguard your web applications against common attack patterns, such as SQL injection, cross-site scripting, and so on.

Similar to WAF, Shield is also a managed service that provides security against DDoS attacks that target your website or web application:

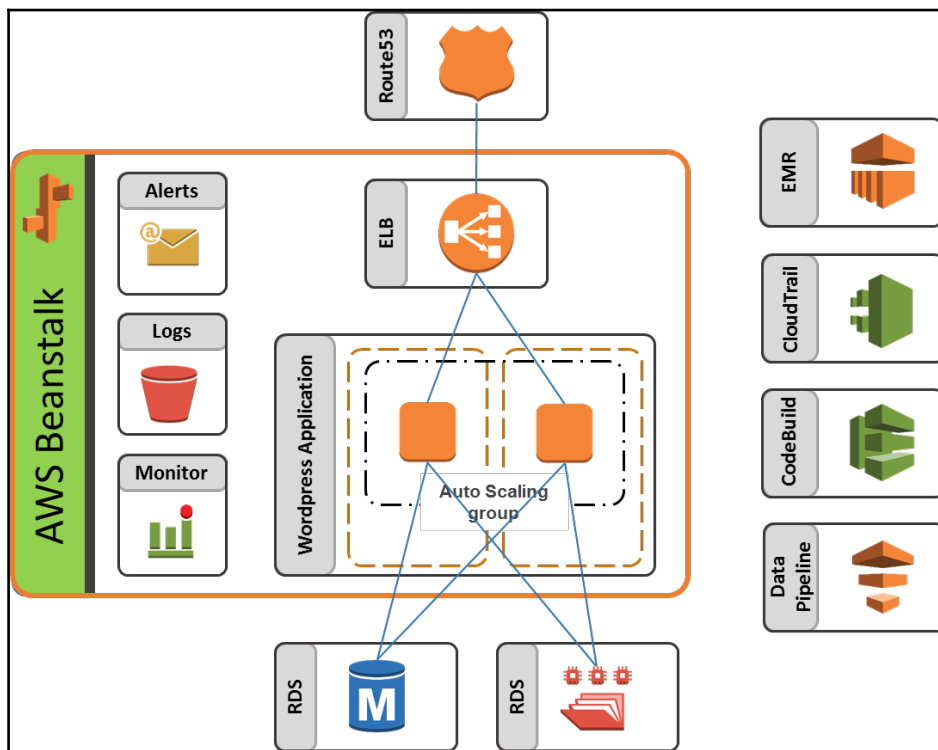
- **CloudTrail and Config:** CloudTrail is yet another service that we will learn about in the coming chapters. It is designed to log and monitor your AWS account and infrastructure activities. This service comes in really handy when you need to govern your AWS accounts against compliances, audits, and standards, and take necessary action to mitigate against them. Config, on the other hand, provides a very similar set of features, however, it specializes in assessing and auditing the configurations of your AWS resources. Both services are used synonymously to provide compliance and governance, which help in operational analysis, troubleshooting issues, and meeting security demands.
- **Cognito:** Cognito is an awesome service which simplifies the build and creation of sign-up pages for your web and even mobile applications. You also get options to integrate social identity providers, such as Facebook, Twitter, and Amazon, using SAML identity solutions.
- **CodeCommit, CodeBuild, and CodeDeploy:** AWS provides a really rich set of tools and services for developers, which are designed to deliver software rapidly and securely. At the core of this are three services that we will be learning and exploring in this book, namely CodeCommit, CodeBuild, and CodeDeploy. As the names suggest, the services provide you with the ability to securely store and version control your application's source code, as well as to automatically build, test, and deploy your application to AWS or your on-premises environment.
- **SQS and SNS:** **SQS**, or **Simple Queue Service**, is a fully-managed queuing service provided by AWS, designed to decouple your microservices-based or distributed applications. You can even use SQS to send, store, and receive messages between different applications at high volumes without any infrastructure management as well. **SNS** is a **Simple Notification Service** used primarily as a pub/ sub messaging service or as a notification service. You can additionally use SNS to trigger custom events for other AWS services, such as EC2, S3, and CloudWatch.

- **EMR: Elastic MapReduce** is a managed *Hadoop as a Service* that provides a clustered platform on EC2 instances for running Apache Hadoop and Apache Spark frameworks. EMR is highly useful for crunching massive amounts of data as well as to transform and move large quantities of data from one AWS data source to another. EMR also provides a lot of flexibility and scalability to your workloads with the ability to resize your cluster depending on the amount of data being processed at a given point in time. It is also designed to integrate effortlessly with other AWS services, such as S3 for storing the data, CloudWatch for monitoring your cluster, CloudTrail to audit the requests made to your cluster, and so on.
- **Redshift**: Redshift is a petabyte scale, managed data warehousing service in the cloud. Similar to its counterpart, EMR, Redshift also works on the concept of clustered EC2 instances on which you upload large datasets and run your analytical queries.
- **Data Pipeline**: Data Pipeline is a managed service that provides end users with an ability to process and move datasets from one AWS service to another as well as from on-premise datastores into AWS storage services, such as RDS, S3, DynamoDB, and even EMR! You can schedule data migration jobs, track dependencies and errors, and even write and create preconditions and activities that define what actions Data Pipeline has to take against the data, such as run it through an EMR cluster, perform a SQL query over it, and so on.
- **IoT and Greengrass**: AWS IoT and Greengrass are two really amazing services that are designed to collect and aggregate various device sensor data and stream that data into the AWS cloud for processing and analysis. AWS IoT provides a scalable and secure platform, using which you can connect billions of sensor devices to the cloud or other AWS services and leverage the same for gathering, processing, and analyzing the data without having to worry about the underlying infrastructure or scalability needs. Greengrass is an extension of the AWS IoT platform and essentially provides a mechanism that allows you to run and manage executions of data pre-processing jobs directly on the sensor devices.

With these services out of the way, let's quickly look at how we plan to move forward with the rest of the chapters in this book!

Plan of attack!

Just as in the previous edition, we will be leveraging a simple plan of attack even for this book! By plan of attack, I just mean how I've planned to structure the contents of the chapters and tie them all together! For the most part of the book, we will be focusing on a simple use case, such as hosting a WordPress application on AWS with the use of some really cool services in the form of Elastic Beanstalk, Elastic File System, WAF and Shield, EMR, and Redshift, and much more! Here's a simple depiction of what we will aim to achieve by the end of the book:



Here is the brief outline of how the next few chapters are spread out:

1. We will begin the setup of our WordPress by first hosting it manually over an EC2 instance as a standalone installation and then learning how to manage those instances with the help of the EC2 Systems Manager utility.
2. With this completed, we shall then use a combination of Elastic Beanstalk and Elastic File System to host the same WordPress with some more control over high availability and scalability, all the while learning the internals of both these services and use cases as we go along.
3. Now that the site is hosted, we will create an added layer of security over it by leveraging both WAF and Shield as well as enabling governance in the form of CloudTrail and Config.
4. Later we will also see how to leverage the code development services provided by AWS, namely CodeCommit, CodeBuild, and CodeDeploy, to create an effective CICD pipeline to push updates to our site.
5. Finally, we will also be executing some essential log analysis over the site using Elastic MapReduce and Redshift, and learn how to back up our site's data using Data Pipeline.
6. But that's not all! As mentioned earlier, we will also be learning about a few additional services in the form of IAM and AWS Cognito services for authentication and security, as well as AWS IoT and AWS Greengrass.

Summary

Let's quickly summarize what we have learned so far in this chapter! We started off by quickly recapping some of the key features and additions included in the core AWS services over the past few years. Remember, however, that this is in no way a complete list! There's a lot more to cover and learn, but for the sake of simplicity, I'll leave that part for self-reading. Later, we also glanced through and understood the services that are going to be included in this particular series of *AWS Administration - The Definitive Guide*. Finally, we topped it all off with a look at how we are going to structure the rest of the chapters by leveraging a simple WordPress application as a focal point for our deployments and use cases!

In the next chapter, we will kick things off by first deploying our simple WordPress application on an EC2 instance and then leverage EC2 Systems Manager along with its peripheral services for managing and tracking an EC2 instance's system configurations, so stick around! We are just getting started!

2

Managing EC2 with Systems Manager

EC2 instances have long been a core service provided by AWS and EC2 still continues to evolve with newer sets of features and instance types added every year. One such really awesome service added during AWS re:Invent 2016 was the EC2 Systems Manager!

In this chapter, we will be learning a lot about the EC2 Systems Manager and its associated sub-services; namely:

- **Run Command:** Service that allows you to execute commands directly on an EC2 Systems Manager enabled EC2 instance
- **State Manager:** Allows you to specify a desired state for an EC2 Systems Manager enabled EC2 instance
- **Patch management:** Provides administrators with the ability to manage the deployment of patches over EC2 instances
- **Automations:** Allows administrators to automate the deployment of certain tasks
- **Inventory:** Service that collects and manages a list of software inventory from your managed EC2 instances

Sound exciting? Then what are we waiting for? Let's get started!

Introducing EC2 Systems Manager

As the name suggests, EC2 Systems Manager is a management service that provides administrators and end users with the ability to perform a rich set of tasks on their EC2 instance fleet such as periodically patching the instances with a predefined set of baseline patches, tracking the instances' configurational state, and ensuring that the instance stays compliant with a state template, runs scripts and commands over your instance fleet with a single utility, and much, much more! The EC2 Systems Manager is also specifically designed to help administrators manage hybrid computing environments, all from the comfort and ease of the EC2 Systems Manager dashboard. This makes it super efficient and cost effective as it doesn't require a specialized set of software or third-party services, which cost a fortune, to manage your hybrid environments!

But how does AWS achieve all of this in the first place? Well, it all begins with the concept of managed instances. A managed instance is a special EC2 instance that is governed and managed by the EC2 Systems Manager service. Each managed instance contains a **Systems Manager (SSM)** agent that is responsible for communicating and configuring the instance state back to the Systems Manager utility. Windows Server 2003–2012 R2 AMIs, Windows Server 2003–2012 R2 AMIs will automatically have the SSM agent installed. For Linux instances, however, the SSM agent is not installed by default. Let's quickly look at how to install this agent and set up our first Dev instance in AWS as a managed instance.

Getting started with the SSM agent

In this section, we are going to install and configure an SSM agent on a new Linux instance, which we shall call as a Dev instance, and then verify it's working by streaming the agent's log files to Amazon CloudWatch Logs. So let's get busy!

Configuring IAM Roles and policies for SSM

First, we need to create and configure IAM Roles for our EC2 Systems Manager to process and execute commands over our EC2 instances. You can either use the Systems Manager's managed policies or alternatively create your own custom roles with specific permissions. For this part, we will be creating a custom role and policy.

To get started, we first create a custom IAM policy for Systems Manager managed instances:

1. Log in to your AWS account and select the **IAM** option from the main dashboard, or alternatively, open the IAM console at <https://console.aws.amazon.com/iam/>.
2. Next, from the navigation pane, select **Policies**. This will bring up a list of existing policies currently provided and supported by AWS out of the box.
3. Type *SSM* in the **Policy Filter** to view the list of policies currently provided for SSM.
4. Select the **AmazonEC2RoleforSSM** policy and copy its contents to form a new policy document. Here is a snippet of the policy document for your reference:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ssm:DescribeAssociation",
        ..... SSM actions list
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "ec2messages:AcknowledgeMessage",
        "ec2messages:DeleteMessage",
        "ec2messages:FailMessage",
        "ec2messages:GetEndpoint",
        "ec2messages:GetMessages",
        "ec2messages:SendReply"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
```

```

        "Action": [
            "ec2:DescribeInstanceStatus"
        ],
        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "ds:CreateComputer",
            "ds:DescribeDirectories"
        ],
        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "logs:CreateLogGroup",
            "logs:CreateLogStream",
            ..... CloudWatch Log actions
        ],
        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "s3:PutObject",
            "s3:GetObject",
            "s3:AbortMultipartUpload",
            "s3:ListMultipartUploadParts",
            "s3:ListBucketMultipartUploads"
        ],
        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "s3:ListBucket"
        ],
        "Resource": "arn:aws:s3:::amazon-ssm-packages-*"
    }
]
}

```

5. Once the policy is copied, go back to the **Policies** dashboard and click on the **Create policy** option. In the **Create policy** wizard, select the **Create Your Own Policy** option.
6. Provide a suitable **Policy Name** and paste the copied contents of the **AmazonEC2RoleforSSM** policy into the **Policy Document** section. You can now tweak the policy as per your requirements, but once completed, remember to select the **Validate Policy** option to ensure the policy is semantically correct.
7. Once completed, select **Create Policy** to complete the process.

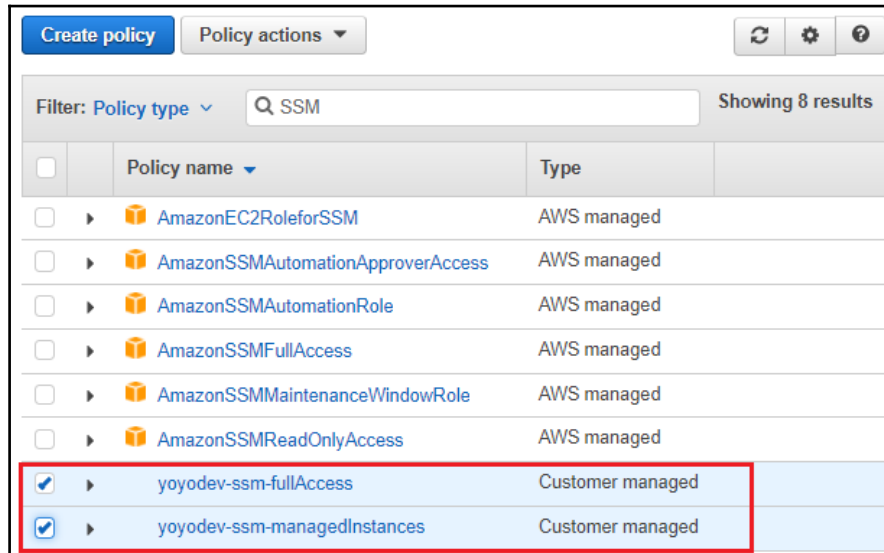
With this step completed, you now have a custom IAM policy for System Manager managed instances.

The next important policy that we need to create is the custom IAM user policy for our Systems Manager. This policy will essentially scope out which particular user can view the System Manager documents as well as perform actions on the selected managed instances using the System Manager's APIs:

1. Once again, log in to your AWS IAM dashboard and select the **Policies** option as performed in the earlier steps.
2. Type **SSM** again in the **Policy Filter** and select the **AmazonSSMFullAccess** policy. Copy its contents and create a custom SSM access policy by pasting the following snippet in the new policy's **Policy Document** section:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:PutMetricData",
        "ds:CreateComputer",
        "ds:DescribeDirectories",
        "ec2:DescribeInstanceStatus",
        "logs:*",
        "ssm:*",
        "ec2messages:*"
      ],
      "Resource": "*"
    }
  ]
}
```

3. Remember to *validate* the policy before completing the creation process. You should now have two custom policies, as shown in the following screenshot:



With the policies created, we now simply create a new instance profile role, attach the full access policy to the new role, and finally verify the trust relationship between Systems Manager and the newly created role:

1. To create a new role, from the IAM management dashboard, select the **Roles** option from the navigation pane.
2. In the **Create Role** wizard, select the **EC2** option from the **AWS service** role type, as shown in the following screenshot. Next, select the **EC2** option as the *use case* for this activity and click on the **Next: Permissions** button to continue:

Select role type


AWS service


Another AWS account


Web identity


Saml 2.0 federation

Allows AWS services to perform actions on your behalf. [Learn more](#)
Choose the service that will use this role

API Gateway

Data Pipeline

IoT

Service Catalog

CloudFormation

EC2

Machine Learning

DMS

Greengrass

SWF

Select your use case

EC2
Allows EC2 instances to call AWS services on your behalf.

[EC2 Role for Simple Systems Manager](#)
Provides EC2 Instances access to Amazon Simple Systems Manager (SSM), CloudWatch, EC2, and supported plugins in SSM documents.

[EC2 Spot Fleet Role](#)
Allows EC2 Spot Fleet to request and terminate Spot Instances on your behalf.

* Required

Cancel **Next: Permissions**

3. In the **Attach permissions policy** page, filter and select the **ssm-managedInstances** policy that we created at the beginning of this exercise. Click on **Review** once done.
4. Finally, provide a suitable **Role name** in the **Review** page and click on **Create role** to complete the procedure!

With the role in place, we now need to verify that the IAM policy for your instance profile role includes `ssm.amazonaws.com` as a trusted entity:

1. To verify this, select the newly created role from the **IAM Roles** page and click on the **Trust relationships** tab.
2. Here, choose the **Edit Trust Relationship** option and paste the following snippet in the policy editor, as shown. Remember to add both *EC2* and *SSM* as the trusted services and not just one of them:

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

3. With the new trust policy in place, click on **Update Trust Policy** to complete the process. Congratulations!
4. You are almost done with configuring the Systems Manager! A final step remains, where we need to attach the second policy that we created (SSM full access) to one of our IAM users. In this case, I've attached the policy to one of my existing users in my AWS environment, however, you can always create a completely new user dedicated to the Systems Manager and assign it the SSM access policy as well.

With the policies out of the way, we can now proceed with the installation and configuration of the SSM agent on our simple Dev instance.

Installing the SSM agent

As discussed at the beginning of the chapter, the Systems Manager or the SSM agent is a vital piece of software that needs to be installed and configured on your EC2 instances in order for Systems Manager to manage it. At the time of writing, SSM agent is supported on the following sets of operating systems:

- **Windows:**
 - Windows Server 2003 (including R2)
 - Windows Server 2008 (including R2)
 - Windows Server 2012 (including R2)
 - Windows Server 2016
- **Linux (64-bit and 32-bit):**
 - Amazon Linux 2014.09, 2014.03 or later
 - Ubuntu Server 16.04 LTS, 14.04 LTS, or 12.04 LTS
 - Red Hat Enterprise Linux (RHEL) 6.5 or later
 - CentOS 6.3 or later
- **Linux (64-bit only):**
 - Amazon Linux 2015.09, 2015.03 or later
 - Red Hat Enterprise Linux 7.x or later
 - CentOS 7.1 or later
 - SUSE Linux Enterprise Server 12 or higher

To install the agent on a brand new instance, such as the one we will create shortly, you simply need to ensure that the instance is provided with the necessary SSM IAM role that we created in the previous section, as well as to provide the following code snippet in the **User data** section of your instance's configuration:

```
#!/bin/bash
cd /tmp
wget
https://s3.amazonaws.com/ec2-downloads-windows/SSMAgent/latest/debian_amd64
/amazon-ssm-agent.deb
sudo dpkg -i amazon-ssm-agent.deb
sudo start amazon-ssm-agent
```



The user data script varies from OS to OS. In my case, the script is intended to run on an Ubuntu Server 14.04 LTS (HVM) instance. You can check your SSM agent install script at <http://docs.aws.amazon.com/systems-manager/latest/userguide/sysman-install-ssm-agent.html#sysman-install-startup-linux>.

Once the instance is up and running, SSH into the instance and verify whether your SSM agent is up and running or not using the following command. Remember, the following command will also vary based on the operating system that you select at launch time:

```
# sudo status amazon-ssm-agent
```

You should see the agent running, as shown in the following screenshot:

```
ubuntu@ip-192-168-32-188:~$  
ubuntu@ip-192-168-32-188:~$ sudo status amazon-ssm-agent  
amazon-ssm-agent start/running, process 1494  
ubuntu@ip-192-168-32-188:~$  
ubuntu@ip-192-168-32-188:~$ █
```

You can, optionally, even install the agent on an existing running EC2 instance by completing the following set of commands.

For an instance running on the Ubuntu 16.04 LTS operating system, we first create a temporary directory to house the SSM agent installer:

```
# mkdir /tmp/ssm
```

Next, download the operating-specific SSM agent installer using the `wget` utility:

```
# wget  
https://s3.amazonaws.com/ec2-downloads-windows/SSMAgent/latest/debian_amd64/  
amazon-ssm-agent.deb
```

Finally, execute the installer using the following command:

```
# sudo dpkg -i amazon-ssm-agent.deb
```

You can additionally verify the agent's execution by tailing either of these log files as well:

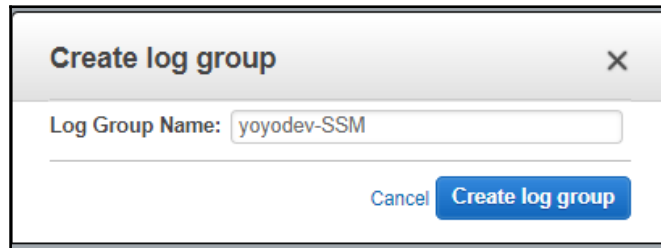
```
# sudo tail -f /var/log/amazon/ssm/amazon-ssm-agent.log  
# sudo tail -f /var/log/amazon/ssm/errors.log
```

Configuring the SSM agent to stream logs to CloudWatch

This is a particularly useful option provided by the SSM agent, especially when you don't want to log in to each and every instance and troubleshoot issues. Integrating the SSM agent's logs with CloudWatch enables you to have all your logs captured and analyzed at one central location, which undoubtedly ends up saving a lot of time, but it also brings additional benefits such as the ability to configure alarms, view the various metrics using CloudWatch dashboard, and retain the logs for a much longer duration.

But before we get to configuring the agent, we first need to create a separate log group within CloudWatch that will stream the agent logs from individual instances here:

1. To do so, from the **AWS Management Console**, select the **CloudWatch** option, or alternatively, click on the following link to open your CloudWatch dashboard from <https://console.aws.amazon.com/cloudwatch/>.
2. Next, select the **Logs** option from the navigation pane. Here, click on **Create log group** and provide a suitable name for your log group, as shown in the following screenshot:



3. Once completed, SSH back into your Dev instance and run the following command:

```
# sudo cp /etc/amazon/ssm/seelog.xml.template  
/etc/amazon/ssm/seelog.xml
```

4. Next, using your favorite editor, open the newly copied file and paste the following content in it. Remember to swap out the `<CLOUDWATCH_LOG_GROUP_NAME>` field with the name of your own log group:

```
# sudo vi /etc/amazon/ssm/seelog.xml  
<seelog minlevel="info" critmsgcount="500" maxinterval="100000000"  
mininterval="2000000" type="adaptive">
```

```

<exceptions>
<exception minlevel="error" filepattern="test*" />
</exceptions>
<outputs formatid="fmtinfo">
<console formatid="fmtinfo" />
<rollingfile type="size" maxrolls="5" maxsize="30000000"
filename="{{LOCALAPPDATA}}\Amazon\SSM\Logs\amazon-ssm-agent.log" />
<filter formatid="fmterror" levels="error,critical">
<rollingfile type="size" maxrolls="5" maxsize="10000000"
filename="{{LOCALAPPDATA}}\Amazon\SSM\Logs\errors.log" />
</filter>
<custom name="cloudwatch_receiver" formatid="fmtdebug" data-log-
group="<CLOUDWATCH_LOG_GROUP_NAME>" />
</outputs>
CODE:

```

5. With the changes made, save and exit the editor. Now have a look at your newly created log group using the CloudWatch dashboard; you should see your SSM agent's error logs, if any, displayed there for easy troubleshooting.

With this step completed, we have now successfully installed and configured our EC2 instance as a **Managed Instance** in Systems Manager. To verify whether your instance has indeed been added, select the **Managed Instance** option provided under the **Systems Manager Shared Resources** section from the navigation pane of your EC2 dashboard; you should see your instance listed, as shown here:

	Name	Instance ID	Ping status	Platform Type	Platform Name
☒	yoyodev-us-east-01	i-0a097f4a8e41a6ad0	Online	Linux	Ubuntu

Managed Instance: i-0a097f4a8e41a6ad0

Description	Inventory	Associations	Patch	Configuration Compliance
Instance ID i-0a097f4a8e41a6ad0 Ping Status ● Online Platform Name Ubuntu IP Address 192.168.32.188 Resource Type EC2Instance				

In the next section, we will deep dive into the various features provided as a part of the Systems Manager, starting off with one of the most widely used: Run Command!

Introducing Run Command

Run Command is an awesome feature of Systems Manager, which basically allows you to execute remote commands over your managed fleet of EC2 instances. You can perform a vast variety of automated administrative tasks, such as installing software or patching your operating systems, executing shell commands, managing local groups and users, and much more! But that's not all! The best part of using this feature is that it allows you to have a seamless experience when executing scripts, even over your on-premises Windows and Linux operating systems, whether they be running on VMware ESXi, Microsoft Hyper-V, or any other platforms. And the cost of all this? Well, it's absolutely free! You only pay for the EC2 instances and other AWS resources that you create and nothing more!

Here's a brief list of a few commonly predefined commands provided by Run Command along with a short description:

- `AWS-RunShellScript`: Executes shell scripts on remote Linux instances
- `AWS-UpdateSSMAgent`: Used to update the Amazon SSM agent
- `AWS-JoinDirectoryServiceDomain`: Used to join an instance to an AWS Directory
- `AWS-RunPowerShellScript`: Executes PowerShell commands or scripts on Windows instances
- `AWS-UpdateEC2Config`: Runs an update to the EC2Config service
- `AWS-ConfigureWindowsUpdate`: Used to configure Windows Update settings
- `AWS-InstallApplication`: Used to install, repair, or uninstall software on a Windows instance using an MSI package
- `AWS-ConfigureCloudWatch`: Configures Amazon CloudWatch Logs to monitor applications and systems

Before we proceed with the actual execution of the Run Commands, it is important to remember that the Run Command requires both the SSM agent as well as the right set of permissions and roles to work with. So if you haven't performed the SSM agent's installation or the setup of the IAM policies and roles, then now would be a good time to revisit this!

In this section, let's look at a simple way of executing a simple set of commands for our newly added managed instance:

1. To begin with, first log in to the **AWS Management Console** and select the **EC2** service from the main dashboard. Alternatively, you can even launch the EC2 dashboard via <https://console.aws.amazon.com/ec2/>.
2. Next, from the navigation pane, select the **Run Command** option from the **Systems Manager Services** section. You will be taken to the **Run Command** dashboard where you will need to select the **Run a command** option to get started.
3. In the **Run a command** page, the first thing we need to do is select a **Command document** that we can work with. A command document is basically a statement or set of information about the command you want to run on your managed instances. For this scenario, we will select the `AWS-RunShellScript` command document to start with.
4. In the next **Select Targets by** section, you can optionally choose whether you wish to execute your command document manually by selecting individual instances or specify a particular group of instances identified by their *tag* name.
5. The **Execute on** criteria provides you with the option to select either the **Targets** or **Percent** of instances you wish to execute the command document on. Selecting **Targets** allows you to specify the exact number of instances that should be allowed to execute the command document. The execution occurs on each instance one at a time. Alternatively, if you select the **Percent** option, then you can provide a percentage value of the instances that should be allowed to run the command at a single time.
6. You can optionally set the **Stop after x errors** to halt the execution of your command document in case an instance encounters an error.
7. Finally, you can paste your execution code or shell script in the **Commands** section as shown in the following screenshot. In this case, we are running a simple script that will install and configure a Zabbix monitoring agent on our Dev instance for easy monitoring of our EC2 resources:

Select Targets by* ☒ Manually Selecting Instances ☐ Specifying a Tag

i-0a097f4a8e41a6ad0 ⓘ

Select instances ▼

Execute on 1 Targets ▼ concurrently

Stop after 2 errors

Commands* `sudo wget http://repo.zabbix.com/zabbix/3.2/ubuntu/pool/main/z/zabbix-release/zabbix-release_3.4-1+xenial_all.deb`
`sudo dpkg -i zabbix-release_3.4-1+xenial_all.deb`
`sudo apt-get update -y`
`sudo apt-get install zabbix-agent -y`
`sudo bash -c "cat > /etc/zabbix/zabbix_agentd.conf <<EOF`
`PidFile=/var/run/zabbix/zabbix_agentd.pid`
`LogFile=/var/log/zabbix/zabbix_agentd.log`
`LogFileSize=0`
`Server=192.168.32.50 # Private IP of my Zabbix Server on EC2`

Working Directory /tmp ⓘ

Execution Timeout 3600 ⓘ



You can learn more about Zabbix and its features at <https://www.zabbix.com/product>.

- Copy and paste the following code snippet or, alternatively, tweak it according to the EC2 instance operating system that you may have selected for this exercise:

```
sudo wget
http://repo.zabbix.com/zabbix/3.2/ubuntu/pool/main/z/zabbix-release
/zabbix-release_3.4-1+xenial_all.deb
sudo dpkg -i zabbix-release_3.4-1+xenial_all.deb
sudo apt-get update -y
sudo apt-get install zabbix-agent -y

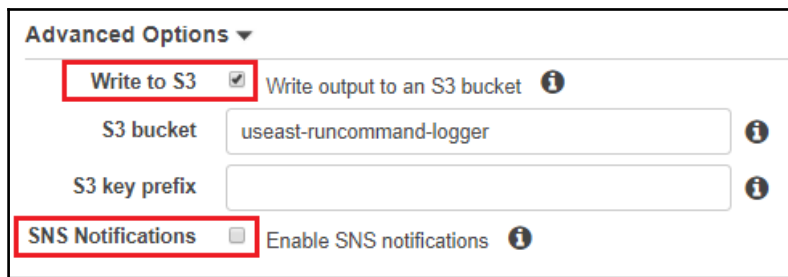
sudo bash -c "cat > /etc/zabbix/zabbix_agentd.conf <<EOF
PidFile=/var/run/zabbix/zabbix_agentd.pid
LogFile=/var/log/zabbix/zabbix_agentd.log
LogFileSize=0
Server=192.168.32.50 # Private IP of my Zabbix Server on EC2
ServerActive=192.168.32.50 # Private IP of my Zabbix Server on EC2
```

```
Include=/etc/zabbix/zabbix_agentd.d/*.conf
EOF"
```

```
sudo service zabbix-agent status
sudo service zabbix-agent restart
```

9. The rest of the options provide other configurational items such as setting up an optional *working directory* where the commands get executed on the remote managed instances.

Additionally, you can even choose to **Enable SNS notifications** as well as write your command output logs to S3 using the **Advanced Options** sections, as shown here:



Advanced Options ▼

Write to S3 ☒ Write output to an S3 bucket ⓘ

S3 bucket ⓘ

S3 key prefix ⓘ

SNS Notifications ☐ Enable SNS notifications ⓘ

10. Once the configuration items are filled in, simply select the **Run** option to start the execution of your command document. During this time, Systems Manager will invoke the execution of your supplied commands over the list of managed instances that you provided. If there is an error during the execution, Systems Manager will halt the execution and display the **Status** of your output as either **Success** or **Failed**.

Simple isn't it? You can use this same mechanism to manage and execute commands remotely over your fleet of EC2 instances with ease and consistency and even leverage the AWS CLI to perform the same set of actions we have explored in this section.

In the next section, we will be learning a bit about yet another really useful feature provided by Systems Manager: State Manager.

Working with State Manager

State Manager is a powerful tool that helps to govern and manage the configuration of a managed system. For example, by using State Manager you can enforce a particular firewall rule for your fleet of managed instances and set that as the required **State** that needs to be enforced at all times. If the rules change outside of State Manager, it will automatically revert to match the required state's configuration, thus maintaining compliance and enforcing standardization over your environment.

Working with State Manager is quite simple and straightforward. You start off by selecting a state document (JSON based) that specifies the settings you need to configure or maintain your EC2 instances. These documents come predefined and you can create customized versions of them. With the document created, you can then select the individual managed instances, which can be either EC2 instances or even on-premises virtual machines, as well as specify a schedule for when and how often you wish to apply these states. It's that simple!

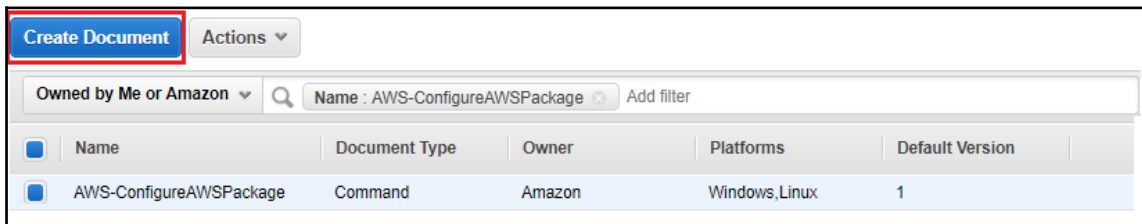
But before we go ahead with the invocation of our State Manager, let's first understand the concept of state documents a bit better as these documents are the foundation on which your Systems Manager works.

State documents are nothing more than simple JSON-based steps and parameters that define certain actions to be performed by Systems Manager. AWS provides dozens of such documents out of the box, which can be used to perform a variety of tasks such as patching your instances, configuring certain packages, configuring the CloudWatch Log agents, and much more! Additionally, you can even create your own custom document as well! There are three types of documents that are supported by Systems Manager:

- **Command:** Command documents are leveraged by the Run Command to execute commands over your managed instances. Alternatively, State Manager uses the command documents to apply certain policies as well. These actions can be run on one or more targets at any point during the life cycle of an instance.
- **Policy:** Used by the State Manager, policy documents are used to enforce a policy on your managed instances.
- **Automation:** These documents are more often used by the automation service within Systems Manager to perform common maintenance and deployment tasks. We will be learning more about automation documents a bit later in this chapter.

To view System Manager's predefined documents, from the EC2 dashboard navigation pane, select the **Documents** option under the **Systems Manager Shared Resources** section. Here you can use any of the predefined documents as per your requirements for State Manager, however let's quickly create a very simple custom document based on the `aws:configurePackage` definition:

1. To create your own document, select the **Create Document** option from the **Documents** dashboard as shown here:



2. In the **Create Document** wizard, start off by providing a suitable **Name** for your document. In this case, I've provided the name `yoyodev-ssm-configure-packages`. Do note that the name cannot contain any spaces.
3. Next, from the **Document Type** dropdown, select **Command** as the option type and paste the following JSON code in the **Content** section as shown here:

```
{
  "schemaVersion": "2.0",
  "description": "Install or uninstall the latest version or
specified version of LAMP stack.",
  "parameters": {
    "action": {
      "description": "(Required) Specify whether or not to
install or uninstall the package.",
      "type": "String",
      "allowedValues": [
        "Install",
        "Uninstall"
      ]
    },
    "name": {
      "description": "(Required) The LAMP package to
install/uninstall.",
      "type": "String",
      "allowedValues": [
        "apache2",
        "mysql-server",
```

```

        "php"
    ]
},
"version": {
    "description": "(Optional) A specific version of the
package to install or uninstall.",
    "type": "String",
    "default": "",
    "allowedPattern":
"^(?:(\\d+)\\.)(?:(\\d+)\\.)(\\d+)$|^$"
}
},
"mainSteps": [{
    "action": "aws:configurePackage",
    "name": "configurePackage",
    "inputs": {
        "name": "{{ name }}",
        "action": "{{ action }}",
        "version": "{{ version }}"
    }
}]
}

```

4. With the document pasted, you can now click on **Create Document** to complete the document creation process.

The document comprises two primary sections: a `parameters` section, which contains a list of actions to be performed by the document, followed by a `mainSteps` section that specifies the action, which in this case is the `aws:configurePackage` to be performed by the document. In this case, the document when invoked will ask the user to select either `apache2`, `mysql-server`, or `php` from the dropdown list followed by an optional version number of the software you select. You can then select whether you wish to install or uninstall this particular package from your fleet of managed EC2 instances and simply execute the document when done!

Now that your custom document is created, let's quickly configure the State Manager to invoke it:

1. From the **Systems Manager Services** section in the EC2 navigation pane, select the **State Manager**. In the State Manager dashboard, select the **Create Association** option to get started with configuring State Manager.
2. Provide a suitable **Association Name** for your association. Note that this is an optional field and you can skip it if you want.

- Next, from the **Select Document** section, filter and select the custom document that we created in our earlier step. On selection, you will notice the subfields change according to what we provided as parameters in the document. Let's quickly configure this and create our association.
- In the **Targets** section, select your Dev instance or any of your managed instances which you wish to associate with this State Manager. Finally, go ahead and configure the **Schedule** that will trigger the association based on either a CRON or a rate schedule.
- Last but not the least, configure the **Action** and select the appropriate package **Name** from the **Parameters** section as shown in the following screenshot:

To run an association automatically set a schedule defining when it will run.

Specify with

- ☒ Cron schedule builder
- ☐ Rate schedule builder

Association runs

- ☐ Every 30 Minutes
- ☐ Every 1 Hours
- ☒ Every Sunday at 12:00 UTC

Parameters

Action* Install ⓘ

Name* mysql-server ⓘ

Version ⓘ

Advanced

Write to S3 ☐

[Cancel](#) [Create Association](#)

- You can optionally enable the **Write to S3** checkbox to log the State Manager's execution in your own custom S3 bucket. For this scenario, I have not selected this option.
- Finally, complete the State Manager's association process by selecting the **Create Association** option.

You can now view and modify your associations using the State Manager dashboard. Alternatively, you can even choose to enable your association immediately by selecting the **Apply Association Now** option as well.

In the next section, we will be looking at yet another simple and easy-to-use feature provided by Systems Manager that helps automate simple instance and deployment tasks, called System Manager Automation!

Simplifying instance maintenance using System Manager Automation

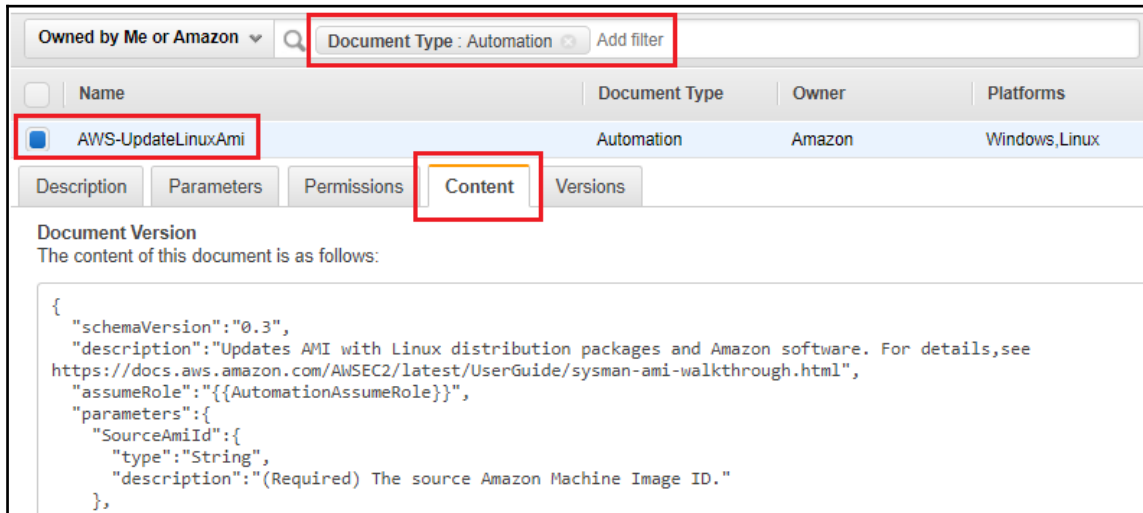
System Manager Automation is a managed service that provides a single, centralized interface for executing and monitoring commonly occurring instance management tasks such as patching, performing backups, executing scripts, and much more. Let's first get started by understanding a few necessary prerequisites that are required to be configured in order for automation to work in your environments.

Working with automation documents

As discussed briefly during the introduction to the State Manager service, automation documents are simple JSON-based documents that are designed to help you get started with the automation service quickly and efficiently. You can leverage the predefined automation documents or, alternatively, create your own set. In this section, we will look at how to leverage an existing automation document to patch your Dev EC2 instance and create a new AMI from it:

1. From the **EC2 Management Console**, select the **Documents** option from the **Systems Manager Shared Resources** section.
2. Using the **Documents** dashboard, you can filter and view only the documents that have **Automation** set as the **Document Type**.

3. Select **AWS-UpdateLinuxAmi** and click on the **Content** tab to view the automation document as shown here:



The **AWS-UpdateLinuxAmi** document comprises five distinctive steps, each explained briefly here:

- **launchInstance:** This step basically launches a new EC2 instance using your Systems Manager IAM instance profile as well as with a user data script that will install the latest copy of the SSM agent on this instance. The SSM agent is vital as it will enable the next steps to be executed using the Run Command as well as State Manager.
- **updateOSSoftware:** With the instance launched and the SSM agent installed, the next step is responsible for updating the packages in your Linux instance. This is done by executing an update script that methodologically updates the packages and any other software that may be marked for upgrades. You also get the capability to include or exclude a particular set of packages from this step using the `IncludePackages` and `ExcludePackages` parameters respectively. If no packages are included, the program updates all available packages on the instance.
- **stopInstance:** Once the instance is updated with the latest set of packages, the next action simply powers off the instance so that it can be prepped for the image creation process.

- **createImage:** This step creates a new AMI from your updated Linux instance. The image contains a descriptive name that links it to the source ID and creation time of the image.
- **terminateInstance:** The final step in the automation document, this step essentially cleans up the execution by terminating the running Linux instance.

Let's look at few simple steps using which we can invoke this particular automation document manually using the automation dashboard.

Patching instances using automation

In this section, we will be manually invoking the **AWS-UpdateLinuxAmi** automation document for patching our Linux instance and later creating a new AMI out of it:

1. To do this, first select the **Automations** option present under the **Systems Manager Services** section.
2. From the **Automations** dashboard, select the **Run automation document** option.
3. From the **Document name** field, select the **AWS-UpdateLinuxAmi** document and populate the required fields in the **Input parameters** section as described here:
 - **SourceAmiId:** Provide the source Amazon Machine Image ID from which the new instance will be deployed.
 - **InstanceIamRole:** Provide the IAM role name that enables Systems Manager to manage the instance. We created this role earlier during the start of this chapter as a part of SSM's prerequisites.
 - **AutomationAssumeRole:** Provide the ARN of the IAM role that allows automation to perform the actions on your behalf.
 - **TargetAmiName:** This will be the name of the new AMI created as a part of this automation document. The default is a system-generated string including the source AMI ID and the creation time and date.
 - **InstanceType:** Specify the instance type of instance to launch for the AMI creation process. By default, the *t2.micro* instance type is selected.
 - **PreUpdateScript:** You can additionally provide the URL of a script to run before any updates are applied. This is an optional field.
 - **PostUpdateScript:** Provide an optional post update script URL of a script to run after package updates are applied.

- **IncludePackages:** Include specific packages to be updated. By default, all available updates are applied.
 - **ExcludePackages:** Provide names of specific packages that you wish to exclude from the updates list.
4. With the fields populated, simply select the **Run automation** option as shown in the following screenshot:

Variable name	Type	Description	Value
SourceAmiId	String	(Required) The source Amazon Machine Image ID.	ami-841f46ff
InstanceIamRole	String	(Required) The name of the role that enables Systems Manager (SSM) to manage the instance.	devyoyo-ssm-role
AutomationAssumeRole	String	(Required) The ARN of the role that allows Automation to perform the actions on your behalf.	arn:aws:iam::{{global.ACC
TargetAmiName	String	(Optional) The name of the new AMI that will be created. Default is a system-generated string including the source AMI id, and the creation time and date.	UpdateLinuxAmi_from_{{S
InstanceType	String	(Optional) Type of instance to launch as the workspace host. Instance types vary by region. Default is t2.micro.	t2.micro
PreUpdateScript	String	(Optional) URL of a script to run before updates are applied. Default ("none") is to not run a script.	none
PostUpdateScript	String	(Optional) URL of a script to run after package updates are applied. Default ("none") is to not run a script.	none
IncludePackages	String	(Optional) Only update these named packages. By default ("all"), all available updates are applied.	all
ExcludePackages	String	(Optional) Names of packages to hold back from updates, under all conditions. By default ("none"), no package is excluded.	none

[Cancel](#)
[Run automation](#)

5. The automation document takes a couple of minutes to completely execute. You can verify the output of the execution using the **Automations** dashboard as well.
6. Simply select your automation job **Execution ID** to view the progress of each individual step as shown in the following screenshot. Optionally, you can verify the output of each step by selecting the adjoining **View Outputs** link as well:

Automation execution: c26bf711-a051-11e7-aebc-97d467435815

[Description](#)
[Steps](#)
[Inputs](#)

Name	Action Type	Status	Start Time	Stop Time	Output
launchInstance	aws:runInstances	Success	September 23, 2017 at 1:24:28 PM UTC+2	September 23,...	View Outputs
updateOSSoft...	aws:runCommand	Success	September 23, 2017 at 1:30:38 PM UTC+2	September 23,...	View Outputs
stopInstance	aws:changeInstanceState	Success	September 23, 2017 at 1:31:21 PM UTC+2	September 23,...	View Outputs
createImage	aws:createImage	Success	September 23, 2017 at 1:33:26 PM UTC+2	September 23,...	View Outputs
terminateInsta...	aws:changeInstanceState	Success	September 23, 2017 at 1:35:31 PM UTC+2	September 23,...	View Outputs

With this completed, you can now run similar automation tasks by creating your own automation documents and executing them using the steps mentioned herein. But what if you wanted to trigger these steps based on some events or schedules? Well, that's exactly what we will look into in the next section, *Triggering automation using CloudWatch schedules and events*.

Triggering automation using CloudWatch schedules and events

Although you can trigger automation documents manually, it's far better to either schedule or automate the execution of automation jobs using CloudWatch schedules and events.

Let's first understand how you can leverage CloudWatch events to trigger simple notifications of Systems Manager Automation events. These events can be used to notify you of whether your automation task succeeded, failed, or simply timed out:

1. First, log in to the CloudWatch dashboard. Alternatively, you can open CloudWatch via <https://console.aws.amazon.com/cloudwatch/>.
2. Next, from the navigation pane, select the **Events** option to bring up the **Create rule** page. Here, select **Event Pattern** from the **Event Source** section.
3. With this done, we now need to build our event source. To do so, from the **Service Name** drop-down list, search and select the option **EC2 Simple Systems Manager (SSM)**, as shown here:

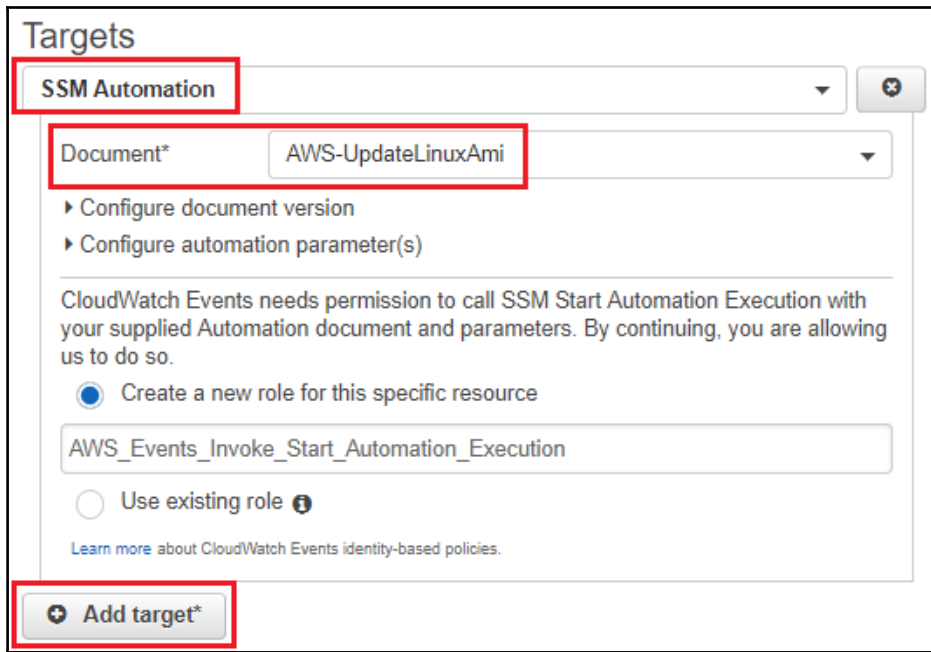
The screenshot displays the 'Create rule' page in the AWS CloudWatch console. The 'Event Pattern' radio button is selected and highlighted with a red box. Below it, the 'Build event pattern to match events by service' section is visible. The 'Service Name' dropdown is set to 'EC2 Simple Systems Manager (SSM)' and the 'Event Type' dropdown is set to 'Automation', both highlighted with red boxes. Under the 'Specific detail type(s)' section, the 'EC2 Automation Execution Status-change Notification' is selected and highlighted with a red box. Finally, in the 'Specific status(es)' section, the 'Failed' status is selected and highlighted with a red box.

4. With the service selected, you can now opt to select a corresponding SSM **Event Type** as well, for example in this case I wish to be notified when a particular **Automation** task fails. So in the **Event Type** drop-down list, I've selected the **Automation** option. You can alternatively select other SSM services as well.
5. Next, in the detail type section, I've opted to go for the **EC2 Automation Execution Status-change Notification** option. Correspondingly, I've also selected **Failed** as the **Specific status(es)** for my event. This means that if and when a failed status event is generated as a result of an automation job, it will trigger a corresponding action which can be as simple as sending a notification using an SNS service or even triggering a corresponding Lambda function to perform some form of remediation action.
6. Your **Event Pattern Preview** should resemble something similar to the snippet here:

```
{
  "source": [
    "aws.ssm"
  ],
  "detail-type": [
    "EC2 Automation Step Status-change Notification",
    "EC2 Automation Execution Status-change Notification"
  ]
}
```

Similarly, you can even configure a CRON expression or fixed rate of execution of your automation jobs by selecting the **Schedule** option in the **Event Source** section:

1. Provide a suitable **Cron expression** depending on your requirements, for example, I wish to run the **AWS-UpdateLinuxAmi** automation document every Sunday at 10 P.M. UTC. In this case, the CRON expression will become `0,18,?,*,SUN,*.`
2. With the schedule configured, move on to the **Targets** section and select the **SSM Automation** option from the **Targets** drop-down list as shown in the following screenshot:



Targets

SSM Automation

Document* AWS-UpdateLinuxAmi

- ▶ Configure document version
- ▶ Configure automation parameter(s)

CloudWatch Events needs permission to call SSM Start Automation Execution with your supplied Automation document and parameters. By continuing, you are allowing us to do so.

☒ Create a new role for this specific resource

AWS_Events_Invoke_Start_Automation_Execution

☐ Use existing role ⓘ

[Learn more](#) about CloudWatch Events identity-based policies.

+ Add target*

3. Next, configure the **AWS-UpdateLinuxAmi** parameters as we discussed earlier, and once the desired fields are populated, click on **Add target*** to complete the configuration.

With this step completed, you can now instantaneously trigger your automation jobs based on events as well as schedules, all powered by CloudWatch! Amazing isn't it?

In the next and final section, we will be learning a bit about yet another simple and easy to use SSM service that enables you to manage and patch your instances with ease.

Managing instance patches using patch baseline and compliance

Regularly patching your instances with the right set of security patches is an important activity that can take up a lot of time and effort if performed manually on each individual instance. Luckily, AWS provides a really efficient and easy way of automating the patching of your managed instances using the concept of Patch Manager services, provided as an out-of-the-box capability with SSM.

As an administrator, all you need to do is scan your instances for missing patches and leverage Patch Manager to automatically remediate the issues by installing the required set of patches. You can, alternatively, even schedule the patching of your managed instance or group of instances with the help of SSM's maintenance window tasks.

In this section, we will explore a quick and easy way of creating a unique patch baseline for our Dev instances and later create and associate a maintenance window for this, all using the EC2 Management dashboard. So let's get started with this right away!

First up, you will need to ensure that your instance has the required set of IAM Roles as well as the SSM agent installed and functioning as described at the beginning of this chapter. With these basics out of the way, we first need to configure the patch baseline with our set of required patches:

1. To do so, launch your EC2 dashboard and select the **Patch Baselines** option from the **Systems Manager Services** section. Patch Manager includes a default patch baseline for each operating system supported by Patch Manager. This includes Windows Server 2003 to 2016, Ubuntu, RHEL, CentOS, SUSE, and even Amazon Linux as well. You can use these default patch baselines or alternatively you can create one based on your requirements. Here, let's quickly create a custom baseline for our Dev instances.
2. Select the **Create Patch Baseline** option to bring up the **Create Patch Baseline** dashboard. Here, provide a suitable **Name** for your custom baseline.
3. From the **Operating System**, select **Ubuntu** as the OS choice. You will notice the patching rules change accordingly based on the OS type you select.

- Next, in the **Approval Rules** section, create suitable patch baseline rules depending on your requirements. For example, I wish to set the Python packages to an **Important** priority and with a **High** compliance level as well. Similarly, you can add up to 10 such rules for one baseline, as shown in the following screenshot:

A Patch Baseline defines Patch Approval Rules and Patch Exceptions.

Name*

Description

Operating System ⓘ

Approval Rules

Product	Section	Priority	Compliance Level	
Ubuntu14.04	python	Important	High	✕
Ubuntu14.04	universe/python	Important	High	✕
Ubuntu14.04	doc	Standard	Medium	✕

7 remaining

- In the final section, **Patch Exceptions**, you can optionally mention the **Approved Packages**, **Rejected Packages**, and the **Compliance Level** for these patches collectively. In this case, I've left these values as their defaults and selected the **Create Patch Baseline** option to complete the process.

With your new patch baseline created, you now have the option to promote the same as the **Default Baseline** by selecting the new baseline from the **Patch Baselines** dashboard and clicking on the **Set Default Patch Baseline** option from the **Actions** tab.

Moving on to the next part of this walkthrough, we will now go ahead and set up the maintenance window for our newly created patch baseline:

- To do so, select the **Maintenance Windows** option from the **Systems Manager Shared Resources** section. Click on **Create maintenance window** to get started with the process.
- In the **Create maintenance window** page, provide a suitable **Name** for your window as well as an optional **Description**.

- Next, in the **Specify schedule** section, you can opt to either use a *CRON scheduler* or a *rate expression* to define the schedule for your maintenance window. For this scenario, I've opted for the **Cron schedule builder** option and provided a window that starts every Sunday at 12:00 UTC:

Specify schedule

Specify with ☒ Cron schedule builder ☐ Rate schedule builder ☐ CRON/Rate expression

Window starts ☐ Every 30 Minutes ☐ Every 1 Hours ☒ Every Sunday at 12:00 UTC

Duration* 2 hours

Stop initiating tasks* 1 hour before the window closes

► AWS Command Line Interface command

Cancel Create maintenance window

- In the **Duration** as well as the **Stop initiating tasks** field, specify the timeline in hours that the maintenance window has to last for, as well as the number of hours before you want the system to stop initiating new tasks. Once all the required fields are populated, click on **Create maintenance window** to complete the creation process.

With the maintenance window created, we next need to add some targets for execution. Targets are individual EC2 instances or a group of EC2 instances that are identified by tags. To configure targets, select your newly created maintenance window then from the **Actions** tab and select the option **Register targets**:

- In the **Register targets** page, provide a **Target Name** for your maintenance window's target with an optional **Description**.
- Next, select the target EC2 instances you wish to associate with this target by either opting to **Specify Tags** or even by **Manually Selecting Instances** as shown in the following screenshot. For this scenario, I've already provided the tag *OS:Linux* to my Dev instances; alternatively, you can manually select your instances as well:

Maintenance window mw-05bc51ca25c0d704a (dev-ubuntu-14-base-maintenance)

Target Name dev-ubuntu-14-Target

Description Ubuntu 14.04 dev instances

Owner information

Targets

Targets are the instances you would like to register with maintenance window. You can choose to target by both managed instance and tag.

Select Targets by ☒ Specifying Tags ☐ Manually Selecting Instances

Select tag key pairs to add targets that are tagged with these key pairs:

Tag Name	Tag Value
OS	Linux

Add Tag 4 remaining

Cancel **Register targets**

3. Once completed, select the **Register targets** option to complete the process.

With the target instances registered with our maintenance window, the final step left to take is associate the maintenance window with our patch baseline:

1. In order to do this, we need to select the newly created *maintenance window*; from the **Actions** tab, select the option **Register run command task**.
2. Here, in the **Register run command task** page, fill in the required details such as a *name* for your new Run Command followed by an optional **Description**.
3. Next, from the **Command document** section, select the **AWS-RunPatchBaseline** document. You will also see the targeted instance associated with this Run Command already, as we configured it in our earlier steps.
4. Finally, in the **Parameters** section, select the appropriate IAM Role, provide a suitable count for the *Run Command to stop after* receiving a certain amount of errors, and last but not least, don't forget to select whether you wish to **Install** or simply **Scan** the target instances for the required set of patches.
5. With all the fields completed, click on **Register task** to complete this configuration.

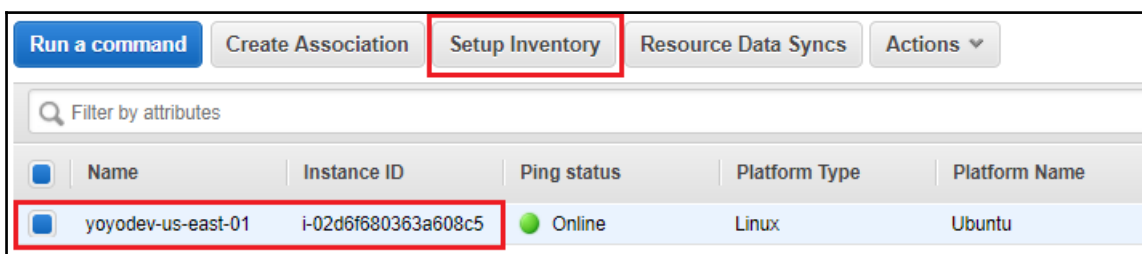
Awesome isn't it? With just a few simple clicks you have now set up an effective patch management solution for your Dev instances, and without the need for any specialized software or expertise! But before we wind up this chapter, let's look at one last simple and really useful service provided by Systems Manager, which helps collect and inventory metadata about your AWS as well as on-premises instances.

Getting started with Inventory Management

Inventory Management or just Inventory is yet another managed service provided by Systems Manager that is responsible for collecting operating system, application, and instance metadata from your AWS instances as well as those present and managed by Systems Manager in your on-premises environments. You can use this service to query the inventory metadata for mapping, understanding, and remediating EC2 instances based on certain software or regulatory compliances.

Let's look at a very simple example of enabling the inventory service for our Dev instance using the AWS Management dashboard:

1. To begin with, you will require both the SSM agent as well as the required IAM Roles configured on your managed instance. Once this is completed, select the **Managed Instances** option from the **Systems Manager Shared Resources** section.
2. Here, select your Dev instance and click on the **Setup Inventory** option as shown in the following screenshot:



3. On the **Setup Inventory** page, most of the options will be quite familiar to you by now, such as the **Targets** and **Schedule** sections, so I'm not going to dwell on them here again. The more important section here is the **Parameters** section, using which you can choose to either **Enable** or **Disable** different types of inventory collections. For example, since we are working with Linux instances, I've chosen to disable the *Windows updates* parameters while keeping the rest enabled.
4. The final field **Write execution history to S3** basically allows you to write and store the inventory data centrally in an S3 bucket. This comes in really handy when you wish to collate your inventory data from multiple instances at a central location and then query this data either using services such as Amazon Athena or QuickInsights. Once completed, click on **Setup Inventory** to complete the inventory setup process.

You can now view the collected metadata of your EC2 instance by selecting it from the **Managed Instances** page and clicking on the **Inventory** tab. Here, choose between the various **Inventory Types** drop-down lists to view your instance specific metadata. You can toggle between **AWS:Application**, **AWS:AWSComponent**, **AWS:Network**, and **AWS:InstanceDetailedInformation**, just to name a few.

With this, we come towards the end of this chapter, but before we move on to the next chapter, here are a few key takeaways and points that you ought to try out!

Planning your next steps

Well, we have covered a lot of new features and services in this chapter, however, there are still a few things that I would recommend you try out on your own. First up is the Parameter Store!

The Parameter Store is yet another managed service provided by Systems Manager and is designed to store your instance's configuration data such as passwords, database strings, license codes, and so on, securely. You have the added option of storing the data either as plain text, or even in encrypted form, and later reference it in your automation documents or policy documents as variables rather than complete plain text. But that's not even the best part of it! Using Parameter Store, you can also reference your data across other AWS services such as EC2 Container Service and AWS Lambda, thus making this a centralized store for storing all your configurational and secure data.

Another important feature that I would recommend using in your environments is Resource Data Sync. Resource Data Sync enables you to store your instance's metadata collected from different Systems Manager services in a centralized location. You can configure it to collect metadata from instance operating systems, applications, Microsoft Windows updates, network configurations, and much more! This comes in really handy in production environments where you want to analyze the software and patch compliances of your instances.

Summary

Well, it has certainly been a long chapter, but we got to learn a lot and I hope that it helps you in your quest towards mastering AWS! Let's quickly recap what we have learned so far.

We started off with a quick introduction to Systems Manager, followed by an in-depth look at how to install, configure and verify the SSM agents on your managed EC2 instances. We then gained an understanding of how to leverage the Run Command to perform certain types of executions on your managed instances, followed by a rundown of maintaining an instance's state configuration using State Manager. We also looked at working with automation documents and how you can effectively patch your instances using them. Towards the end of the chapter, we learned about patching your instances using Patch Manager and also learned how you can effectively inventory your instance metadata using inventory manager.

In the next chapter, we will be introducing two new services, which will help us to deploy our simple WordPress application to the cloud. So, stay tuned!

3

Introducing Elastic Beanstalk and Elastic File System

In the previous chapter, we started off by learning a lot about an awesome managed service called Systems Manager, which can perform virtually any and all tasks related to your EC2 instances, such as automating script executions, patching your instances, maintaining state and compliance, and much more. In this chapter, we will take things up a notch by introducing two really awesome services: Elastic Beanstalk, a service that can help you develop and deploy rich web applications in just a few clicks, and Elastic File System, a service that provides a massively scalable shared filesystem for your EC2 instances! So, keeping this in mind, let's have a quick look at the various topics that we will be covering in this chapter:

- Introducing Elastic Beanstalk and how it works
- How to manage applications, environments, and configurations with ease using Elastic Beanstalk
- Pushing your applications to AWS using the Elastic Beanstalk CLI
- Getting started with Elastic File System and its various use cases
- Hosting a highly scalable and available WordPress site using Elastic Beanstalk and Elastic File System

So much to do, so let's get started right away!

Introducing Amazon Elastic Beanstalk

One of the key features of a cloud is to provide its users and developers with a seamless and easy to use platform for developing and deploying their applications. That's exactly where Elastic Beanstalk comes in. Elastic Beanstalk was first launched in the year 2011, and has continuously evolved to become a full-fledged PaaS offering from AWS.

Elastic Beanstalk is your one-stop shop for quickly deploying and managing your web applications in AWS. All you need to do is upload your code to Beanstalk, and voila! Elastic Beanstalk takes care of the entire application's deployment process, from EC2 capacity provisioning to auto-scaling the instances and even load balancing using an ELB! Elastic Beanstalk does it all so that you can concentrate on more important tasks, such as developing your applications and not getting bogged down with complex operational nuances.

But for me, Beanstalk is much more than just the deployment and management of your applications. Let's look at some of the key benefits of leveraging Elastic Beanstalk for your web applications:

- **Deployment support:** Today, Beanstalk supports standard EC2 instances and Docker containers as the basis for your application's deployment. This enables you to host your web applications and your microservices-based apps on AWS with relative ease.
- **Platform support:** Beanstalk provides a rich set of platforms for developers to deploy their apps on. Today, the list includes Java, PHP, Python, .NET, Node.js, and Ruby, with more languages and platforms to be added in the future.
- **Developer friendly:** It is extremely easy to build and deploy your applications over to AWS using Beanstalk. You can leverage a wide variety of options, including the AWS Management Console or its CLI, a code repository such as Git, or even an IDE such as Eclipse or Visual Studio to upload your application, and the rest is all taken care of by Beanstalk itself.
- **Control:** With Beanstalk, you get complete control over your underlying AWS resources as well as the environments on which your application runs. You can change the instance types, scale the resources, add more application environments, configure ELBs, and much more!
- **Costs:** One of the best things about Beanstalk is that it's absolutely free! Yes, you heard it right! Free! You only pay for the AWS resources that are spun up based on the configurations that you provide and nothing more. Amazing isn't it?

With these pointers in mind, let's look at some of the essential concepts and terminologies that you ought to know before getting started with Elastic Beanstalk.

Concepts and terminologies

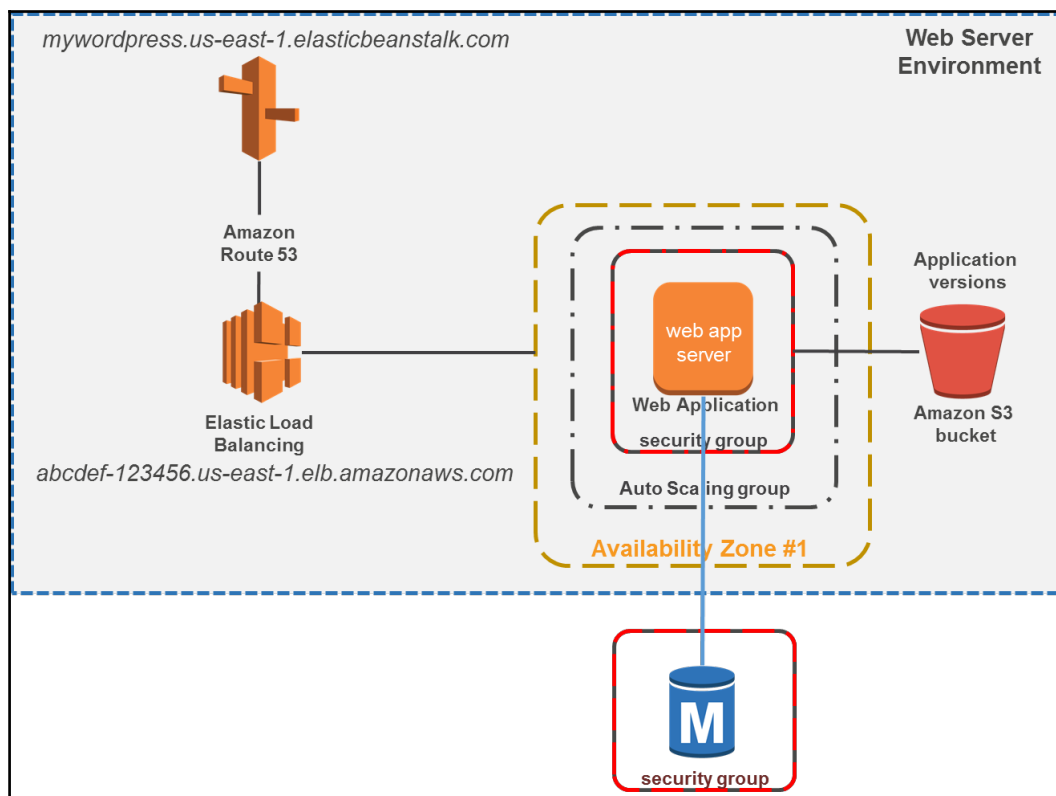
Here's a look at some of the common concepts and terminologies that you will often come across while working with Elastic Beanstalk:

- **Applications:** An application in Elastic Beanstalk is basically a collection of Beanstalk's internal components, and includes environments, versions, events, and various other things. Think of an Elastic Beanstalk application as a high-level container which contains different aspects of your application.
- **Application versions:** Application versions are nothing more than different versions of an application's code. Each version of your application's code is stored in an S3 Bucket that is auto-created and managed by Beanstalk itself. You can create multiple versions of your application code and use this for deployment to one or more environments for testing and comparison.
- **Environments:** An Elastic Beanstalk environment is yet another logical container that hosts one application version at a time on a specified set of instances, load balancers, auto scaling groups, and so on. Typically, you would have an environment for development, one for acceptance testing, and another one for production hosting, however, there are no hard and fast rules on this.

An environment comes in two flavors, and you can choose between the two during your initial environment setup phase. The first is called a **web server environment**, and is basically created for applications that support HTTP requests, such as web applications and so on. The second is called a **worker environment**, where the application pulls tasks from an Amazon SQS Queue. Here's a look at each of these flavors in a bit more detail:

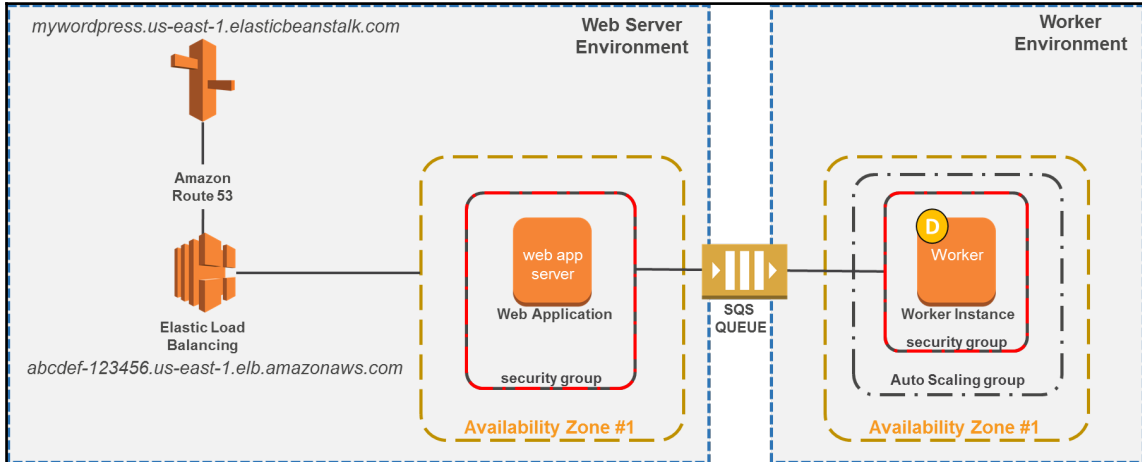
- **Web server environment:** As mentioned earlier, this particular environment is well suited to hosting and managing web frontend applications, such as websites, mobile applications, and so on. As part of this environment, Beanstalk provisions an internet-facing Elastic Load Balancer, an autoscaling group with some minimalistic configuration settings, and a small number of EC2 instances that contain your application code along with a pre-installed agent called **Host Manager**. The Host Manager agent is a key component in the entire setup process, as it is responsible for deploying and monitoring the application as well as periodically patching the instance and rotating the logs.

Here's a representational diagram depicting a simple application being scaled using a web server environment. Note the RDS instance in the diagram as well. You can also choose to set up an RDS instance for your application using Elastic Beanstalk, or add it to the application stack manually later:



An additional point worth mentioning here is that every environment has a unique CNAME, for example `mywordpress`. The CNAME maps to a URL which is in the form of `mywordpress.us-east-1.elasticbeanstalk.com`. This URL is aliased in Amazon's DNS service Route53 to an Elastic Load Balancing URL, something like `abcdef-123456.us-east-1.elb.amazonaws.com`, by using a CNAME record.

- **Worker environment:** The worker environment works in a very different way to the web server environment. In this case, Elastic Beanstalk starts up an SQS Queue in your environment and installs a small daemon into each of the worker instances. The daemon is responsible for regularly polling the queue for newer messages, and if a message is present, the daemon pulls it into the worker instance for consumption, as depicted in the following diagram:



Ideally, you can use a combination of web and worker environments to host your applications, so that there is a clear decoupling of your web frontend resources and your backend processing worker instances. Keep in mind that there are a lot more design considerations that you also ought to think about while setting up your environments, such as how the scalability is going to be handled and storage options for data depending on the type of data, for example S3 for logs and RDS for application-centric data, security, fault tolerance, and much more.

With this section completed, let's move on to the fun part and see how to get started with using Elastic Beanstalk!

Getting started with Elastic Beanstalk

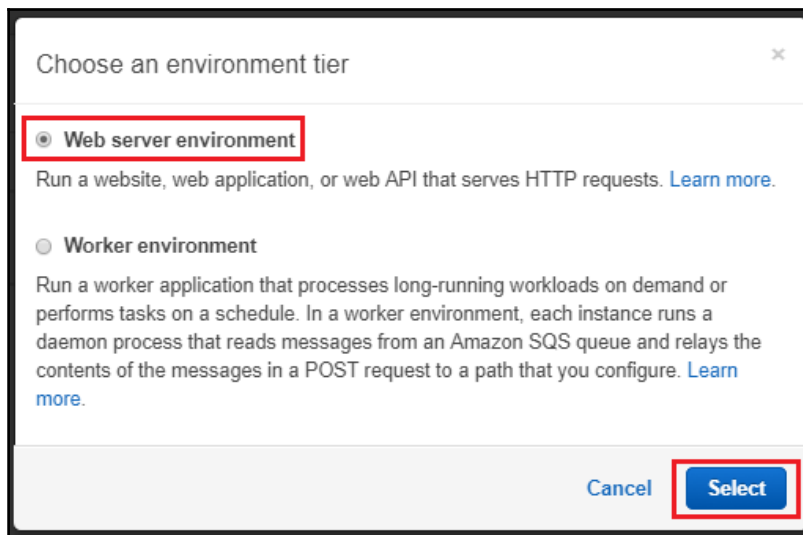
In this section, we will be performing a deep dive into how to set up a fully-functional Dev and Prod environment for our simple WordPress application using Elastic Beanstalk. Before we get started, here is a list of some prerequisite items that you need to have in place before we can proceed:

- A valid AWS account and user credentials with the required set of privileges to run the AWS CLI and the Elastic Beanstalk CLI.
- A sandbox/Dev instance to download the WordPress installation and later use it to push the application code over to the respective Beanstalk environment. Note that you can also use other resources, such as a Git URL or an IDE, but for now we will be focusing on this approach.

Creating the Dev environment

Let's first start off by creating a simple and straightforward development environment for our WordPress site. To do so, execute the following steps:

1. Sign in to the AWS Console and select the Elastic Beanstalk option from the **Services** filter, or alternatively, launch the Elastic Beanstalk console by launching the URL <https://console.aws.amazon.com/elasticbeanstalk> in a browser of your choice.
2. Next, select the **Create New Application** option to get started. Remember, an application is the highest level of container for our application code, which can contain one or more environments as required.
3. In the **Create New Application** dialog box, provide a suitable **Application Name** and an optional **Description** to get started with. Click on **Create** once completed.
4. With the basic application container created, you can now go ahead and create the development environment. To do so, from the **Actions** drop-down list, select the **Create New Application** option.
5. Here, you will be provided with an option to either opt for the **Web server environment** or the **Worker environment** configuration. Remember, an environment type can be selected only once here, so make sure that you select the correct tier based on your application's requirements. In this case, I've opted to select the **Web server environment**, as shown in the following screenshot:



6. On the **Create a new environment** wizard page, provide a suitable **Environment name** for your WordPress site. Since this is a development environment, I've gone ahead and named it `YoyoWordpress-dev`. Next, in the **Domain** field, provide a unique name for your website's domain URL. The URL will be suffixed by the region-specific Elastic Beanstalk URL, as shown in the following screenshot:

Create a new environment

Environment information

Choose the name, subdomain, and description for your environment. These cannot be changed later.

Application name `yoyo-Wordpress`

Environment name `YoyoWordpress-dev`

Domain `yoyoclouds` `.us-east-1.elasticbeanstalk.com` [Check availability](#)

`yoyoclouds.us-east-1.elasticbeanstalk.com` is available

Description `Dev Environment for my Wordpress site`

- Next, type in a suitable **Description** for your new environment, and move on toward the **Base configuration** section. Here, from the **Platform** dropdown, select the **Preconfigured platform** option and opt for the **PHP** platform, as depicted in the following screenshot. PHP is our default option as WordPress is built on PHP 5.6. Today, Beanstalk supports packer builder, Docker containers, Go, Java SE, Java with Tomcat, .NET on Windows Server with IIS, Node.js, PHP, Python, and Ruby, with more platform support coming shortly:



Base configuration

Tier Web Server (Choose tier)

Platform ☒ Preconfigured platform

Platforms published and maintained by AWS Elastic Beanstalk.

PHP

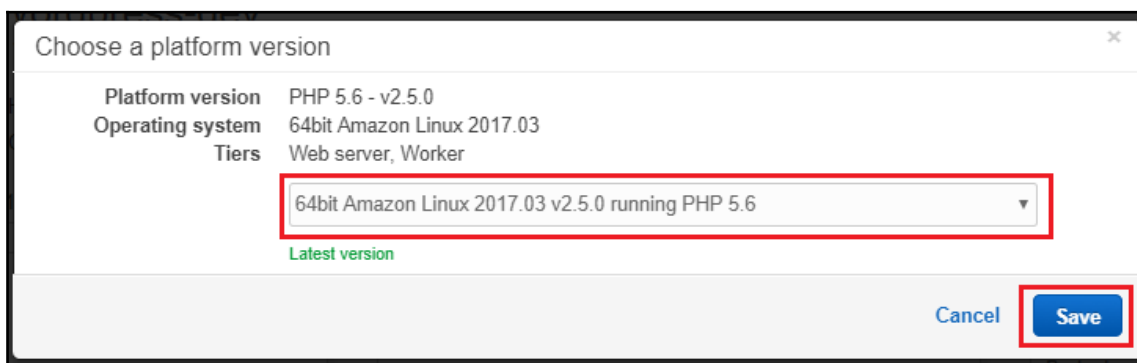
- Now, here's the part where you need to keep your calm! Leave the rest of the options as their default values and select **Configure more options**, not the **Create environment** option! Yes, we will be configuring a few additional items first and will create our development environment later!
- On the **Configure Environment** page, you can opt to select one of the three preconfigured **Configuration presets** options, based on your application's requirements. The presets are briefly explained here:
 - **Low cost (Free tier eligible):** This particular configuration will launch a single (**t2.micro**) instance with no load balancing or autoscaling group configured. This is ideal if you just want to get started with an application using the basics or wish to set up a minimalistic Dev environment, as in this case.
 - **High availability:** Unlike the low-cost preset, the high-availability configuration comes pre-equipped with an autoscaling group that can scale up to a default of four instances or more, and an Elastic Load Balancer that has cross-zone load balancing and connection draining enabled by default. Besides this, you also get a host of CloudWatch alarms created for monitoring, as well as security groups for your instance and Load Balancer.
 - **Custom configuration:** You can additionally opt to configure your environment based on other parameters. You can select this preset, and modify each and every component present within your environment as you see fit.

10. With the **Configuration preset** set to **Low cost (Free tier)**, the next item that we can modify is the **Platform configuration**. Elastic Beanstalk supports the following PHP platform configurations:

PHP Language	Amazon Linux AMI	PHP version
PHP 7.1	2017.03.1	PHP 7.1.7
PHP 7.0	2017.03.1	PHP 7.0.21
PHP 5.6	2017.03.1	PHP 5.6.31
PHP 5.5	2017.03.1	PHP 5.5.38
PHP 5.4	2017.03.1	PHP 5.4.45

11. Since we are using a WordPress application, we need to modify the platform as well, to accommodate for the correct PHP version.

To do so, select the **Change platform configuration** option. This will bring up the **Choose a platform version** dialog, as shown here:



12. Here, from the drop-down list, search and select the **64bit Amazon Linux 2017.03 v2.5.0 running PHP 5.6** option, as WordPress execution is stable with PHP 5.6. Once done, click on **Save** to complete the process.
13. With the **Platform configuration** changed as per our requirements, we can now move on to configuring the add-on services such as **security, notifications, network, database**, and much more! For example, let's quickly configure the networking for our WordPress Dev environment by selecting the **Modify** option in the **Network** pane.

14. In the **Network** pane, you can opt to launch your environment in a custom **VPC**, as well as other instance-specific settings such as enabling **Public IP address**, selecting the **Instance subnets** based on your VPC design, and finally assigning **Instance security groups** for your Dev instances. In this case, I already have a custom VPC created specifically for the development environment that contains one public subnet and one private subnet, with a default security group as well. Here is an overview of the network configuration setup for my environment. You can tweak this to match your requirements:

Public IP address ☒ Assign a public IP address to the Amazon EC2 instances in your environment.

Instance subnets

	Availability Zo...	Subnet	CIDR	Name
☒	us-east-1b	subnet-9850b0d3	192.168.32.0/24	yoyodev-public-01
☐		subnet-5a5aba11	192.168.64.0/24	yoyodev-private-01

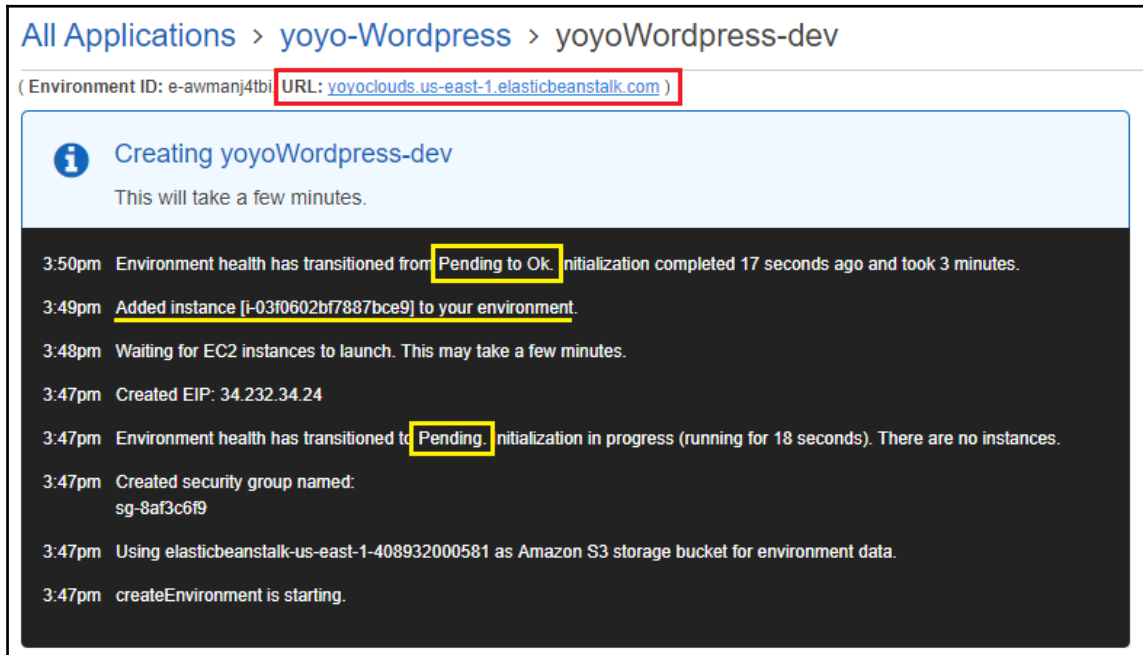
Instance security groups

	Group name	Group ID	Name
☒	default	sg-1daa486e	

Cancel Save

15. Once the settings are made, click on **Save** to complete the networking changes. You can perform other configurational changes as you see fit, however, since this is only a development environment, I've opted to leave the rest of the options as default for now. Once completed, select the **Create environment** option to finish the environment creation process.

Once the environment creation process is initiated, it will take a couple of minutes to complete its execution, as depicted in the following screenshot. Here, you will see Elastic Beanstalk create a new security group as well as an Elastic IP address for your EC2 dev instance. During this stage, the environment also transitions from a **Pending** to an **Ok** state, and you can view the environment, your application's logs, and the status:



With your environment up and running, you can also verify it using the URL provided as an output of your environment's creation by using the environment dashboard. Upon selecting the URL, you will be redirected to a new application landing page in your web browser which basically verifies that your environment is configured to work with PHP 5.6.

But where is our WordPress application? That's exactly what we will be deploying next using a really simple and easy-to-use Elastic Beanstalk CLI.

Working with the Elastic Beanstalk CLI

With your environment deployed using the AWS Management Console, we now shift our focus to leveraging the Elastic Beanstalk CLI, or EB CLI, to push the application code over to the newly created environment.

The EB CLI is a powerful utility that can be used to operate and manage your entire Elastic Beanstalk environment using a few simple CLI commands. It is also designed to work with AWS development services such as CodeBuild and CodeCommit, as well as other third-party code repository services such as Git.

In this section, we will first be looking at a few simple steps for installing the EB CLI on a simple Linux instance, later followed by configuring and pushing our WordPress application to its respective development environment:

1. To do so, we first need to ensure that the instance is updated with the latest set of packages. In my case, I'm performing the steps on a simple Ubuntu 14.04 LTS instance, however, you can alternatively use your own on-premises virtual machines.
2. Run the following command to update your OS. The command will vary based on your operating system variant:

```
# sudo apt-get update
```

3. Next, we need to ensure that the instance has the required Python packages installed in it. Note that if you are using the Amazon Linux AMI, then by default it will already have the necessary Python packages installed in it:

```
# sudo apt-get install python python-pip
```



AWS CLI and the EB CLI require Python 2 version 2.6.5+ or Python 3 version 3.3+.

4. With the Python packages installed, we now move forward and set up the AWS CLI using the following commands:

```
# pip install awscli  
# aws configure
```



The first command installs the CLI, while the other runs you through a simple wizard to set up the AWS CLI for your instance. To learn more about how to configure the AWS CLI, you can check out this URL: <http://docs.aws.amazon.com/cli/latest/userguide/installing.html>.

5. Finally, we go ahead and install the EB CLI. The installation is pretty straightforward and simple:

```
# pip install awsebcli
```

6. That's all there is to it! You now have a functioning Elastic Beanstalk CLI installed and ready for use. So let's now go ahead and download the required WordPress code ZIP file locally and use the EB CLI to push it into the development environment:

```
# sudo git clone https://github.com/WordPress/WordPress.git
```

7. Extract the contents of your WordPress ZIP file into a new folder, and run the following command from within the WordPress directory:

```
# eb init
```

The `eb init` command is used to initialize and sync the EB CLI with your newly created development environment. Follow the on-screen instructions to configure the EB CLI's settings, such as **Selecting a default region** to operate from, **Selecting an application to use**, and so on. Remember, the default region has to match your current development environment's region as well, which in my case is **us-east-1**:

1. With the EB CLI set up, the only step left now is to deploy the WordPress application to the development environment using yet another EB CLI command called simply `eb deploy`:

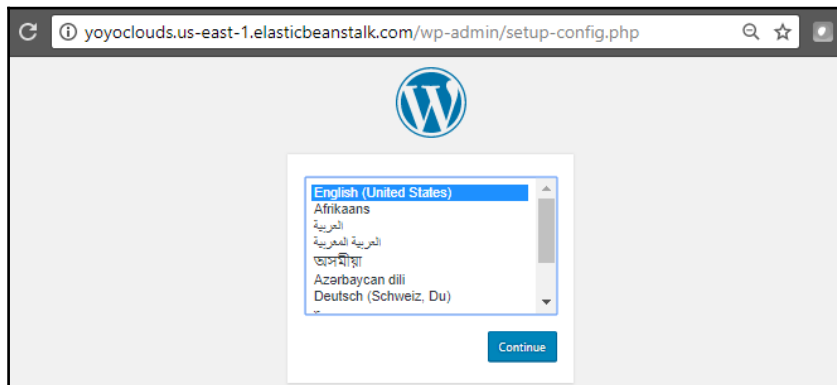
```
# eb deploy
```

During the deployment process, the CLI creates an application version archive in a new S3 bucket within your environment. Each application deployment will result in subsequent version creations within S3 itself. After this, you will see your application code get uploaded to your development environment, as depicted in the following screenshot:

```
root@YoYoNux:~/WordPress#
root@YoYoNux:~/WordPress# eb deploy
Creating application version archive "app-a83e4-171007_132524".
Uploading: [#####] 100% Done...
INFO: Environment update is starting.
INFO: Deploying new version to instance(s).
INFO: New application version was deployed to running EC2 instances.
INFO: Environment update completed successfully.

root@YoYoNux:~/WordPress#
```

2. The environment simultaneously changes its state from **Ok** to **Pending** as the application is uploaded and set up in your development instance. Once the application becomes available, the health state of the environment will yet again transition from **Pending** to **OK** to **Info**.
3. You can verify whether your application has uploaded or not by refreshing the application URL (`yoyoclouds.us-east-1.elasticbeanstalk.com`) on your environment's dashboard. You should see the WordPress welcome screen, as shown in the following screenshot:



Note, however, that this setup still requires a MySQL database, so don't forget to go to the RDS Management Console and create a minimalistic MySQL database, or even better, an Aurora DB instance, for your development environment. Remember to note down the database username, password, and the DB host and database name itself; you will need these during your WordPress configuration!

With this step completed, let's take a few minutes to understand the various options for configuring and monitoring your newly deployed application using the environment dashboard!

Understanding the environment dashboard

The environment dashboard is your one-stop shop for managing and monitoring your newly deployed applications, as well as the inherited instances. In this section, we will quickly look at each of the sections present in the environment dashboard and how you can leverage them for your applications.

To start off with, the **Dashboard** view itself provides you with some high-level information and event logs depicting the current status of your environment. To learn more about the recent batch of events, you can opt to select the **Show All** option in the **Recent Events** section, or alternatively select the **Events** option from the navigation pane.

The **Dashboard** also allows you to upload a newer version of your application by selecting the **Upload and Deploy** option, as shown in following screenshot. Here, you can see a **Running Version** of your WordPress application as well. This is the same application that we just deployed using the EB CLI.

You can also control various aspects of your environment, such as **Save Configuration**, **Clone Environment**, and **Terminate Environment**, as well using the **Actions** tab provided in the right-hand corner of the environment dashboard:

The screenshot shows the AWS Elastic Beanstalk environment dashboard for an environment named 'YoyoWordpress-dev'. The breadcrumb navigation at the top reads 'All Applications > yoyo-Wordpress > YoyoWordpress-dev'. The environment ID is 'e-vp5zjrkczz' and the URL is 'yoyoclouds.us-east-1.elasticbeanstalk.com'. The 'Actions' dropdown menu is visible in the top right corner.

The left-hand navigation pane contains the following items: Dashboard (highlighted with a red box), Configuration, Logs, Health, Monitoring, Alarms, Managed Updates, Events, and Tags.

The main content area is titled 'Overview' and includes a 'Refresh' button. It displays the following information:

- Health:** A green checkmark icon and the text 'Ok'. A 'Causes' button is located below the status.
- Running Version:** The version identifier 'app-a83e4-171007_132524'. Below it is a button labeled 'Upload and Deploy' (highlighted with a red box).
- Configuration:** Details include '64bit Amazon Linux 2017.03 v2.5.0 running PHP 7.1'. A 'Change' button is located below the configuration details.

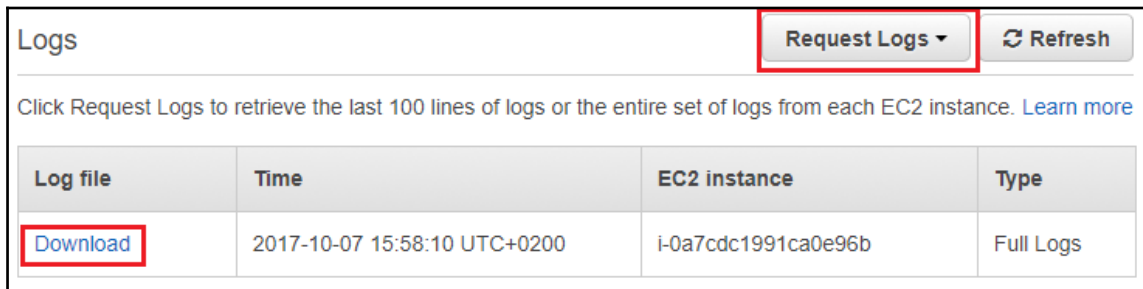
The 'Recent Events' section at the bottom contains a 'Show All' button (highlighted with a red box) and a table of events.

Time	Type	Details
2017-10-07 13:29:44 UTC+0200	INFO	Pulled logs for environment instances.

Moving on from the **Dashboard**, the next tab in the navigation pane that is worth checking out is the **Configuration** section. Let's look at each of the configuration options in a bit more detail, starting off with the **Scaling** tile:

- **Scaling:** Here, you can opt to change your **Environment Type** from a **Single instance** deployment to a **Load balancing, auto scaling** enabled environment simply by selecting the correct option from the **Environment Type** drop-down list. You can even enable **Time-based scaling** for your instances by opting for the **Add scheduled action** option.
- **Instances:** In the next tile, you can configure your instance-specific details for your environment, such as the **Instance type**, the **EC2 key pair** to be used for enabling SSH to your instances, the **Instance profile**, and other options as well, such as the root volume type and its desired size.
- **Notifications:** Here, you can specify a particular **Email** address, using which, notifications pertaining to your environment—such as its events—are sent using the Amazon SNS.
- **Software configuration:** This section allows you to configure some key parameters for your application, such as the application's **Document root**, the **Memory limit** for running your PHP environment, and the logging options. But the thing that I really love about the software configuration is the **Environment properties** section. With this, you can pass secrets, endpoints, debug settings, and other information to your application without even having to SSH into your instances, which is simply amazing! We will be learning a bit about environment properties and how you can create simple environment variables and pass them to your WordPress application a bit later in this chapter.
- **Health:** One of the most important configuration items in your environment, the **Health** section allows you to configure the **Health Check URL** for your application, as well as to enable detailed health reporting for your environment using a special agent installed on your systems. This agent monitors the vitals of your EC2 instance, captures application-level health metrics, and sends them directly to Beanstalk for further analysis. This, in conjunction with the **Application Logs**, helps you to drill down into issues and mitigate them all using the Elastic Beanstalk Console itself.
- **NOTE:** You can find the agent's logs in your instance's `/var/log/healthd/daemon.log` file.

Apart from the **Configuration** tab, Elastic Beanstalk also provides you with a **Logs** option, where you can request either the complete set of logs or the last 100 lines. You can download each instance's log files using this particular section as well:

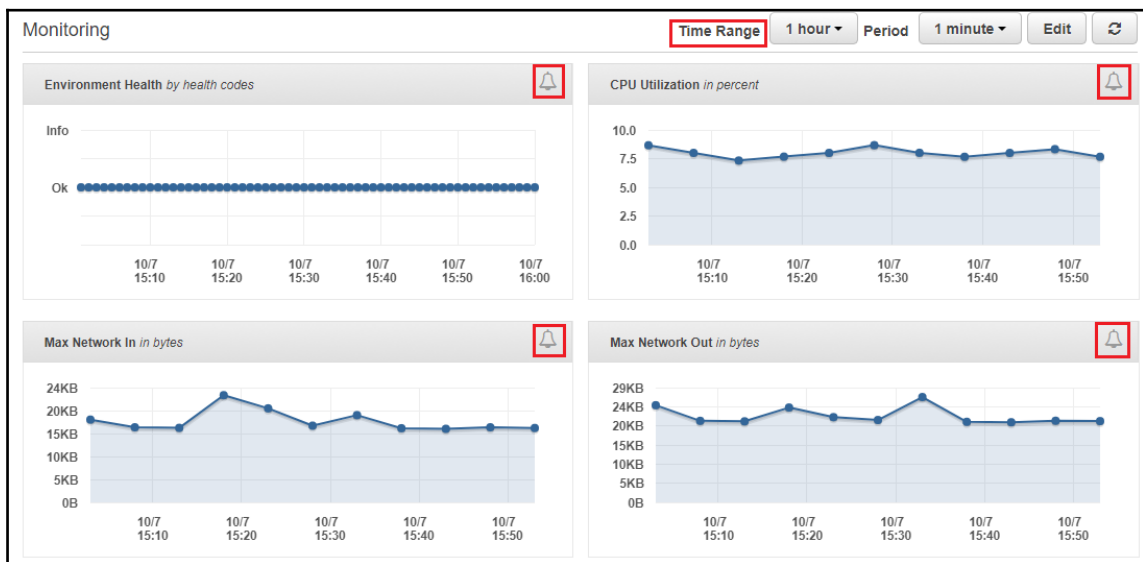


Logs Request Logs ▾ Refresh

Click Request Logs to retrieve the last 100 lines of logs or the entire set of logs from each EC2 instance. [Learn more](#)

Log file	Time	EC2 instance	Type
Download	2017-10-07 15:58:10 UTC+0200	i-0a7cdc1991ca0e96b	Full Logs

And last but not least, you can also leverage the **Monitoring** and **Alarms** sections to view the overall **Environment Health**, as well as other important metrics, such as **CPU Utilization**, **Max Network In**, and **Max Network Out**. To configure the alarms for individual graphs, all you need to do is select the alarm icon adjoining each of the graphs present in the **Monitoring** dashboard, as shown in the following screenshot:



A corresponding **Add Alarm** widget will pop up, using which you can configure the alarm's essentials, such as its **Name**, the **Period**, and **Threshold** settings, as well as the required **Notification** settings.

In this way, you can use the environment dashboard and the EB CLI together to perform daily application administration and monitoring tasks. In the next section, we will be leveraging this environment dashboard to clone and create a new production environment from the existing development environment.

Cloning environments

With the development environment all set and working, it is now time to go ahead and create a production environment. Now, technically, you could repeat all the processes that we followed earlier for the development environment creation, and that would work out well indeed, but Elastic Beanstalk offers a really simple and minimalistic approach to creating new environments while using an existing one as a template. The process is called cloning, and it can be performed in a few simple clicks, using the environment dashboard itself:

1. To get started, simply select the **Actions** tab from the environment dashboard page and select the option **Clone Environment**. This will bring up the **New Environment** page, as shown here:

New Environment

Environment name

Environment URL .us-east-1.elasticbeanstalk.com

URL is available.

Description

Maximum length of 200 characters.

Platform

Service role

2. Here, start off by providing an **Environment name** for the new environment, followed by a unique prefix for the **Environment URL**. Remember, this is a clone from the earlier development environment that we created, so, by default, it will contain the same Amazon Linux instance with the WordPress application that we pushed in during the Dev stages. This is not a concern as we can always use the EB CLI to push the production version of the application as well. But for now, fill in the rest of the details and select the **Clone** option.

The new environment undergoes the same initialization and creation process as it did earlier, creating separate security groups, assigning a new Elastic IP, and launching a new EC2 instance with the same application version that was pushed in the development environment.

Once completed, you should now have two very similar environments up and running side by side, but isn't a production environment supposed to be more than just one instance? Well, that's precisely what we will be configuring in the next section.

Configuring the production environment

Now that we have had a good tour of the environment dashboard, it should be relatively easy to configure the production environment as per our requirements. Let's start off by increasing the instance count for our production environment:

1. Select the **Scaling** configuration tile from the newly created production environment's configuration dashboard and change the **Environment type** from **Single instance** to **Load balancing, auto scaling**. The instance count settings as well as the auto scaling features, will only be available once the new changes are reflected in the environment. Click **Apply** once done.



To verify that the changes have indeed been propagated, you can copy the newly created Elastic Load Balancer DNS name into a web browser and verify that you can access the WordPress getting started wizard.

2. Next, you can also change the default instance type from **t1.micro** to something a bit more powerful, such as **t2.medium** or **t2.large**, using the **Instances** configuration section.
3. Once your major settings are done, you will also require a new RDS backed MySQL database for your production instances. So go ahead and create a new MySQL DB instance using the RDS Management Console at <https://console.aws.amazon.com/rds/>.



For handling production-grade workloads, I would strongly recommend enabling multi-AZ deployment for your MySQL database.

4. Remember to make a note of the database name, the database endpoint, as well as the username and password, before moving on to the next steps!
5. Next, using the production environment URL, launch your WordPress site and fill in the required database configuration details, as depicted in the following screenshot:

Below you should enter your database connection details. If you're not sure about these, contact your host.

Database Name	prod-wordpress	The name of the database you want to use with WordPress.
Username	yoyo	Your database username.
Password	p@\$w0rD	Your database password.
Database Host	wordpress-database.cluster-cq	You should be able to get this info from your web host, if localhost doesn't work.
Table Prefix	wp_	If you want to run multiple WordPress installations in a single database, change this.

This method of configuring database settings is not ideal, especially when it comes to a production environment. Alternatively, Elastic Beanstalk provides you with the concept of environment properties that enable you to pass key-value pairs of configurations directly to your application.

6. To do so, you need to select the **Configuration** section from your **Production** dashboard, and within that, opt to modify the **Software** configuration.
7. Here, under the **Environment Properties** section, fill out the required production database variables, as depicted in the following screenshot:

Property Name	Property Value
DB_HOST	wordpress-cluster.cluster-cqre34l ✕
DB_USER	yoyo ✕
DB_PASSWORD	p@\$w0rD ✕
DB_NAME	prod-wordpress +

But where do these variables actually end up getting configured? That's where we leverage the WordPress configuration file called `wp-config.php` and configure all these variables into it. Upon loading, PHP will read the values of each of these properties from the environment property that we just set in Elastic Beanstalk.

8. Open your `wp-config.php` file using your favorite text editor, and change the database section, as shown in the following snippet:

```
/** The name of the database for WordPress */
define('DB_NAME', getenv('DB_NAME'));

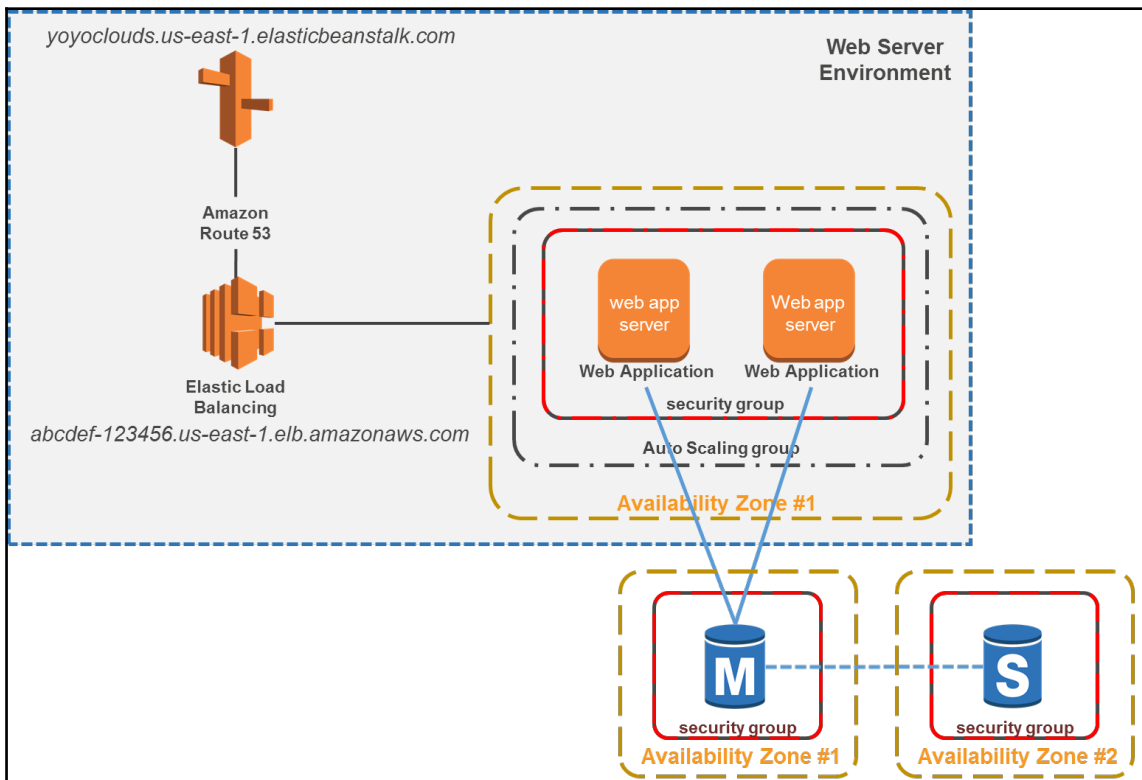
/** MySQL database username */
define('DB_USER', getenv('DB_USER'));

/** MySQL database password */
define('DB_PASSWORD', getenv('DB_PASSWORD'));

/** MySQL hostname */
define('DB_HOST', getenv('DB_HOST'));
```

9. Save the file and push the newly modified code into the production environment using the `eb deploy` command. Simple isn't it?

Here's what the new environment should look like after the deployments:



Looking good so far, right? With this done, your WordPress setup should be able to scale in and out efficiently without you having to worry about the load balancing needs or even about the MySQL instances. Additionally, now that we have configured the instances to fetch the database information from Elastic Beanstalk itself, we no longer have to worry about what will happen to our site if the underlying WordPress instances restart or terminate. This is exactly what we set out to do in the first place, but there's still a small catch. What about the content files that you will eventually upload on your WordPress website, such as images and videos? These uploads will end up getting stored on your instance's local disks, and that's a potential issue as you may end up losing all of your data if that instance gets terminated by the auto-scaling policies. Luckily for us, AWS has a solution to this problem, and that's exactly what we are going to learn about next.

Introducing Amazon Elastic File System

AWS, for one, has really put in a lot of innovation and effort to come up with some really awesome services, and one such service that I personally feel has tremendous potential is the Elastic File System. Why is it so important? Well, to answer this question, we need to take a small step back and understand what type of storage services AWS offers at the moment.

First up, we have the object stores in the form of Amazon S3 and Amazon Glacier. Although virtually infinite in scaling capacity, both these services are known to be a tad slower performance-wise compared to the EC2 instance storage and the EBS. This is bound to happen, as the likes of EBS is specially designed to provide fast and durable block storage, but, as a trade-off, you cannot extend an EBS volume across multiple Availability Zones. Elastic File System or EFS, on the other hand, provides a mix of both worlds by giving you the performance of an EBS volume combined with the availability of the same volume across multiple AZs, and that is really awesome! To summarize, EFS is a massively scalable file storage system that allows you to mount multiple EC2 instances to it simultaneously across AZs, without having to worry about the durability, availability, or performance of the system.

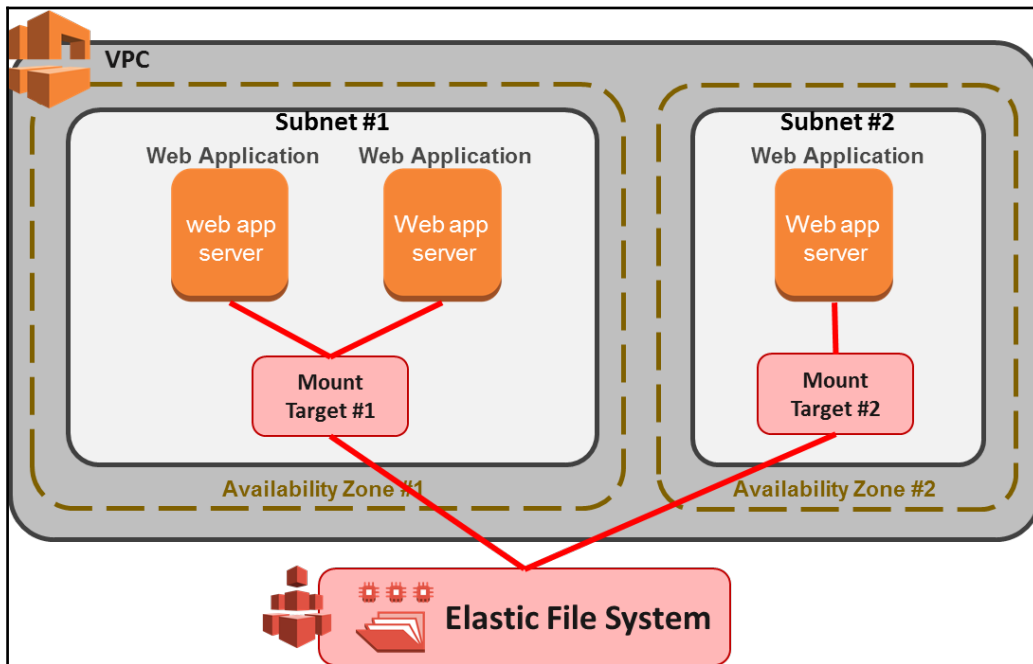
How does EFS actually work, you ask? Well, that's exactly what we will learn about in the next section.

How does it work?

EFS works in a very simple and minimalistic way, so as to reduce the amount of configurations that you need to perform and manage as an end user. To start off, EFS provides you with the ability to create one or more filesystems. Each filesystem can be mounted to an instance or instances, and data can be read as well as written to them. Mounting the filesystem requires your instances to have support for the Network File System version 4.0 and 4.1 (NFSv4) protocol. Most Linux operating systems come with the necessary support, however, you may have to install the NFS client on these machines if it is not there to connect to an EFS. So, how is this useful for our WordPress application? Well, for starters, once you have an Amazon Elastic File System in place, you can have multiple EC2 instances connect to it simultaneously and use it as a scalable shared drive that can extend even to petabytes if the need arises. Also, the Amazon Elastic File System does not have any downtime or repercussions if your EC2 instances reboot or even terminate; the data will persist on the filesystem until you manually delete it or terminate the filesystem itself. There are some rules and limitations, however, when it comes to using the Elastic File System, which you ought to keep in mind.

You can mount an Amazon EFS on instances in only one VPC at a time, and both the filesystem and the VPC must be in the same AWS region.

Once the filesystem is created, you will be provided with a DNS name for identifying it within your region. Additionally, you will also be required to create one or more supporting mount targets within your VPC, which basically acts as a connectivity medium between your instances present within a subnet and the filesystem. Here is a representational diagram of how an Elastic File System interacts with EC2 instances using mount targets:



As an administrator, you can create one mount target in each Availability Zone present in a given region. You can also create a mount target in each of the subnets presents within a particular VPC, so that all EC2 instances in that VPC share that mount target. In the next section, we will be exploring a few simple steps required for setting up your own Elastic File System.

Creating an Elastic File System

Setting up your own Elastic File System is as easy as it gets! You can start off by launching the Elastic File System dashboard from the AWS Management Console, or alternatively, by visiting the URL <https://console.aws.amazon.com/efs/>:

1. On the EFS landing page, select the option **Create file system** to get started.
2. In the **Configure file system access** page, you can start off by first selecting the **VPC** you want to associate the filesystem with. Remember, you can have multiple filesystems per VPC, however, they cannot be extended across regions:

Configure file system access

An Amazon EFS file system is accessed by EC2 instances running inside one of your VPCs. Instances connect to a file system by using a network interface called a mount target. Each mount target has an IP address, which we assign automatically or you can specify.

VPC vpc-da9695a3 - yoyode... ⓘ

Create mount targets

	Availability Zone	Subnet ⓘ	IP address ⓘ	Security groups ⓘ
☐	us-east-1a			
☒	us-east-1b	subnet-9850b0d3 - yoyodev-public-01	Automatic ✎	sg-1daa486e - default ✕
☒	us-east-1c	subnet-48848c12 - yoyodev-public-02	Automatic ✎	sg-1daa486e - default ✕
☒	us-east-1d	subnet-f4932e90 - yoyodev-public-03	Automatic ✎	sg-1daa486e - default ✕

3. With the VPC selected, the associated subnets will automatically populate themselves based on the Availability Zones that they are a part of in the **Create mount targets** section. Here, you can select the appropriate subset that you wish to associate with the Elastic File System, along with its corresponding security group. In my case, I've selected the individual public subnets from my VPC, as the WordPress application instances will be deployed here, and these instances will require access to the filesystem for storing the images and other content.
4. With the fields populated, select the **Next Step** option to proceed.

5. The next step is all about **Configuring optional settings** for your Elastic File System. Here, you can **Add tags** to describe your filesystem and select the appropriate **Performance mode** for the filesystem, based on your requirements. Today, EFS provides two modes: the **General Purpose**, which is ideal for running the majority of workloads, and a **Max I/O** mode, which is specifically designed for when your environment needs to scale to tens of thousands of EC2 instances, all connecting to this single filesystem itself. **Max I/O** mode provides much better performances compared to **General Purpose**, however, there is the chance of a slightly higher latency when handling file operations here.
6. The final option left is **Enable encryption**, which, if checked, will leverage a KMS key from your existing AWS account and encrypt all the data stored in the filesystem at rest:

The screenshot shows the 'Add tags' section of the AWS EFS configuration page. It features a table with two columns: 'Key' and 'Value'. Two tags are added: 'Name' with value 'yoyo-prod-wordpress' and 'Environment' with value 'Production'. Below the tags, the 'Choose performance mode' section is visible, with 'General Purpose (default)' selected. The 'Enable encryption' section is also shown, with the 'Enable encryption' checkbox unchecked. At the bottom right, the 'Next Step' button is highlighted.

Key	Value	Remove
Name	yoyo-prod-wordpress	✕
Environment	Production	✕

Choose performance mode

We recommend **General Purpose** performance mode for most file systems. **Max I/O** performance mode is optimized for applications where tens, hundreds, or thousands of EC2 instances are accessing the file system — it scales to higher levels of aggregate throughput and operations per second with a tradeoff of slightly higher latencies for file operations.

☒ **General Purpose (default)**

☐ Max I/O

Enable encryption

If you enable encryption for your file system, all data on your file system will be encrypted at rest. You can select a KMS key from your account to protect your file system, or you can provide the ARN of a key from a different account. Encryption can only be enabled during file system creation. [Learn more](#)

☐ Enable encryption

Cancel Previous **Next Step**

7. Complete the EFS setup process by reviewing the configuration changes on the **Review and create** page, and finally, click on **Create** to enable the filesystem. This process takes a couple of minutes, but once completed, you will be shown the **DNS name** of your newly created filesystem. Make a note of this name as you will be required to reference it in our Elastic Beanstalk environment as well.

So far, so good! We have our production environment up and running on Elastic Beanstalk, and now we have created a simple yet powerful Elastic File System. In the next section, we will look at how you can integrate the two services for use by WordPress using Elastic Beanstalk's configuration files concept.

Extending EFS to Elastic Beanstalk

Although Elastic Beanstalk takes complete care of your environment's provisioning and configuration, there are still methods which you can use to control the advanced configuration of your environment, such as integrating your application with the likes of other AWS services, such as ElastiCache, or even EFS for that matter. This can be performed using a variety of services provided by Beanstalk itself; for example, by leveraging Beanstalk's **Saved configurations**, or even using **Environment Manifest** (YML) files. But in this particular section, we will be concentrating on integrating the EFS service with our WordPress application using specialized **Configuration Files** called `.ebextensions`.

These `.ebextensions` are simple YAML formatted documents ending with a `.config` file extension. Once the `.ebextensions` file is created, you need to place this in the root folder of your application's source code, within a special directory named `.ebextensions`, and finally, deploy your application over to Beanstalk.

These configuration files are so powerful that you don't even have to connect to your instances through SSH to issue configuration commands. You can configure your environment entirely from your project source by using `.ebextensions`:

1. To start using `.ebextensions` for your WordPress setup, first we need to create a folder named `.ebextensions` within the root of your WordPress application. Type the following command from your Dev instance:

```
# cd wordpress && sudo mkdir .ebextensions
```

2. Create a new file with an extension of `.config`, and paste the following contents into it:

```
# sudo vi efs.config

### PASTE THE FOLLOWING CONTENTS ###
packages:
  yum:
    nfs-utils: []
    jq: []
files:
```

```

"/tmp/mount-efs.sh" :
  mode: "000755"
  content: |
    #!/usr/bin/env bash
    mkdir -p /mnt/efs
    EFS_NAME=$(/opt/elasticbeanstalk/bin/get-config environment |
jq -r '.EFS_NAME')
    mount -t nfs4 -o
nfsvers=4.1,rsz=1048576,wsz=1048576,hard,timeo=600,retrans=2
$EFS_NAME:/ /mnt/efs || true
    mkdir -p /mnt/efs/uploads
    chown webapp:webapp /mnt/efs/uploads
  commands:
    01-mount:
      command: "/tmp/mount-efs.sh"
  container_commands:
    01-rm-wp-content-uploads:
      command: rm -rf /var/app/ondeck/wp-content/uploads
    02-symlink-uploads:
      command: ln -snf /mnt/efs/uploads /var/app/ondeck/wp-
content/uploads

```



You can find the complete copy of the previous code at <https://github.com/yoyoclouds/Administering-AWS-Volume2>.

This file causes Elastic Beanstalk to mount the newly created EFS volume on the instance's `/mnt/efs` directory, and also removes the `wp-content/uploads` directory if it exists, and symlinks it to `/mnt/efs/uploads` so that it persists and is shared between instances:

1. Once the file is created, use the `eb deploy` command once again to push the application directory and the newly added `.ebextensions` directory to your production environment.
2. Last but not least, sign in to your production environment and select the **Configuration** option from the *environment* dashboard. Here, select the **Software** configuration tile and add the following key-value pair into the **Environment Properties** section, as shown:

Property Name	Property Value
DB_HOST	wordpress-cluster.cluster-cqre34l ✕
DB_USER	yoyo ✕
DB_PASSWORD	p@\$w0rD ✕
DB_NAME	prod-wordpress ✕
EFS_NAME	fs-c260278.efs.us-east-1.amazon +

Cancel
Apply

- Here, the **EFS_NAME** has to have the newly created EFS filesystem's DNS name as its value. This is the same DNS name that we copied a while back once the EFS was created.

Once the deployment changes states and is made available, select the environment URL and verify whether the WordPress configurations are all working as intended or not. If you have made it this far, then you should have a really awesome, highly available, scalable, WordPress site up and running! Awesome, isn't it?

Planning your next steps

Well, we have covered a lot of new features and services in this chapter, however, there are still a few things that I would recommend you try out on your own. First up is Elastic Beanstalk's advanced configurations.

As mentioned earlier, Beanstalk provides a lot of different ways for you to customize and extend your application with other AWS services using a variety of built-in services such as `.ebextensions`, which we covered in the previous section. One similar service that can be used to configure a Beanstalk environment's configuration is called the **environment manifest** file. This is a simple YAML file containing your environment's manifest configurations, such as the environment name, solution stack, and environment links to use when creating your environment. The file is placed in your application's root directory and is generally named `env.yaml`. One of the key uses of this file is to provide support for environment links that enable you to connect two application environments using simple names as references.

For example, if you have a website as a front-ending application that accepts certain inputs from the users, and another application that processes these inputs, you can create a link between the worker and the frontend application using this `env.yml` file. On invocation, the link between these two environments is set up and managed automatically by Beanstalk.

Here's a small snippet of the `env.yml` file's contents:

```
AWSConfigurationTemplateVersion: 1.1.0.0
SolutionStack: 64bit Amazon Linux 2015.09
EnvironmentName: frontend-environment
EnvironmentLinks:
  "WORKERQUEUE" : "worker-environment"
```

You can learn more about *Environment Manifest* (`env.yml`) at <https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/environment-cfg-manifest.html>.

Alternatively, Beanstalk also provides you with an easier configuration saving mechanism, which you can invoke using either the environment dashboard or the EB CLI. This is called a **Saved Configuration** and can be enabled by selecting the **Save configuration** option under the **Actions** tab in your environment dashboard. Once applied, the environment configurations accept any custom configurations that are stored in an S3 bucket as an object. You can even download this configuration object and create clones of your environment using the EB CLI. To learn more about saved configurations, check out this URL: <http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/environment-configuration-savedconfig.html>.

Another very interesting thing worth exploring is the support for Docker containers provided in Elastic Beanstalk! As you may already be aware, Docker containers are the next big thing when it comes to creating microservices-backed applications that can be deployed and scaled at tremendous scale. The Docker platform for Elastic Beanstalk has two generic configurations, a single container and multi-container option, and also provides several preconfigured container images to choose from.

From an Elastic File System perspective, one key aspect that is worth reading and exploring is the filesystem's overall performance considerations. This documentation especially highlights the different performance levels and use cases compared to EBS-provisioned IOPS volumes. The document also provides some keen insights and considerations for how to maximize the filesystem's performance. You can check out the documentation at <http://docs.aws.amazon.com/efs/latest/ug/performance.html>.

Summary

So, here we are. Yet another chapter comes to an end! But, before we move on to the next chapter, here's a quick round up of the things we have learned so far.

We started off with a quick introduction to Amazon Elastic Beanstalk, followed by a dive into its concepts and terminologies. Then we created a simple development environment for our WordPress application using the Elastic Beanstalk management console and the EB CLI. Along the way, we also learnt how to deploy the application to a specific environment using the EB CLI, and finally, learned how to quickly clone an environment and configure it for handling production workloads. Last but not least, we explored and learned how to leverage Elastic File System to create a durable and scalable file sharing system to be used by our WordPress setup, and concluded the chapter with some key insights and next steps.

In the next chapter, we will be starting off by exploring some security services in the form of WAF and Shield. So, stay tuned—we still have a lot to learn!

4

Securing Workloads Using AWS WAF

In the previous chapter, we learned a lot about how to leverage Amazon Elastic Beanstalk as well as Amazon Elastic File System to build and deploy highly scalable and available applications with the utmost of ease! However, there is one critical aspect that we didn't talk too much about in the previous chapter, and that is, of course, security! *"How do I safeguard my applications and workloads against malicious software and threats?"* This is exactly the question we will try and answer through a combination of two simple, yet very powerful, AWS services, namely AWS Shield and AWS **Web Application Firewall**, or **WAF**.

Keeping this in mind, let's have a quick look at the various topics that we will be covering in this chapter:

- Introducing AWS WAF and how it works
- Securing our WordPress site by leveraging WAF and using web ACLs
- Learning about additional WAF conditions for protection against cross-site scripting and SQL injections
- Automated deployment and configuration of AWS WAF using CloudFormation templates
- Monitoring AWS WAF using Amazon CloudWatch
- A brief introduction to AWS Shield and how it works
- Understanding AWS Shield Advanced and how to leverage it

There is so much to do, so let's get started right away!

Introducing AWS Web Application Firewall

Security has always been, and always will be, a key concern for a lot of organizations that run their workloads and applications on the cloud. That is precisely why AWS offers a wide assortment of managed services that you, as a cloud administrator, should leverage in order to protect and safeguard your workloads from any compromises or threats. In this section, we are going to explore one such simple, yet really powerful, service, called AWS WAF, or Web Application Firewall.

AWS WAF is basically a firewall that helps you to protect your internet-facing applications from common web-based threats and exploits. It is basically a service that enables you to specify a set of web security rules or ACLs that can allow or restrict a certain type of web traffic across Amazon CloudFront as well as the **Application Load Balancer (ALB)**. As of now, WAF can be used to create customized rules that can safeguard your applications against attacks, such as SQL injections, cross-site scripting, **Distributed Denial of Services (DDoS)**, bad bots, scrapers, and much more! You can easily create new rules and attach them to your existing ACL list as per your requirements, enabling you to respond to and mitigate changing traffic patterns more rapidly.

WAF also comes equipped with a powerful API, by using which you can automate the deployments of ACL rules as well as manage them programmatically. Alternatively, for the UI people out there, WAF provides customization CloudFormation templates which will allow you to get started with a complete WAF-based security solution in less than a few minutes! We will be looking at how to deploy this template for securing our own WordPress application as well a bit later in this chapter.



WAF is priced based on the number of ACL rules which you deploy, as well as on the number of web requests that your application receives.

Here is a quick summary of benefits that you can obtain by leveraging AWS WAF:

- **Enhanced protection:** Apart from your standard VPC and security groups, you can additionally safeguard your applications against commonly occurring web attacks by leveraging WAF's ACL rules.
- **Advanced traffic filtering:** Unlike your simple NACLs or security groups, WAFs provide you with an ability to define custom rules and conditions based on the characteristics of your incoming web request, such as values present in the headers, origin IP address of the request, whether the request has any SQL code present in it, and so on. Using these conditions, you now have the ability to basically allow, block, or filter traffic based on such preset conditions.

- **Easy management:** With WAF rules defined and managed in one central location, you can easily reuse and propagate your custom ACLs across multiple CloudFront CDNs as well as Application Load Balancers, and monitor the traffic as well as mitigate any issues, all using the same WAF API or web user interface.
- **Cost effective security solution:** One of the best parts of leveraging WAF is that there are absolutely no upfront fees or costs associated with it. You simply pay based on the number of rules you create using WAF as well as the amount of traffic your web application receives, and not a penny more!

With this basic set of information, let's have a look at how WAF actually works!

Concepts and terminologies

As discussed briefly, WAF can be enabled over your standard ALBs and over your CloudFront distributions. But before we get started with configuring WAF and its various rules and ACLs, we first need to understand some of its commonly used terms and terminologies:

- **Conditions:** Conditions form the core of your WAF rulesets. These are basically configurable characteristics that you want WAF to monitor in each of your incoming web requests. At the time of writing this book, WAF supports the following list of conditions:
 - **IP match:** You can use this condition to check whether the incoming web request originated from a specified black/whitelisted IP addresses or not. You can then plot corresponding actions to be performed against the same based on your requirements, such as not allowing any incoming traffic other than the whitelisted IP range, and so on. AWS WAF supports /8, /16, /24, /32 CIDR blocks for an IPv4 address.
 - **String and regex match:** A string match or a regex match condition can be used to specify a part of an incoming web request and its corresponding text that you wish to control access to. For example, you can create a match or regex condition that checks the user agent headers and its value against a preset *string* or *expression*. If the condition matches, you can opt to either allow or block that particular traffic using WAF rules.

- **SQL injection match:** You can use this condition to inspect certain parts of your incoming web requests, such as the URI or query string, for any malicious SQL code. If a pattern matches, you can then opt to block all traffic originating from that particular request's IP range.
- **Cross-site scripting match:** Hackers and exploiters can often embed malicious scripts within web requests that can potentially harm your application. You can leverage the cross-site scripting match condition to inspect your incoming request URI or headers for any such scripts or code, and then opt to block the same using WAF rules.
- **Geographic match:** You can use this condition to list countries that your web request originated from and accordingly block or allow the same based on your requirements.
- **Size constraint match:** You can use the size constraint match condition to check the lengths of specified parts of your incoming web requests, such as the query string or the URI. For example, you can create a simple WAF rule to block all requests which have a query string greater than 100 bytes, and so on.
- **Rules:** With your conditions defined, the next important aspect of configuring WAF are the rules. Rules basically allow you to combine one or more conditions into a logical statement, which can then be used to either allow, block, or count a particular incoming request. Rules are further classified into two categories:
 - **Regular rules:** Regular or standard rules, apply one or more conditions to your most recent batch of incoming web requests. For example, a rule to block all incoming traffic from the IP range `40.40.5.0/24` or if there is any SQL-like code in the query string of your request, and so on.
 - **Rate-based rules:** Rate-based rules are very much like your regular rules apart from one addition: the rate limit. You can now configure conditions and pass a rate limit along them as well. The rule will only trigger if the conditions match or exceed that particular rate limit which was set.

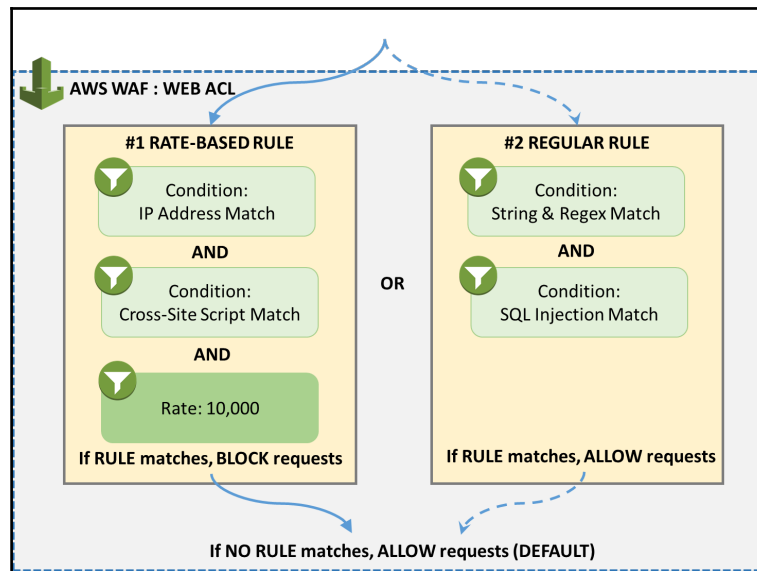


Rate limits are checked by WAF within a 5-minute window period.

For example, you may configure a simple condition that blocks all incoming traffic from the IP range `40.40.5.0/24` with a rate limit of 10,000. In this case, the rule will only trigger a corresponding action (allow, block, count) if the condition is met and the number of incoming requests in a 5-minute period exceed 10,000 requests. Requests that do not meet both the conditions are simply not compared towards the rate limit and hence will not be blocked by this rule.

- **Web ACLs:** Once the rules are defined, you combine them into one or more web ACLs. Here, you have the ability to define an action for your rule if it gets triggered; for example, allow, block, count, or even perform a default action that gets triggered in case a request doesn't match any of the conditions or rules specified. Web ACLs work on a priority basis, so the rule listed first is the one that gets compared to the incoming request first. This makes it extremely important to know the order in which you create and assign your rules in a web ACL.

Here is a simple representation of how conditions, rules, and web ACLs work together in WAF:

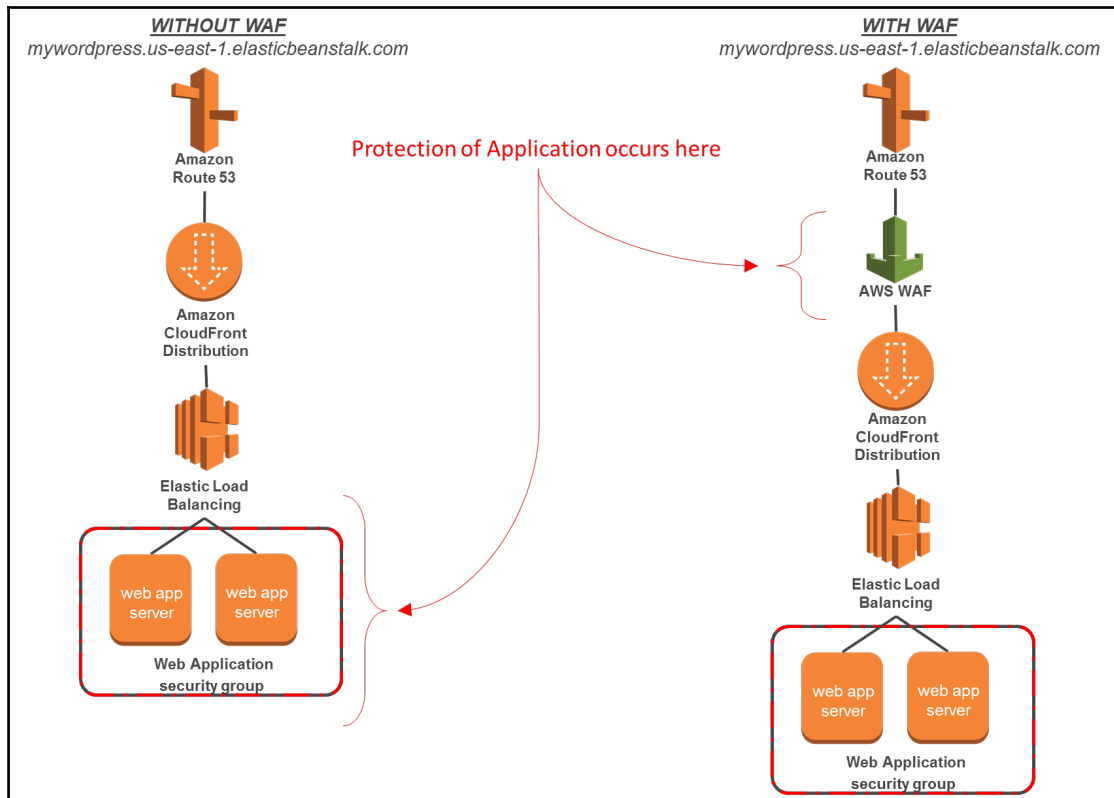


With the concepts out of the way, let's look at a few simple steps that allow you to set up and configure WAF Web ACLs for safeguarding your web applications.

Getting started with WAF

In this section, we are going to look at a few simple and easy-to-follow steps for getting started with AWS WAF. For demonstration purposes, we will be leveraging the same environments and application that we deployed from our previous chapter here, so, if you haven't gone through the use case, this might be a good time for a quick revisit!

In the previous chapter, we leveraged Elastic Beanstalk as well as Elastic File System services to deploy a scalable and highly available WordPress application over the internet. In this section, we will leverage the same setup and secure it even further by introducing AWS WAF into it. Why use WAF for our WordPress application? Well, the simplest answer is to completely abstract the security checks from the underlying web server instance(s), and instead place the security checks at the point of entry of our application, as depicted in the following diagram:



To get started, you will first need to ensure that your WordPress application has a CloudFront CDN attached to it, or alternatively an Application Load Balancer frontend its requests. This is a crucial step, as without a CloudFront CDN or an Application Load Balancer, WAF will simply not work! In my case, I have configured and deployed a simple CloudFront CDN for my production-grade WordPress application. You can refer to the following step-by-step guide for setting up your own CDN using CloudFront, at <http://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/>.

Creating the web ACL

Once you are done with your CDN, head over to the AWS Management Console and filter out WAF and Shield services using the dashboard, or alternatively, navigate to this URL <https://console.aws.amazon.com/waf/home> to bring up the WAF dashboard:

1. Assuming that this is the first time you are configuring WAF, you will be prompted by a welcome screen to either opt for AWS WAF or AWS Shield services. Select the **Go to AWS WAF** option. This will redirect you to the WAF dashboard, where we select the **Configure web ACL** option to get started.
2. Selecting the **Configure web ACL** option will bring up a **Set up a web access control list (web ACL)** wizard that will guide you through your first web ACL setup.
3. The first page on the wizard basically covers the concepts of conditions, rules, and ACLs, so simply select the **Next** option to proceed further.
4. In the **Name web ACL** page, provide a suitable **Web ACL Name** for your new ACL. You will notice that the **CloudWatch metric name** field gets correspondingly auto-populated with a matching name. You can change the name as per your requirements. This metric name will be later used to monitor our web ACLs using CloudWatch's dashboards.
5. Moving on, from the **Region** drop-down list, select either **Global (CloudFront)** or an alternative **Region** name, based on whether you want to secure a CDN or an Application Load Balancer. In my case, since I already have a CDN set up, I've opted for the **Global (CloudFront)** option.



WAF for the Application Load Balancer is currently supported only for the following regions: US East (N. Virginia), US West (N. California), US West (Oregon), EU (Ireland), and Asia Pacific (Tokyo).

6. In the **AWS resource to associate** field, you can opt to select your CloudFront distribution or your Application Load Balancer using the drop-down list; however, for the sake of simplicity, do not configure this option for the time being. Remember, you can always associate your web ACLs with one or more AWS resources after completing this wizard! Once done, click **Next** to proceed:

Name web ACL ?

Web ACL name*

CloudWatch metric name*

The metric name can contain only alphabetic characters.

Region*

Use global to create WAF resources that you would associate with CloudFront distributions and other regions for WAF resources that you would associate with ALBs in that region.

AWS resource to associate

You can associate this web ACL with more resources after you finish the wizard. On the Web ACLs page for this web ACL, see the Rules tab.

* Required

[Cancel](#) [Previous](#) [Next](#)

7. With the web ACL named, we move on to the next section where we can configure our conditions. On the **Create conditions** page, select an appropriate condition that you wish to configure for your web application. In this scenario, we will be configuring an IP match condition along with a string match condition. The idea here is to only grant access to our WordPress administrator login page (`wp-login.php`) from my local laptop's IP, and, conversely, for any other IP that wishes to access the `wp-login.php` page, the traffic should get dropped.

Creating the conditions

As mentioned earlier, conditions are configurable characteristics that you want WAF to monitor in each of your incoming web requests:

1. To get started with a condition, select the **Create condition** option from the **IP match conditions** tile.
2. Here, provide a suitable **Name** for your match condition and select the **IPv4** option from the **IP Version**. Provide your desktop's or laptop's public IP in the **Address** field. You can alternatively provide a range of IP addresses here using either of the supported CIDR blocks.
3. Remember to select the **Add IP address or range** option before creating the match condition:

IP addresses

Add one or more IP addresses or IP address ranges by using CIDR notation.

IP Version* ☒ IPv4 ☐ IPv6

Address* 80.85.186.200/32

Add IP address or range

Filters in IP match condition

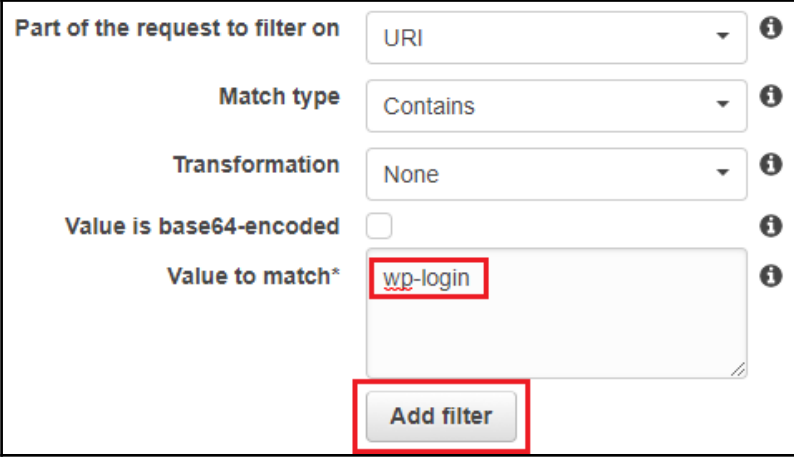
IP address of the request to filter on

This condition has no filters.

* Required Cancel Create

4. With the IP match condition created, let's move on to creating the second condition for our ACL as well. For this, select the **Create condition** option from the **String and regex match conditions** section.

5. Once again, we start by providing a suitable **Name** for our string match condition, followed by selecting the **Type** of string to match with. Here, select the **String match** option to begin with.
6. Next, in the **Part of the request to filter on** section, select the appropriate section of your request that you wish to filter, using the match condition. In my case, I have selected the **URI** option as we need to match the resource `wp-login.php` from the URI. Alternatively, you can also opt to select the following values based on your requirements:
 - **Header**: Used to match a specific request header, such as user-agent.
 - **HTTPMethod**: Used to indicate the type of operation the request intends to perform on the origin, such as `PUT`, `GET`, `DELETE`, and so on.
 - **QueryString**: Used to define a query string in a URL.
 - **Body**: Used to match the body of the request. In this case, WAF only inspects the first 8,192 bytes (8 KB) contained within the request's body. You can alternatively set up a **Size Constraint** condition that blocks all requests that are greater than 8 KB in size.
7. Next, in the **Match type** drop-down list, select the option **Contains**, as shown in the following screenshot. The **Contains** option means that the string to match can appear anywhere in the request. Alternatively, you can also opt to select from these options, based on your requirement:
 - **ContainsWord**: Used to specify a specific **Value to match** in the request
 - **Exactly matches**: Used to match the string and the request value exactly
 - **Starts with**: Used to check for a matching string at the beginning of a request
 - **Ends with**: Used to check for a matching string at the end of the request



Part of the request to filter on: URI

Match type: Contains

Transformation: None

Value is base64-encoded: ☐

Value to match*: wp-login

Add filter

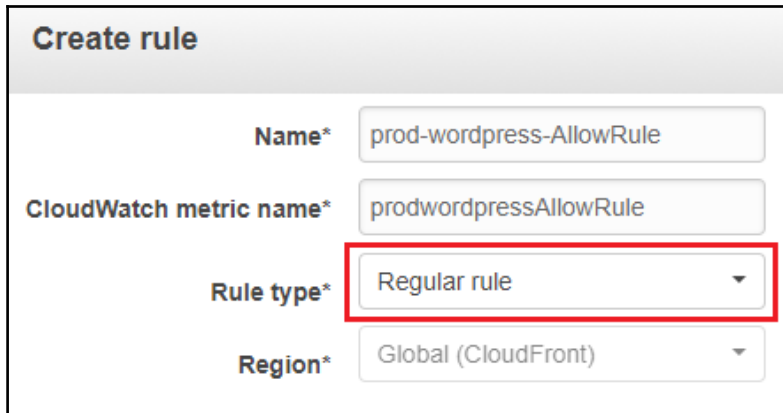
8. The **Transformation** field is handy when you need to re-format the web request before WAF inspects the same. This can involve **Converting to lowercase**, **HTML decoding**, **Whitespace normalization**, **URL Decode**, and so on. For this particular use case, we don't have any particular transformation to perform on the request, and hence I've selected the **None** option.
9. Finally, in the **Value to match** field, enter the text (`wp-login`) that we want WAF to search for in the web requests. Once completed, remember to click on the **Add filter** option before you proceed with the `Create` command.
10. With this step completed, our basic conditions are in place. Alternatively, you can set up other relevant conditions based on your criteria and requirements. Once done, select the **Next** option to proceed with the wizard.

Creating rules

With your conditions defined, we now move on to the next important aspect of configuring WAF: rules. Rules basically allow you to combine one or more condition, into a logical statement, which can then be used to either allow, block, or count a particular incoming request:

1. In the **Create rules** page, you can now merge the conditions we created a while back and assign each rule a corresponding action, such as allow, block, or count. To get started, select the **Create rule** option.

2. In the **Create rule** popup, we will be creating two rules: one rule that will basically allow me to access the WordPress admin login page (`wp-login.php`) from my local laptop, and another rule that blocks traffic to the same login page. Let's first create the **Allow traffic** rule.
3. To do so, type in a suitable **Name** for your rule. You will notice the corresponding **CloudWatch metric name** field auto-populate itself with the same name as well. You can choose to change this name as per your requirements, or leave it to its default value.
4. Next, in the **Rule type** drop-down list, select whether you want this rule to be a **Regular rule** or a **Rated rule**. For this scenario, I've opted for the **Regular rule**, as shown in the following screenshot:



The screenshot shows the 'Create rule' form in the AWS WAF console. It contains the following fields and values:

- Name***: prod-wordpress-AllowRule
- CloudWatch metric name***: prodwordpressAllowRule
- Rule type***: Regular rule (highlighted with a red box)
- Region***: Global (CloudFront)

5. Once done, move on to the **Add conditions** section, where we can associate our rule with one or more conditions. Start by selecting the appropriate drop-down option to form the following rule:

```
When a request: "Does": "Originate from an IP Address in":  
"<SELECT_YOUR_IP_ADDRESS_MATCH_CONDITION_HERE>"
```

Here's what your new rule should look like once it is properly set up. Click on **Create** once completed:

Add conditions

When a request ✕

does ▾

originate from an IP address in ▾

prod-wordpress-IPAddress ▾

IP Addresses in prod-wordpress-IPAddress

80.85.186.200/32

Add condition

* Required Cancel Create

6. With your **Allow rule** created, we use the same steps once again to create a **Block rule** as well. Select the **Create rule** option once again, and provide a suitable **Name** for your rule. Similar to the previous case, I've opted for a **Regular rule** here as well.
7. Next, in the **Add conditions** section, we first add a condition that matches the following statement:

```
When a request: "Does not": "Originate from an IP Address in":
"<SELECT_YOUR_IP_ADDRESS_MATCH_CONDITION_HERE>"
```

8. Next, select the **Add condition** option to add the string match condition as well:

```
When a request: "Does": "Match at least one of the filters in the
string match condition":
"<SELECT_YOUR_STRING_MATCH_CONDITION_HERE>"
```

Here's what your rule should look like once both the conditions are added to it:

Add conditions

When a request ✕

does not ▾

originate from an IP address in ▾

prod-wordpress-IPAddress ▾

IP Addresses in prod-wordpress-IPAddress

80.85.186.200/32

And

When a request ✕

does ▾

match at least one of the filters in the string match condition ▾

prod-wordpress-StringMatch ▾

Filters in prod-wordpress-StringMatch

URI contains: "wp-login".

Add condition

* Required Cancel **Create**

9. With the conditions in place, select the **Create** option to finally create your blocking rule.

10. Now that your two rules are created, you should see them both listed in the **Add rules to a web ACL** page, as shown in the following screenshot:

Add rules to a web ACL

Rules: prod-wordpress-BlockRule ▾ Add another rule Create rule

If a request matches all of the conditions in a rule, take the corresponding action

Order	Rule	Action
▼ 1	prod-wordpress-BlockRule	☐ Allow ☒ Block ☐ Count ✕
▲ 2	prod-wordpress-AllowRule	☒ Allow ☐ Block ☐ Count ✕

If a request doesn't match any rules, take the default action

Default action* ☒ Allow all requests that don't match any rules ☐ Block all requests that don't match any rules

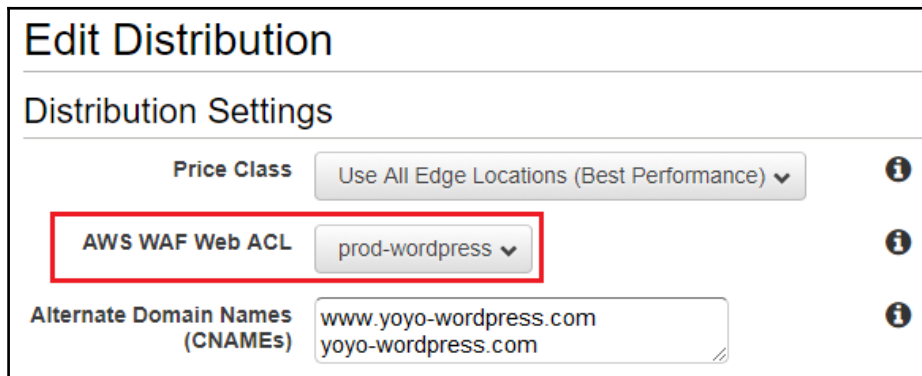
* Required Cancel Previous Review and create

11. Here, make sure you order your rules correctly, based on their precedence, by selecting the **Order** option as required. You can additionally configure the **Default action** for your web ACL as well. This default action will only get triggered if the request does not match any of the conditions mentioned in either the *allow* or the *blocking* rules. Once you are confident with your configurations, select the **Review and create** option, as shown earlier. And voila! Your basic WAF is now up and running!

Assigning a WAF Web ACL to CloudFront distributions

With the web ACL created, you can now easily assign it to one or more CloudFront distributions, as per your requirements. To do so, simply log in to your AWS dashboard and filter the **CloudFront** service, or alternatively, navigate to <https://console.aws.amazon.com/cloudfront/home> to view the CloudFront dashboard directly:

1. Once logged into the CloudFront dashboard, select the appropriate **Distribution ID** for which you wish to enable the **WAF Web ACL** rules.
2. Select the **Edit** option from the **General** tab to bring up your distribution's configurations and settings.
3. Here, in the **Edit Distribution** page, select your newly created web ACL from the **AWS WAF Web ACL** drop-down list, as shown in the following screenshot:



The screenshot shows the 'Edit Distribution' page in the AWS CloudFront console. The 'Distribution Settings' section is visible. The 'Price Class' is set to 'Use All Edge Locations (Best Performance)'. The 'AWS WAF Web ACL' dropdown menu is highlighted with a red box, and 'prod-wordpress' is selected. Below this, the 'Alternate Domain Names (CNAMEs)' field contains 'www.yoyo-wordpress.com' and 'yoyo-wordpress.com'.

4. Once the ACL is selected, I would also recommend that you enable the logging of your distribution in case you already haven't done that. This is just an added measure of precaution and security that is a must for any production-grade environment that you may be working on. Scroll down on the **Edit Distribution** page, and select the **On** option adjoining the **Logging** field. Provide your logging bucket's name in the **Bucket for Logs** field and click on the **Yes, Edit** option once the required fields are all filled in.

The changes will take a good few minutes to propagate through the CloudFront distribution. You can then move on to testing your WAF once the distribution's **Status** has changed to **Enabled**.

To test your WAF, simply open a browser and type in the URL of your WordPress application (`<http://YOUR_CLOUDFRONT_URL>/wp-login.php`) from your own laptop/desktop. In this case, you should be able to see the `wp-login.php` page without any issues whatsoever. However, if you try accessing the same page from a different laptop or machine, you will be thrown the following error on screen:



At this point, your WordPress administrator login page is now protected from all IPs except those that you specified in your Web ACL's allow list! Amazing, isn't it?



You can create a custom error page using the CloudFront distribution settings and redirect your users to this page rather than showing them the standard *error page*, as depicted in the preceding screenshot.

With this, we come towards the end of this basic web ACL configuration section. In the next section, we will be looking at how to enhance your basic ACL setup with more conditions, with more emphasis towards SQL injections and cross-site scripting.

Working with SQL injection and cross-site scripting conditions

Besides restricting access to a specific set of IP addresses, WAF additionally provides defense capabilities against more exploitative attacks, such as SQL injections and cross-site scripting. In this section, we will take a closer look at both of these conditions and how you can leverage them for protecting your own applications.

To start off, let's have a closer look at SQL injections. An SQL injection basically consists of the insertion of an SQL query within a request that is made from a client to your application.

SQL injections, if successfully implemented, can read as well as modify sensitive data from the database, and are even capable enough to execute administration operations on your database, such as restoring from a previous backed up file, shutting the database down completely, and much more.

Here's a list of some common conditions and their associated configurations that you can choose to apply in your web ACL rules:

HTTP request component to match	Relevant input transformations to apply	Justification
QUERY_STRING	URL_DECODE, HTML_ENTITY_DECODE	The most common component to match. Query string parameters are frequently used in database lookups.
URI	URL_DECODE, HTML_ENTITY_DECODE	If your application is using friendly or clean URLs, then parameters might appear as part of the URL path segment, and not the query string.
BODY	URL_DECODE, HTML_ENTITY_DECODE	A common component to match if your application accepts form input. AWS WAF only evaluates the first 8 KB of the body content.
HEADER: Cookie	URL_DECODE, HTML_ENTITY	A less common component to match. But, if your application uses cookie-based parameters in database lookups, consider matching on this component as well.
HEADER: Authorization	URL_DECODE, HTML_ENTITY_DECODE _DECODE	A less common component to match. But, if your application uses the value of this header for database validation, consider matching on this component as well.

To configure your own SQL injection conditions and rules, log in to your WAF dashboard once again by navigating to <https://console.aws.amazon.com/waf/home>.

1. Next, select the **SQL Injection** option from the navigation pane and, within that, select the **Create condition** to get started.
2. In the **Create SQL injection match condition** page, start off by providing a **Name** for your new condition. You can additionally select whether you want to enable this condition for your CloudFront CDNs (Global) or for your individual Application Load Balancers. In this case, I've opted for the **Global (CloudFront)** option, as depicted in the following screenshot:

The screenshot shows the 'Create SQL injection match condition' page. The 'Name*' field is set to 'prod-wordpress-SQLInjection'. The 'Region*' dropdown is set to 'Global (CloudFront)'. Under the 'Filter settings' section, the 'Part of the request to filter on' dropdown is set to 'Query string' and the 'Transformation' dropdown is set to 'URL decode'. An 'Add filter' button is located below these dropdowns.

3. Next, in the **Filter settings** section, select the appropriate **Part of the request to filter on** as well as the associated **Transformation** section. You can refer to the SQL injection *common conditions* table, as discussed previously, for the same. Once the values are provided, click on the **Add filter** option to complete the process.

Now, here's a really handy tip! There will definitely be cases where you will be writing more than one filter for your SQL injection condition based on different parts of the request you wish to filter, such as URI, query string, and so on. In such cases, it is always recommended to create multiple filters within the same SQL injection condition and then attach that one condition to a web ACL rule. The reason? A web request needs only to match one of the filters in the SQL injection match condition for WAF to allow or block the request based on that condition. On the other hand, if you add only one filter per SQL injection match condition, and you create more than one such SQL injection condition, the request has to match all the conditions in order for WAF to allow or block it.

The same can also be applied for protection against cross-site scripting or XSS. Cross-site scripting generally occurs when web applications include user-provided data in web pages that is sent to the browser without proper sanitization. If the data isn't properly validated or escaped, an attacker can use those vectors to embed scripts, inline frames, or other objects into the rendered page. These, in turn, can be used for a variety of malicious purposes, including stealing user credentials by using keyloggers, installing system malware, and much more. The impact of the attack is magnified if that user data persists on the server side in a data store and is then delivered to a larger set of users.

Here's a list of some common conditions and their associated configurations that you can choose to apply in your web ACL rules:

HTTP request component to match	Relevant input transformations to apply	Justification
BODY	URL_DECODE, HTML_ENTITY_DECODE	A very common component to match if your application accepts form input. AWS WAF only evaluates the first 8 KB of the body content.
QUERY_STRING	URL_DECODE, HTML_ENTITY_DECODE	Recommended if query string parameters are reflected back into the web page. An example is the current page number in a paginated list.
HEADER: Cookie	URL_DECODE, HTML_ENTITY_DECODE	A common component to match if your application accepts form input. Recommended if your application uses cookie-based parameters that are reflected back on the web page. For example, the name of the user who is currently logged in is stored in a cookie and embedded in the page header. WAF only evaluates the first 8 KB of the body content.
URI	URL_DECODE, HTML_ENTITY_DECODE	Less common, but if your application is using friendly URLs, then parameters might appear as part of the URL path segment, not the query string (they are later rewritten server side). There are similar concerns as with query strings.

To configure your own cross-site scripting conditions and rules, log in to your WAF dashboard once again by visiting <https://console.aws.amazon.com/waf/home>.

1. Next, select the **Cross-site scripting** option from the navigation pane and, within that, select the **Create condition** to get started.
2. In the **Create cross-site scripting match condition** page, start off by providing a **Name** for your new condition. You can additionally select whether you want to enable this condition for your CloudFront CDNs (Global) or for your individual Application Load Balancers. In this case, I've opted for the **Global (CloudFront)** option for now.
3. Next, from the **Part of the request to filter on** section, select the part of the request you wish WAF to filter on. You can choose between **Header**, **HTTP method**, **Query string**, **URI**, and **Body** as valid parameters. Note, however, that by selecting the **Header** option, you will be provided with an additional field in which you can either select the header from a list of headers or, alternatively, type in the name of the header.
4. Finally, select the appropriate **Transformation** operation you wish WAF to perform over the request before it is actually inspected. Once done, remember to select the **Add filter** option before completing the condition's creation process.

You now have two additional conditions that you can add to your existing web ACL, or even, go ahead and create a new web ACL. In this way, you can create different filters and conditions based on your requirements and keep attaching them to your web ACL as and when required. But this manual way of setting up rules and conditions can get a bit tricky after some time, especially when you don't have a dedicated security team and need to deploy the ACLs a lot faster into your environment. That's precisely what we are going to cover in the next section.

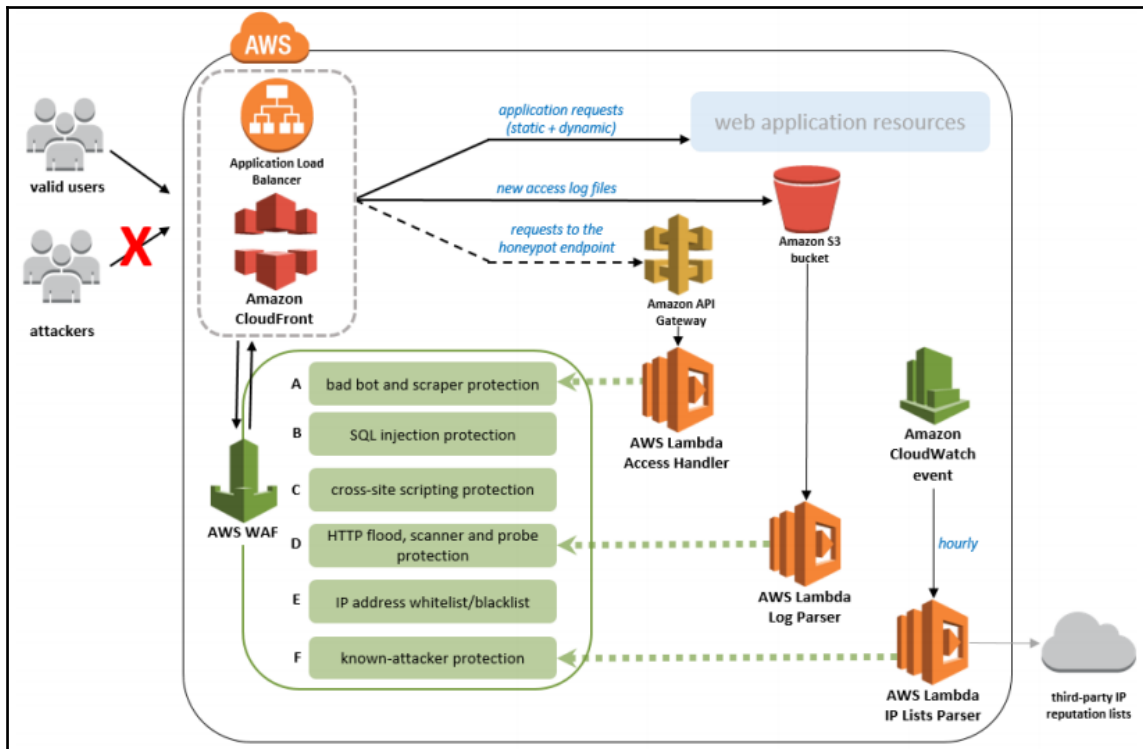
Automating WAF Web ACL deployments using CloudFormation

Working with web ACLs can be really difficult at times, especially when you have a large, distributed environment and don't necessarily have a dedicated security team to create and manage the rules on a regular basis. Luckily for us, AWS makes things far simpler by providing easy to use and customize CloudFormation templates that can spin up a single web ACL, with all the basic security conditions configured, in a mere matter of minutes! The collective solution is called AWS WAF Security Automations, and, is available free of charge for all to use. All the end user has to do is specify which security feature is required, configure that, and deploy the solution! The rest is completely taken care of by AWS itself!

The architecture of the AWS WAF Security Automation solution is relatively simple, and comprises of a few more AWS services than AWS WAF, such as AWS Lambda, Amazon CloudWatch, Amazon API Gateway, and Amazon S3, as depicted in the diagram later in this section.

At the core of the solution is the WAF service that acts as the central point for making all security-related decisions and filtering. Based on the inputs specified by the user during the CloudFormation template's configuration, the respective solution components get activated accordingly. These components are further explained here:

- **Honeypot for bad bots and scraper protection (A):** This security component automatically sets up a *honeypot* to lure and deflect a possible attack on your application. The solution provides you with an API Gateway endpoint that you need to insert into your web application as a trap to detect and lure inbound requests from various bots and scrapers. If a source accesses the trap request, an associated Lambda function intercepts that request, gathers its source IP address, and adds the same to the WAF's web ACL block list.
- **SQL injection protection (B) and cross-site scripting protection (C):** Selecting this solution enables the creation of two AWS WAF rules that provide protection against commonly occurring SQL injection or cross-site scripting patterns:



- **HTTP flood, scanner and probe protection (D):** Also called **log parsing protection**, this solution comes in handy when you want to analyze your web application's access logs for any abnormalities that can cause a potential threat. This is performed by a dedicated AWS Lambda function that does the parsing of the access logs which get stored in a S3 bucket created by the CloudFormation template itself.
- **IP address whitelist/blacklist (E):** Similar to the SQL injection and cross-site scripting solution, WAF creates two rules to allow you to manually enter IP addresses that you wish to either allow or block to your application.
- **Known-attacker protection (F):** This solution also leverages a simple Lambda function that monitors certain third-party sites for a list of potential IP addresses to block against threats. The sites include *Spamhaus* (<https://www.spamhaus.org/drop/>), *Proofpoint* (<https://rules.emergingthreats.net/fwrules/emerging-Block-IPs.txt>), and *TOR* (<https://check.torproject.org/exit-addresses>), to name a few.

With these basics in mind, let's quickly move on to deploying these solutions using the CloudFormation templates. At the time of writing this book, AWS WAF Security Automations provides two templates for use: one intended for the CloudFront CDN based deployments and the other for the Application Load Balancer. Both of the templates provide a default configuration that consists of a web ACL with eight pre-configured sets of rules that you can always change or extend as required. The following are the links to download the respective templates:

- **CloudFront CDN-based template:** <https://s3.amazonaws.com/solutions-reference/aws-waf-security-automations/latest/aws-waf-security-automations.template>
- **Application Load Balancer-based template:** <https://s3.amazonaws.com/solutions-reference/aws-waf-security-automations/latest/aws-waf-security-automations-alb.template>



You can alternatively copy the links and deploy the stacks in CloudFormation.

With the correct template downloaded, we can now move on to configuring and deploying the solution using CloudFormation:

1. To do so, first log in to your CloudFormation dashboard by navigating to <https://console.aws.amazon.com/cloudformation/home>. Note that in this case, we will be deploying the CloudFront CDN-based template in the **N.Virginia** region.
2. In the CloudFormation dashboard, select the option **Create new stack** to get started. Here, in the **Select Template** page, you can either choose to *upload* your downloaded template, or simply copy the template's URL in the **Specify an Amazon S3 template URL** field, as shown in the following screenshot. Click on **Next** to continue with the setup:

Select Template

Design a template Use AWS CloudFormation Designer to create or modify an existing template. [Learn more.](#)

Choose a template A template is a JSON/YAML-formatted text file that describes your stack's resources and their properties. [Learn more.](#)

☐ Select a sample template

☐ Upload a template to Amazon S3

☒ Specify an Amazon S3 template URL

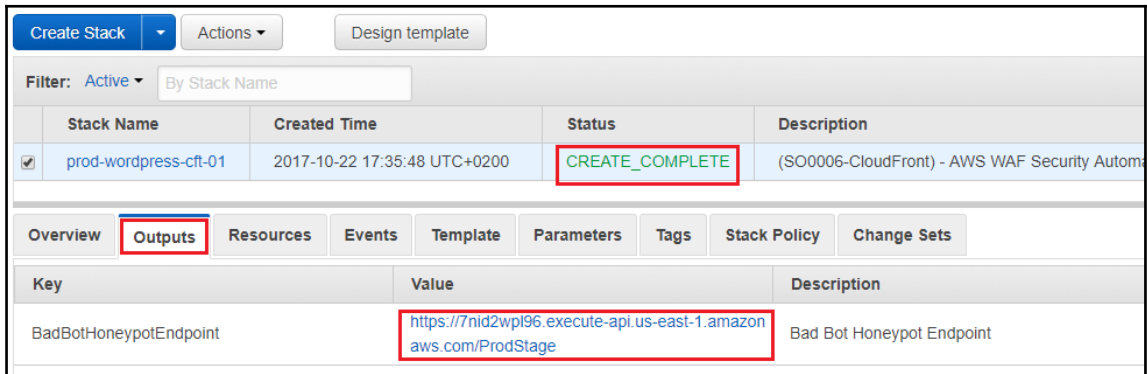
[Choose File](#) No file chosen

[View/Edit template in Designer](#)

[Cancel](#) [Next](#)

3. In the **Specify Details** page, you can start off by providing a suitable **Stack name** for your CloudFormation stack.
4. Next, in the **Parameters** section, select the **Protection services** that you wish to opt for. Remember, these are the same services that we discussed at the beginning of this section. In this case, I've opted for a rather simple setup that involves activating protection against SQL injection, cross-site scripting, and bad bots. You can alternatively select your own protection services, as required.
5. Moving on, in the **CloudFront Access Log Bucket Name** field, provide a unique name for the S3 bucket that will store your Amazon CloudFront's access logs. You can either provide the name of an existing bucket or a new one.
6. Finally, in the **Advanced Settings** section, you can additionally opt to modify a few parameters, such as **Request Threshold**, **Error Threshold**, and the **WAF Block Period**, as per your requirements. These parameters would come in handy especially if you select the **HTTP Flood Protection** or the **Activate Scanners & Probes Protection**, otherwise, you can leave these values to their defaults, as I have done in my case. Select **Next** to continue with the deployment process.

7. In the **Options** page, you can specify **Tags** for the resources that will be created by the CloudFormation template, as well as opt to provide any special *IAM Role* to allow CloudFormation to create, modify, or delete the resources in the stack. Click on **Next** to review the changes made and, finally, go ahead with the stack's creation by selecting the **Create** option on the **Review** page.
8. The stack takes a good few minutes to deploy successfully. Once done, you can verify the status of your stack's completion by checking the **Status** column, as depicted in the following screenshot:



The screenshot shows the AWS CloudFormation console. At the top, there are buttons for 'Create Stack', 'Actions', and 'Design template'. Below these is a filter section with 'Filter: Active' and a search box 'By Stack Name'. A table lists stacks with columns: Stack Name, Created Time, Status, and Description. One stack is listed: 'prod-wordpress-cft-01' with a status of 'CREATE_COMPLETE'. Below the table is a tabbed interface with 'Overview', 'Outputs', 'Resources', 'Events', 'Template', 'Parameters', 'Tags', 'Stack Policy', and 'Change Sets'. The 'Outputs' tab is selected. It shows a table with columns: Key, Value, and Description. One output is listed: 'BadBotHoneypotEndpoint' with a value of 'https://7nid2wpl96.execute-api.us-east-1.amazonaws.com/ProdStage' and a description of 'Bad Bot Honeypot Endpoint'.

Stack Name	Created Time	Status	Description
prod-wordpress-cft-01	2017-10-22 17:35:48 UTC+0200	CREATE_COMPLETE	(SO0006-CloudFront) - AWS WAF Security Autom...

Key	Value	Description
BadBotHoneypotEndpoint	https://7nid2wpl96.execute-api.us-east-1.amazonaws.com/ProdStage	Bad Bot Honeypot Endpoint

9. Additionally, based on your **Protection service** selection, you can also verify the additional outputs created by the template for your application, such as the **Honeypot Endpoint**, that's actually an Amazon API Gateway endpoint that you need to insert somewhere in your application to capture bots and scrapers. In my case, the template created a Lambda function for bad bots scraping, WAF rules for SQL injections, IP whitelisting, and XSS detection, along with an API Gateway as well.

10. With the stack up and running, you can additionally go back to the WAF dashboard and check out the individual rules that the template auto-populated against some of the protection services. In this case, the *SQL injection* and *cross-site scripting* conditions were auto-populated; however, the *IP Whitelist* and *Bad Bot* rules still require you to manually provide the IP ranges and addresses to start off with, or allow the Lambda function to inject the IP addresses into the lists at runtime as well. Here is a snapshot of the list of rules created for the SQL injection condition:

prod-wordpress-cft-01 - SQL Injection Rule

Edit rule

When a request matches at least one of the filters in the SQL injection match condition [prod-wordpress-cft-01 - SQL injection Detection](#)

Filters in prod-wordpress-cft-01 - SQL injection Detection

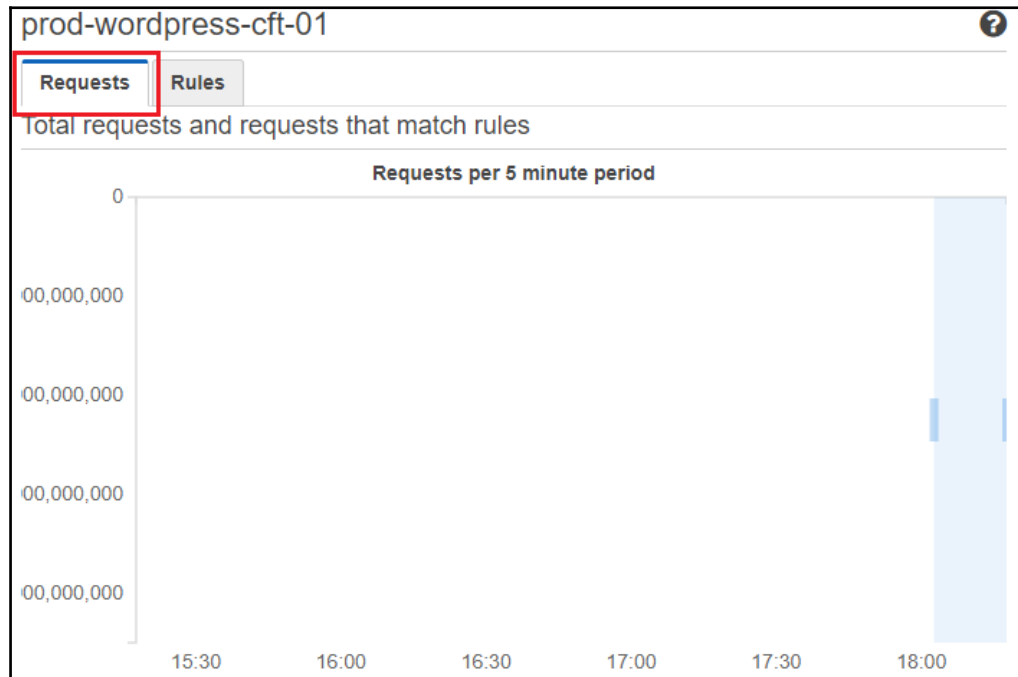
URI contains SQL injection threat after decoding as HTML tags.
Query string contains SQL injection threat after decoding as HTML tags.
Body contains SQL injection threat after decoding as URL.
Body contains SQL injection threat after decoding as HTML tags.
Query string contains SQL injection threat after decoding as URL.
URI contains SQL injection threat after decoding as URL.

With this we come to the end of this particular section. You can additionally use these templates to spin up newer web ACLs for your Application Load Balancers. Just remember to delete your stack once your testing is completed, to avoid incurring any unnecessary charges. In the next section, we will be briefly looking at how to effectively monitor your WAF rules using a few essential monitoring tools provided by AWS.

Monitoring WAF using CloudWatch

Monitoring of your WAF rules, conditions, as well as your application's web traffic, plays an important part towards identifying and mitigating possible attacks and exploits. AWS provides a wide assortment of tools and services that you, as an administrator, can leverage for the monitoring and reporting of such activities. The following are the list of services briefly explained:

- **AWS WAF dashboard:** Yes, you read it right! AWS WAF also provides a simple monitoring dashboard that lists the total requests made to your application via either the CloudFront CDN or the Application Load Balancer, as well as the number of requests that actually match to your specified rules. To view the dashboard, all you need to do is log in to your AWS WAF, select the **Web ACLs** page, and click on the **Requests** tab, as shown in the following screenshot:

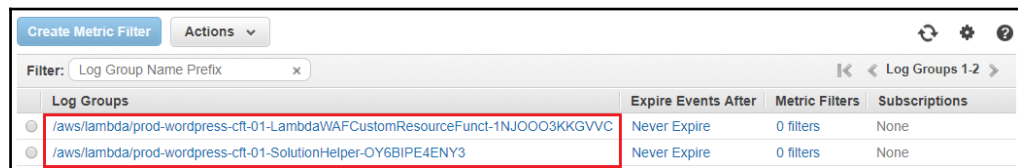


The graph aggregates and displays the requests on a five-minute period basis. You can alternatively open the same graph using Amazon CloudWatch for further analysis.

- **Amazon CloudWatch:** Amazon CloudWatch has been around for some time, and definitely provides various metrics that you can select and configure as a part of a customized requests monitoring dashboard. Here is a list of the supported WAF metrics, with a brief description:
 - **AllowedRequests:** Captures the number of allowed web requests. The valid dimensions for this metric are **Rule** and **WebACL**.
 - **BlockedRequests:** Captures the number of blocked web requests. The valid dimensions for this metric are **Rule** and **WebACL**.
 - **CountedRequests:** Typically used to test your web ACLs and rules, this metric provides a count of the web requests that match all of the conditions in a particular rule.

You can use these metrics to monitor your WAF rules, and even configure CloudWatch alarms to trigger and send notifications in case their threshold values are crossed. Based on your requirements, you can additionally take things a step further and configure CloudWatch events that trigger an appropriate Lambda function to mitigate against a possible attack, as we performed during the Security Automations solutions. You can even leverage Amazon CloudWatch to monitor the traffic flowing into the CloudFront CDNs as well as your Application Load Balancers.

- **AWS CloudTrail:** AWS CloudTrail is yet another service that you can and should leverage for parsing and analyzing your application's access and error logs, as well as logs generated by the AWS services' logs themselves. Here is a sample of few **Log Groups**, created automatically by the Security Automations Solution, for capturing WAF traffic flow and events. We will be exploring more on AWS CloudTrail in the next chapter:



Log Groups	Expire Events After	Metric Filters	Subscriptions
/aws/lambda/prod-wordpress-cft-01-LambdaWAFCustomResourceFunct-1NJ003KKGVC	Never Expire	0 filters	None
/aws/lambda/prod-wordpress-cft-01-SolutionHelper-OY6BIPE4ENY3	Never Expire	0 filters	None

With this, we come towards the end of yet another chapter, but before we sign off, here's some interesting things that I feel you ought to try out as a part of AWS WAF.

Planning your next steps

Well, we have covered a lot of new features and services in this chapter. However, there are still a few things that I recommend you need to read up on on your own. First up is the AWS Shield service!

Introduction to AWS Shield

AWS Shield is an extension of AWS WAF, but is targeted to provide security around potential DDoS attacks. It is a fully managed service that provides **Always-on** detection and automatic mitigations that minimize application downtime and latency. AWS Shield provides two tiers of services: **Standard** and **Advanced**:

- **AWS Shield Standard:** Provided at no additional costs, this service is enabled on your account and AWS services by default, and is designed to protect your web applications against the most common and frequently occurring DDoS attacks.
- **AWS Shield Advanced:** Designed for providing a higher level of protection for your web applications, AWS Shield Advanced is intended to work with applications that are currently running on Elastic or Application Load Balancers, Amazon CloudFront, and Amazon Route 53 resources. AWS Shield Advanced also provides near real-time visibility into potential attacks, along with mitigation capabilities as well. To top it all, you also get access to a dedicated 24x7 **DDoS Response Team (DRT)** that looks into potential DDoS attacks occurring on your web application, and provides quick resolutions against the same.



AWS Shield Advanced is priced at \$ 3,000 per month.

Here's a brief comparison between the various services offered by AWS Shield Standard and Advanced tiers:

Features	AWS Shield Standard	AWS Shield Advanced
Network flow monitoring	Yes	Yes
Automated application (layer 7) traffic monitoring	No	Yes
Helps protect from common DDoS attacks, such as SYN floods and UDP reflection attacks	Yes	Yes
Access to additional DDoS mitigation capacity	No	Yes
Layer 3/4 attack notification and attack forensic and history reports	No	Yes
Incident management during high-severity events	No	Yes
Custom mitigations during attacks	No	Yes
Post-attack analysis	No	Yes
Reimburse related Route 53, CloudFront, and ELB DDoS charges	No	Yes

To activate AWS Shield Advanced for your environments, simply log in to your AWS WAF dashboard and select the **Protected resources** option present under the **AWS Shield** section in the navigation pane. Here, click on the **Activate AWS Shield Advanced** button to start your Shield Advanced protection plan. Here, you will be asked to select a particular **Resource** to protect against DDoS attacks. Select your CloudFront CDN or the Elastic/Application Load Balancer, based on the resource you wish to protect, and provide a suitable **Name** for the resources that you are specifying for protection. Finally, remember to select the **Enable** checkbox to associate your resources with a web ACL, if you have one created already. Once done, select the **Add DDoS protection** option, and voila! You are up and running with AWS Shield Advanced! Simple isn't it?

Summary

Here's a quick round up of the topics that we have covered so far in this chapter.

We started off by learning and understanding a bit about the Web Application Firewall service and how it works to protect against potential security threats and exploits. We later looked at how to get started with WAF by safeguarding our previously deployed WordPress application against restrictive access by leveraging the IP Match, as well as the string/regex match conditions. We also looked at how to mitigate and safeguard your applications by leveraging advanced WAF conditions in the form of SQL injection and cross-site scripting. Towards the end, we covered how to leverage certain pre-built CloudFormation templates to automate the deployments of our WAF rules and, finally, we concluded the chapter with a brief introduction to AWS Shield and its various tiers.

In the next chapter, we will be continuing on our security journey and will cover two really amazing services as well: AWS CloudTrail and AWS Config, so stay tuned!

5

Governing Your Environments Using AWS CloudTrail and AWS Config

In the previous chapter, we learned how to leverage and utilize AWS WAF for protecting your web applications against commonly occurring web attacks and exploitations. In this chapter, we will be exploring two really useful and must-have security and governance services in the form of AWS CloudTrail and AWS Config!

Keeping this in mind, let's have a quick look at the various topics that we will be covering in this chapter:

- Introducing AWS CloudTrail, its concepts, and how it works
- Enabling CloudTrail for your AWS environment by creating your very own Trail
- Integrating and managing CloudTrail Logs using Amazon CloudWatch
- Automating Amazon CloudWatch alarms for CloudTrail using CloudFormation
- Viewing CloudTrail Logs using Amazon Elasticsearch
- Introducing Amazon Config and how it works

There is so much to do, so let's get started right away!

Introducing AWS CloudTrail

As we learned in the previous chapter, AWS provides a wide variety of tools and managed services which allow you to safeguard your applications running on the cloud, such as AWS WAF and AWS Shield. But this, however, just forms one important piece in a much larger jigsaw puzzle! What about compliance monitoring, risk auditing, and overall governance of your environments? How do you effectively analyze events occurring in your environment and mitigate against the same? Well, luckily for us, AWS has the answer to our problems in the form of AWS CloudTrail.

AWS CloudTrail provides you with the ability to log every single action taken by a user, service, role, or even API, from within your AWS account. Each action recorded is treated as an event which can then be analyzed for enhancing the security of your AWS environment. The following are some of the key benefits that you can obtain by enabling CloudTrail for your AWS accounts:

- **In-depth visibility:** Using CloudTrail, you can easily gain better insights into your account's usage by recording each user's activities, such as which user initiated a new resource creation, from which IP address was this request initiated, which resources were created and at what time, and much more!
- **Easier compliance monitoring:** With CloudTrail, you can easily record and log events occurring within your AWS account, whether they may originate from the Management Console, or the AWS CLI, or even from other AWS tools and services. The best thing about this is that you can integrate CloudTrail with another AWS service, such as Amazon CloudWatch, to alert and respond to out-of-compliance events.
- **Security automations:** As we saw in the previous chapter, automating responses to security threats not only enables you to mitigate the potential threats faster, but also provides you with a mechanism to stop all further attacks. The same can be applied to AWS CloudTrail as well! With its easy integration with Amazon CloudWatch events, you can now create corresponding Lambda functions that trigger automatically each time a compliance is not met, all in a matter of seconds!

With these key points in mind, let's have a quick look at some of CloudTrail's essential concepts and terminologies:

- **Events:** Events are the basic unit of measurement in CloudTrail. Essentially, an event is nothing more than a record of a particular activity either initiated by the AWS services, roles, or even an AWS user. These activities are all logged as API calls that can originate from the Management Console, the AWS SDK, or even the AWS CLI as well. By default, events are stored by CloudTrail with S3 buckets for a period of 7 days. You can view, search, and even download these events by leveraging the events history feature provided by CloudTrail.
- **Trails:** Trails are essentially the delivery mechanism, using which events are dumped to S3 buckets. You can use these trails to log specific events within specific buckets, as well as to filter events and encrypt the transmitted log files. By default, you can have a maximum of *five trails* created per AWS region, and this limit cannot be increased.
- **CloudTrail Logs:** Once your CloudTrail starts capturing events, it sends these events to an S3 bucket in the form of a CloudTrail Log file. The log files are JSON text files that are compressed using the `.gzip` format. Each file can contain one or more events within itself. Here is a simple representation of what a CloudTrail Log looks like. In this case, the event was created when I tried to add an existing user by the name of `Mike` to an `administrator` group using the AWS Management Console:

```
{ "Records": [{
  "eventVersion": "1.0",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "12345678",
    "arn": "arn:aws:iam::012345678910:user/yohan",
    "accountId": "012345678910",
    "accessKeyId": "AA34FG67GH89",
    "userName": "Alice",
    "sessionContext": { "attributes": {
      "mfaAuthenticated": "false",
      "creationDate": "2017-11-08T13:01:44Z"
    }}
  },
  "eventTime": "2017-11-08T13:09:44Z",
  "eventSource": "iam.amazonaws.com",
  "eventName": "AddUserToGroup",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "127.0.0.1",
  "userAgent": "AWSConsole",
  "requestParameters": {
```

```
        "userName": "Mike",  
        "groupName": "administrator"  
    },  
    "responseElements": null  
  }  
}
```

You can view your own CloudTrail Log files by visiting the S3 bucket that you specify during the trail's creation. Each log file is named uniquely using the following format:

```
AccountID_CloudTrail_RegionName_YYYYMMDDTHHmz_UniqueString.json.gz
```

Where:

- **AccountID**: Your AWS account ID.
- **RegionName**: AWS region where the event was captured: **us-east-1**, and so on.
- **YYYYMMDDTHHmz**: Specifies the year, month, day, hour (24 hours), minutes, and seconds. The **z** indicates time in UTC.
- **UniqueString**: A randomly generated 16-character-long string that is simply used so that there is no overwriting of the log files.

With the basics in mind, let's quickly have a look at how you can get started with CloudTrail for your own AWS environments!

Working with AWS CloudTrail

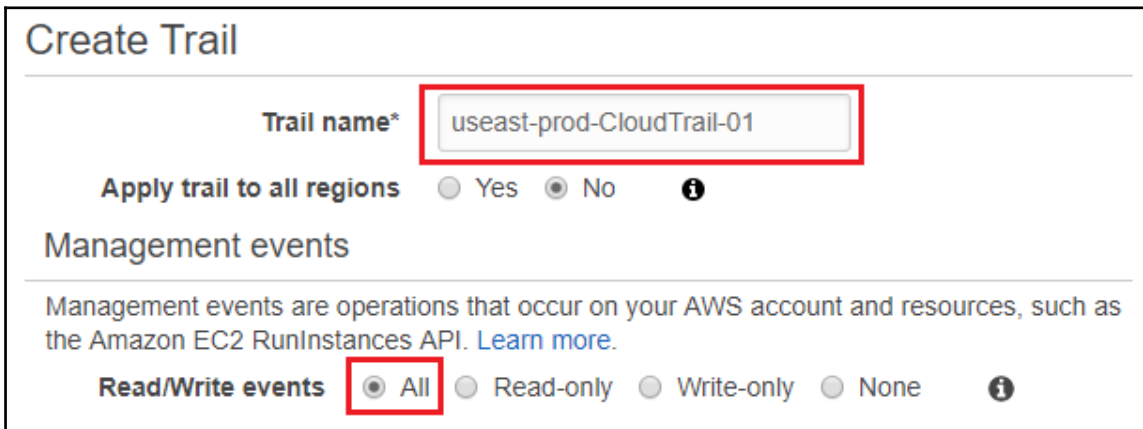
AWS CloudTrail is a fairly simple and easy to use service that you can get started with in a couple of minutes. In this section, we will be walking through a simple setup of a CloudTrail Trail using the AWS Management Console itself.

Creating your first CloudTrail Trail

To get started, log in to your AWS Management Console and filter the **CloudTrail** service from the **AWS services** filter. On the CloudTrail dashboard, select the **Create Trail** option to get started:

1. This will bring up the **Create Trail** wizard. Using this wizard, you can create a maximum of five-trails per region. Type a suitable name for the Trail in to the **Trail name** field to begin with.

2. Next, you can either opt to **Apply trail to all regions** or only to the region out of which you are currently operating. Selecting all regions enables CloudTrail to record events from each region and dump the corresponding log files into an S3 bucket that you specify. Alternatively, selecting to record out of one region will only capture the events that occur from the region out of which you are currently operating. In my case, I have opted to enable the Trail only for the region I'm currently working out of. In the subsequent sections, we will learn how to change this value using the AWS CLI:



Create Trail

Trail name*

Apply trail to all regions ☐ Yes ☒ No ⓘ

Management events

Management events are operations that occur on your AWS account and resources, such as the Amazon EC2 RunInstances API. [Learn more.](#)

Read/Write events ☒ All ☐ Read-only ☐ Write-only ☐ None ⓘ

3. Next, in the **Management events** section, select the *type* of events you wish to capture from your AWS environment. By default, CloudTrail records all management events that occur within your AWS account. These events can be API operations, such as events caused due to the invocation of an EC2 **RunInstances** or **TerminateInstances** operation, or even non-API based events, such as a user logging into the AWS Management Console, and so on. For this particular use case, I've opted to record **All** management events.



Selecting the **Read-only** option will capture all the GET API operations, whereas the **Write-only** option will capture only the PUT API operations that occur within your AWS environment.

4. Moving on, in the **Storage location** section, provide a suitable name for the S3 bucket that will store your CloudTrail Log files. This bucket will store all your CloudTrail Log files, irrespective of the regions the logs originated from. You can alternatively select an existing bucket from the **S3 bucket** selection field:

5. Next, from the **Advanced** section, you can optionally configure a **Log file prefix**. By default, the logs will automatically get stored under a folder-like hierarchy that is usually of the form `AWSLogs/ACCOUNT_ID/CloudTrail/REGION`.
6. You can also opt to **Encrypt log files** with the help of an AWS KMS key. Enabling this feature is highly recommended for production use.
7. Selecting **Yes** in the **Enable log file validation** field enables you to verify the integrity of the delivered log files once they are delivered to the S3 bucket.

8. Finally, you can even enable CloudTrail to send you notifications each time a new log file is delivered to your S3 bucket by selecting **Yes** against the **Send SNS notification for every log file delivery** option. This will provide you with an additional option to either select a predefined SNS topic or alternatively create a new one specifically for this particular CloudTrail. Once all the required fields are filled in, click on **Create** to continue.

With this, you should be able to see the newly created Trail by selecting the **Trails** option from the CloudTrail dashboard's navigation pane, as shown in the following screenshot:

Trails					
Deliver logs to an Amazon S3 bucket. CloudTrail events can be processed by one trail for free. There is a charge for processing events with additional trails. For more information, see AWS CloudTrail Pricing .					
Create trail					
Name	Region	S3 bucket	Log file prefix	CloudWatch Logs Log group	Status
useast-prod-CloudTrail-01	US East (N. Virginia)	useast-prod-cloudtrail-01			✓

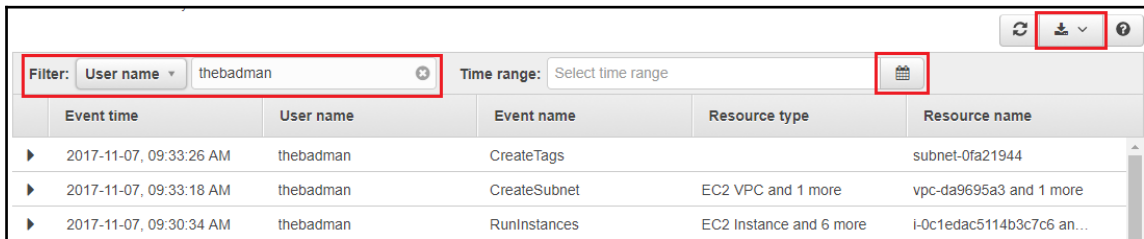
Viewing and filtering captured CloudTrail Logs and Events

With the Trail created, you can now view the captured events and filter them using the *event history* option from the CloudTrail dashboard's navigation pane. Here, you can view the last 7 days of captured events, and even filter specific ones by using one or more supporting filter attributes.

Here's a quick look at the **Filter** attributes that you can use in conjunction with the **Time range** to extract the required events and logs:

- **Event ID:** Each event captured by CloudTrail has a unique ID that you can filter and view.
- **Event name:** The name of the event. For example, EC2 events **RunInstances**, **DescribeInstances**, and so on.

- **Event source:** The AWS service to which the request was made. For example, `iam.amazonaws.com` or `ec2.amazonaws.com`.
- **Resource name:** The name or ID of the resource referenced by the event. For example, a bucket named `useast-prod-wordpress-code` or an instance ID `i-1234567` for an EC2 instance.
- **Resource type:** The type of resource referenced by the event. For example, a resource type can be a **Bucket** for S3, an **Instance** for EC2, and so on.
- **User name:** The name of the user that created or performed an action on the said event. For example, an IAM user logging into the AWS Management Console, and so on:



The screenshot shows the AWS CloudTrail console interface. At the top, there is a filter bar with a dropdown menu set to 'User name' and a text input containing 'thebadman'. To the right of the filter bar is a 'Time range' dropdown set to 'Select time range'. Below the filter bar is a table with the following columns: Event time, User name, Event name, Resource type, and Resource name. The table contains three rows of event data.

Event time	User name	Event name	Resource type	Resource name
2017-11-07, 09:33:26 AM	thebadman	CreateTags		subnet-0fa21944
2017-11-07, 09:33:18 AM	thebadman	CreateSubnet	EC2 VPC and 1 more	vpc-da9695a3 and 1 more
2017-11-07, 09:30:34 AM	thebadman	RunInstances	EC2 Instance and 6 more	i-0c1edac5114b3c7c6 an...

Once you have selected a particular filter and provided its associated attribute value, you can use the **Time range** to narrow your search results based on a predefined time window. To analyze further, you can select the **View event** option present in the details pane of an **Event** as well. Selecting this option will view the event in a JSON format, as shown in the following code:

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDAIZZ25SDDZAQTF2K3I",
    "arn": "arn:aws:iam::01234567890:user/yohan",
    "accountId": "01234567890",
    "accessKeyId": "ASDF56HJERW9PQRST",
    "userName": "yohan",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2017-11-07T08:13:26Z"
      }
    }
  },
  "invokedBy": "signin.amazonaws.com",
  "eventTime": "2017-11-07T08:25:32Z",
```

```
"eventSource": "s3.amazonaws.com",
"eventName": "CreateBucket",
"awsRegion": "us-east-1",
"sourceIPAddress": "80.82.129.191",
"userAgent": "signin.amazonaws.com",
"requestParameters": {
  "bucketName": "sometempbucketname"
},
"responseElements": null,
"requestID": "163A30A312B21AB2",
"eventID": "e7b7dff6-f196-4358-be64-aae1f5e7fed6",
"eventType": "AwsApiCall",
"recipientAccountId": "01234567890"
}
```



You can additionally select the Download icon and select whether you wish to export all the logs using the **Export to CSV** or **Export to JSON** option.

You can alternatively even download the log files by accessing your CloudTrail S3 bucket and downloading the individual compressed JSON files, as per your requirements.

With this, we come towards the end of this section. You can use these same steps and create different Trails for capturing data as well as management activities. In the next section, we will see how we can leverage the AWS CLI and update our newly-created Trail.

Modifying a CloudTrail Trail using the AWS CLI

With the Trail in place, you can now use either the AWS Management Console or the AWS CLI to modify its settings. In this case, we will look at how to perform simple changes to the newly created Trail using the AWS CLI itself. Before proceeding with this section, however, it is important that you have installed and configured the AWS CLI on your desktop/laptop, based on the guides provided at <http://docs.aws.amazon.com/cli/latest/userguide/installing.html>.

Once the CLI is installed and configured, we can now run some simple commands to verify its validity. To start off, let's first check the status of our newly-created Trail by using the `describe-trails` command, as shown in the following command:

```
# aws cloudtrail describe-trails
```

```
yoyo@YoYoNux:~$  
yoyo@YoYoNux:~$ aws cloudtrail describe-trails  
{  
  "trailList": [  
    {  
      "IncludeGlobalServiceEvents": true,  
      "Name": "useast-prod-CloudTrail-01",  
      "TrailARN": "arn:aws:cloudtrail:us-east-1:██████████:trail/useast-prod-CloudTrail-01",  
      "LogFileValidationEnabled": true,  
      "SnsTopicARN": "arn:aws:sns:us-east-1:██████████:NotifyMe",  
      "IsMultiRegionTrail": false,  
      "HasCustomEventSelectors": false,  
      "S3BucketName": "useast-prod-cloudtrail-01",  
      "SnsTopicName": "arn:aws:sns:us-east-1:██████████:NotifyMe",  
      "HomeRegion": "us-east-1"  
    }  
  ]  
}
```

This will display the essential properties of your CloudTrail Trails, such as the `Name`, the `TrailARN`, whether the log file validation is enabled or not, and whether the Trail is a multi-regional Trail or it belongs to a single region. In this case, the `IsMultiRegionTrail` value is set to `false`, which means that the Trail will only record events for its current region, that is, `us-east-1`. Let's go ahead and modify this using the AWS CLI.

To do so, we will be using the `update-trail` command:

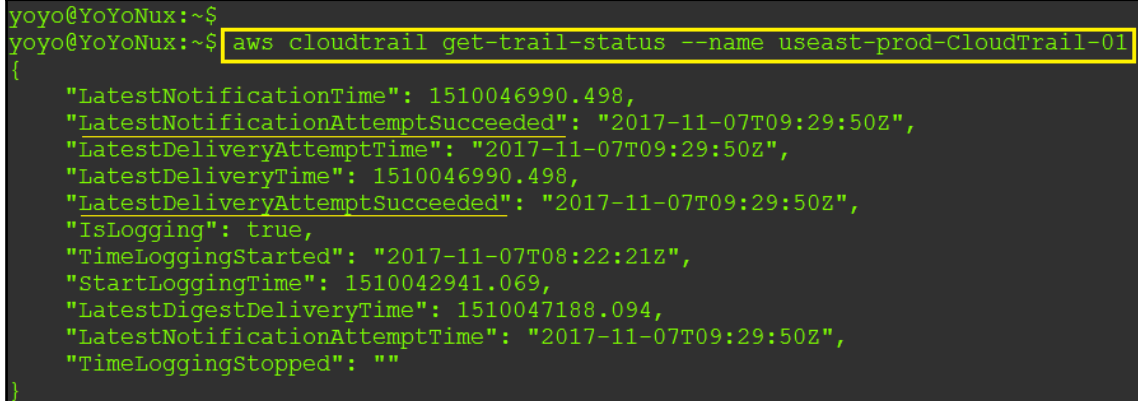
```
# aws cloudtrail update-trail \  
--name useast-prod-CloudTrail-01 \  
--is-multi-region-trail
```

The following code will simply change the `IsMultiRegionTrail` value from `false` to `true`. You can verify the same by using the `describe-trails` command, as performed earlier. Similarly, you can use the `update-trail` command to change other settings for your CloudTrail Trail, such as enabling the log file validation feature, as described in the following command:

```
# aws cloudtrail update-trail \  
--name useast-prod-CloudTrail-01 \  
--enable-log-file-validation
```

Finally, you can even use the AWS CLI to check the current status of your Trail by executing the `get-trail-status` command, as shown in the following command:

```
# aws cloudtrail get-trail-status \  
--name useast-prod-CloudTrail-01
```



```
yoyo@YoYoNux:~$  
yoyo@YoYoNux:~$ aws cloudtrail get-trail-status --name useast-prod-CloudTrail-01  
{  
  "LatestNotificationTime": 1510046990.498,  
  "LatestNotificationAttemptSucceeded": "2017-11-07T09:29:50Z",  
  "LatestDeliveryAttemptTime": "2017-11-07T09:29:50Z",  
  "LatestDeliveryTime": 1510046990.498,  
  "LatestDeliveryAttemptSucceeded": "2017-11-07T09:29:50Z",  
  "IsLogging": true,  
  "TimeLoggingStarted": "2017-11-07T08:22:21Z",  
  "StartLoggingTime": 1510042941.069,  
  "LatestDigestDeliveryTime": 1510047188.094,  
  "LatestNotificationAttemptTime": "2017-11-07T09:29:50Z",  
  "TimeLoggingStopped": ""  
}
```

Apart from these values, the `get-trail-status` command will additionally show two more fields (`LatestNotificationError` and `LatestDeliveryError`) in case an Amazon SNS subscription fails or if a CloudTrail Trail was unsuccessful at writing the events to an S3 bucket.

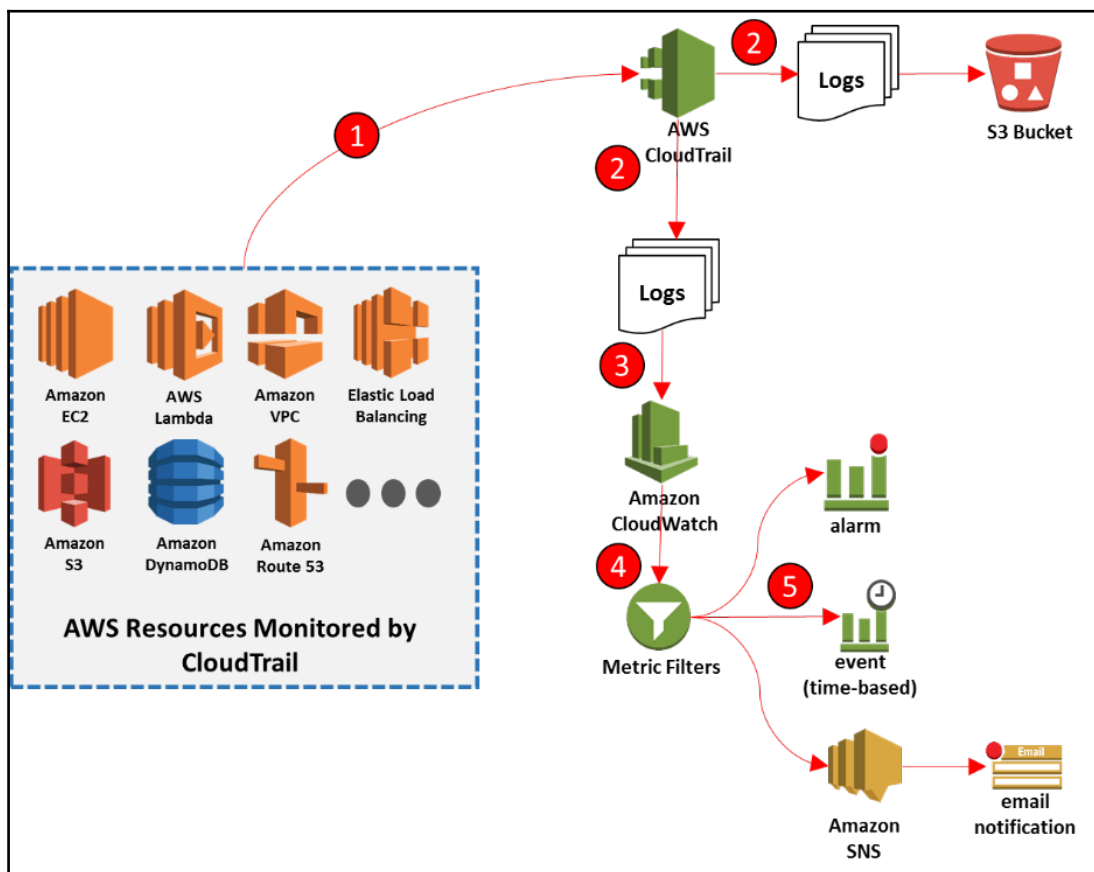
With this completed, we will now move on to the next section of this chapter, in which we will learn how you can effectively monitor your Trails with the help of CloudWatch Logs.

Monitoring CloudTrail Logs using CloudWatch

One of the best features of using CloudTrail is that you can easily integrate it with other AWS services for an enhanced security auditing and governance experience. One such service that we are going to use and explore here with CloudTrail is Amazon CloudWatch.

Using CloudWatch, you can easily set up custom metric filters and an array of alarms that can send notifications to the right set of people in case a specific security or governance issue occurs in your AWS environment. To get started with CloudWatch using CloudTrail, you will first need to configure your Trail to send the captured log events to CloudWatch Logs. This can be easily configured using both the AWS Management Console and the AWS CLI. Next, once this is done, you will be required to define custom CloudWatch metric filters to evaluate the log events for specific matches. Once a match is made, you can then additionally configure CloudWatch to trigger corresponding alarms, send notifications, and even perform a remediation action based on the type of alarm generated.

Here is a diagrammatic representation of CloudTrail's integration with CloudWatch:



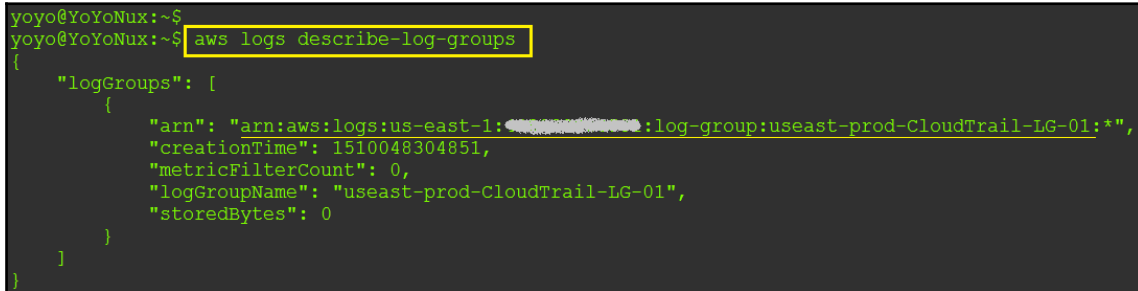
In this section, we will be using the AWS CLI to integrate the Trail's logs with Amazon CloudWatch Logs:

1. First, we will need to create a new CloudWatch Log Group using the following command:

```
# aws logs create-log-group --log-group-name useast-prod-CloudTrail-LG-01
```

2. Next, you will need to extract and maintain the newly created Log Group's ARN for the forthcoming steps. To do so, type in the following command and make a note of the Log Group's ARN, as shown here:

```
# aws logs describe-log-groups
```



```
yoyo@YoYoNux:~$  
yoyo@YoYoNux:~$ aws logs describe-log-groups  
{  
  "logGroups": [  
    {  
      "arn": "arn:aws:logs:us-east-1:123456789012:log-group:useast-prod-CloudTrail-LG-01:*",  
      "creationTime": 1510048304851,  
      "metricFilterCount": 0,  
      "logGroupName": "useast-prod-CloudTrail-LG-01",  
      "storedBytes": 0  
    }  
  ]  
}
```

3. With the Log Group successfully created, we will now need to create a new IAM Role that will essentially enable CloudTrail to send its logs over to the CloudWatch Log Group. To do so, we first need to create a policy document that assigns the AssumeRole permission to our CloudTrail Trail. Create a new file and paste the following contents into that file. Remember to create the file with a .json extension:

```
# vi policy.json  
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "",  
      "Effect": "Allow",  
      "Principal": {  
        "Service": "cloudtrail.amazonaws.com"  
      },  
      "Action": "sts:AssumeRole"  
    }  
  ]  
}
```

```
    ]  
  }  
}
```

4. With the file created, use the `create-role` command to create the role with the required permissions for CloudTrail:

```
# aws iam create-role --role-name useast-prod-CloudTrail-Role-01 \  
--assume-role-policy-document file://policy.json
```

5. Once this command executed, make a note of the newly created role's ARN. Next, copy and paste the following role policy document into a new file. This policy document grants CloudTrail the necessary permissions to create a CloudWatch Logs log stream in the Log Group that you created a while back, so as to deliver the CloudTrail events to that particular log stream:

```
# vi permissions.json  
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "CloudTrailCreateLogStream",  
      "Effect": "Allow",  
      "Action": [  
        "logs:CreateLogStream"  
      ],  
      "Resource": [  
        "<YOUR_LOG_GROUP_ARN>"  
      ]  
    },  
    {  
      "Sid": "CloudTrailPutLogEventsToCloudWatch",  
      "Effect": "Allow",  
      "Action": [  
        "logs:PutLogEvents"  
      ],  
      "Resource": [  
        "<YOUR_LOG_GROUP_ARN>"  
      ]  
    }  
  ]  
}
```

6. Next, run the following command to apply the permissions to the role. Remember to provide the name of the policy that we created during the earlier steps here:

```
# aws iam put-role-policy --role-name useast-prod-CloudTrail-  
Role-01 \  
  
--policy-name cloudtrail-policy \  
--policy-document file://permissions.json
```

7. The final step is to update the Trail with the Log Group ARN as well as the CloudWatch Logs role ARN, using the following command snippet:

```
# aws cloudtrail update-trail --name useast-prod-CloudTrail-01 \  
--cloud-watch-logs-log-group-arn <YOUR_LOG_GROUP_ARN> \  
--cloud-watch-logs-role-arn <YOUR_ROLE_ARN>
```

With this you have now integrated your CloudTrail Logs to seamlessly flow into the CloudWatch Log Group that we created. You can verify this by viewing the **Log Groups** provided under the **CloudWatch Logs** section of your CloudWatch dashboard.

In the next section, we will be leveraging this newly created Log Group and assign a custom metric as well as an alarm for monitoring and alerting purposes.

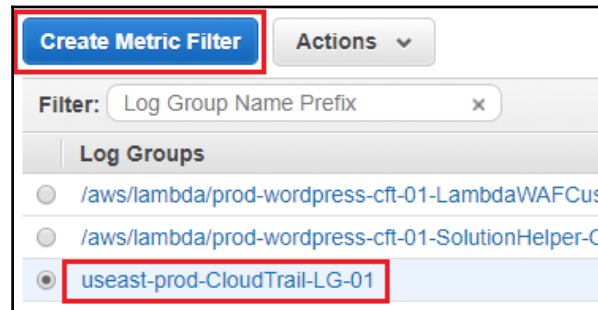
Creating custom metric filters and alarms for monitoring CloudTrail Logs

With the Log Group created and integrated with the CloudTrail Trail, we can now continue to create and assign custom metric filters as well as alarms. These alarms can be leveraged to trigger notifications whenever a particular compliance or governance issue is identified by CloudTrail.

To begin with, let's first create a custom metric filter using CloudWatch Logs. In this case, we will be creating a simple filter that triggers a CloudWatch alarm each time an S3 bucket API call is made. This API call can be either a simple PUT or DELETE operation on the bucket's policies, life cycle, and so on:

1. Log in to your Amazon CloudWatch dashboard or, alternatively, select the link provided here to get started, at <https://console.aws.amazon.com/cloudwatch/>.

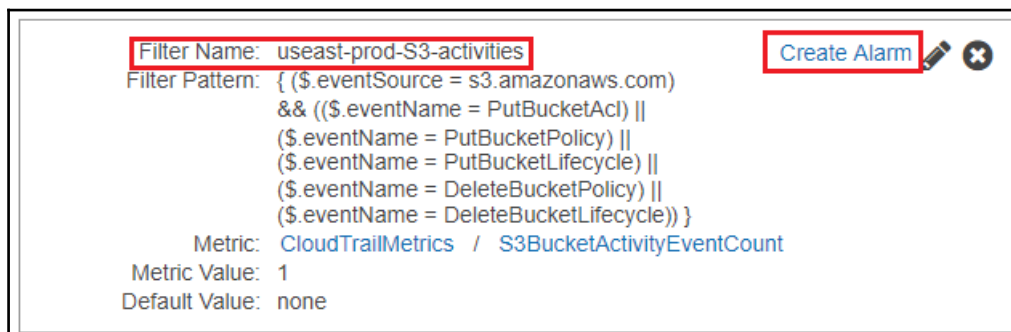
2. Once logged in, select the **Logs** option from the navigation pane. Select the newly created Log Group that we created a while back, and opt for the **Create Metric Filter** option, as depicted in the following screenshot:



3. Here, in the **Create Metric Filter and Assign a Metric** page, start off by providing a suitable **Filter Name** for the new metric, followed by populating the **Filter Pattern** option with the following snippet:

```
{ ($.eventSource = s3.amazonaws.com) && (($.eventName =
PutBucketAcl) || ($.eventName = PutBucketPolicy) || ($.eventName =
PutBucketLifecycle) || ($.eventName = DeleteBucketPolicy) ||
($.eventName = DeleteBucketLifecycle)) }
```

4. Once done, type in a suitable **Metric Namespace** value followed by a **Metric Name** as well. Leave the rest of the values to their defaults, and select the option **Create filter** to complete the process.
5. With this step completed, you now have a working CloudWatch filter up and running. In order to assign this particular filter an alarm, simply select the **Create Alarm** option adjacent to the filter, as depicted in the following screenshot:



6. Creating an alarm is a fairly straightforward and simple process, and I'm sure you would be more than qualified enough to set it up. Start off by providing a **Name** and an optional **Description** to your alarm, followed by configuring the trigger by setting the event count as ≥ 1 for 1 consecutive period. Consequently, also remember to set up the **Actions** section by selecting an SNS **Notification List** or, alternatively, creating a new one. With all the settings configured, select the **Create Alarm** option to complete the process.

With this step completed, the only thing remaining is to give the filter a try! Log in to your S3 dashboard and create a new bucket, or alternatively, update the bucket policy of an existing one. The CloudTrail Trail will pick up this change and send the logs to your CloudWatch Log Group, where our newly created metric filter triggers an alarm by notifying the respective cloud administrator! Simply awesome isn't it? You can use more custom filters and alarms for configuring CloudWatch's notifications, as per your requirements.

In the next section, we will be looking at a fairly simple and automated method for creating and deploying multiple CloudWatch alarms using a single CloudFormation template.

Automating deployment of CloudWatch alarms for AWS CloudTrail

As discussed in the previous section, you can easily create different CloudWatch metrics and alarms for monitoring your CloudTrail Log files. Luckily for us, AWS provides a really simple and easy to use CloudFormation template, which allows you to get up and running with a few essential alarms in a matter of minutes! The best part of this template is that you can extend the same by adding your own custom alarms and notifications as well. So without any further ado, let's get started with it.

The template itself is fairly simple and easy to work with. You can download a version at https://s3-us-west-2.amazonaws.com/awscloudtrail/cloudwatch-alarms-for-cloudtrail-api-activity/CloudWatch_Alarms_for_CloudTrail_API_Activity.json.

At the time of writing this book, this template supports the creation of metric filters for the following set of AWS resources:

- Amazon EC2 instances
 - IAM policies
 - Internet gateways
 - Network ACLs
 - Security groups
1. To create and launch this CloudFormation stack, head over to the CloudFormation dashboard by navigating to <https://console.aws.amazon.com/cloudformation>.
 2. Next, select the option **Create Stack** to bring up the CloudFormation template selector page. Paste https://s3-us-west-2.amazonaws.com/awsccloudtrail/cloudwatch-alarms-for-cloudtrail-api-activity/CloudWatch_Alarms_for_CloudTrail_API_Activity.json in the **Specify an Amazon S3 template URL** field, and click on **Next** to continue.
 3. In the **Specify Details** page, provide a suitable **Stack name** and fill out the following required parameters:
 - **Email:** A valid email address that will receive all SNS notifications. You will have to confirm this email subscription once the template is successfully deployed.
 - **LogGroupName:** The name of the Log Group that we created earlier in this chapter.
 4. Once the required values are filled in, click on **Next** to proceed. Review the settings of the template on the **Review** page and finally select the **Create** option to complete the process.

The template takes a few minutes to completely finish the creation and configuration of the required alarms. Here is a snapshot of the alarms and metrics that get created for your environment:

Logical ID of resources created	Type of resource
AlarmNotificationTopic	AWS::SNS::Topic
AuthorizationFailuresAlarm	AWS::CloudWatch::Alarm
CloudTrailChangesAlarm	AWS::CloudWatch::Alarm
CloudTrailChangesMetricFilter	AWS::Logs::MetricFilter

ConsoleSignInFailuresAlarm	AWS::CloudWatch::Alarm
ConsoleSignInFailuresMetricFilter	AWS::Logs::MetricFilter
EC2InstanceChangesAlarm	AWS::CloudWatch::Alarm
EC2InstanceChangesMetricFilter	AWS::Logs::MetricFilter
EC2LargeInstanceChangesAlarm	AWS::CloudWatch::Alarm
EC2LargeInstanceChangesMetricFilter	AWS::Logs::MetricFilter
GatewayChangesAlarm	AWS::CloudWatch::Alarm
GatewayChangesMetricFilter	AWS::Logs::MetricFilter
IAMPolicyChangesAlarm	AWS::CloudWatch::Alarm
IAMPolicyChangesMetricFilter	AWS::Logs::MetricFilter
NetworkAclChangesAlarm	AWS::CloudWatch::Alarm
NetworkAclChangesMetricFilter	AWS::Logs::MetricFilter
SecurityGroupChangesAlarm	AWS::CloudWatch::Alarm
SecurityGroupChangesMetricFilter	AWS::Logs::MetricFilter
VpcChangesAlarm	AWS::CloudWatch::Alarm
VpcChangesMetricFilter	AWS::Logs::MetricFilter

So far, we have seen how to integrate CloudTrail's Log files with CloudWatch Log Groups for configuring custom metrics as well as alarms for notifications. But how do you effectively analyze and manage these logs, especially if you have extremely large volumes to deal with? This is exactly what we will be learning about in the next section, along with the help of yet another awesome AWS service called **Amazon Elasticsearch!**

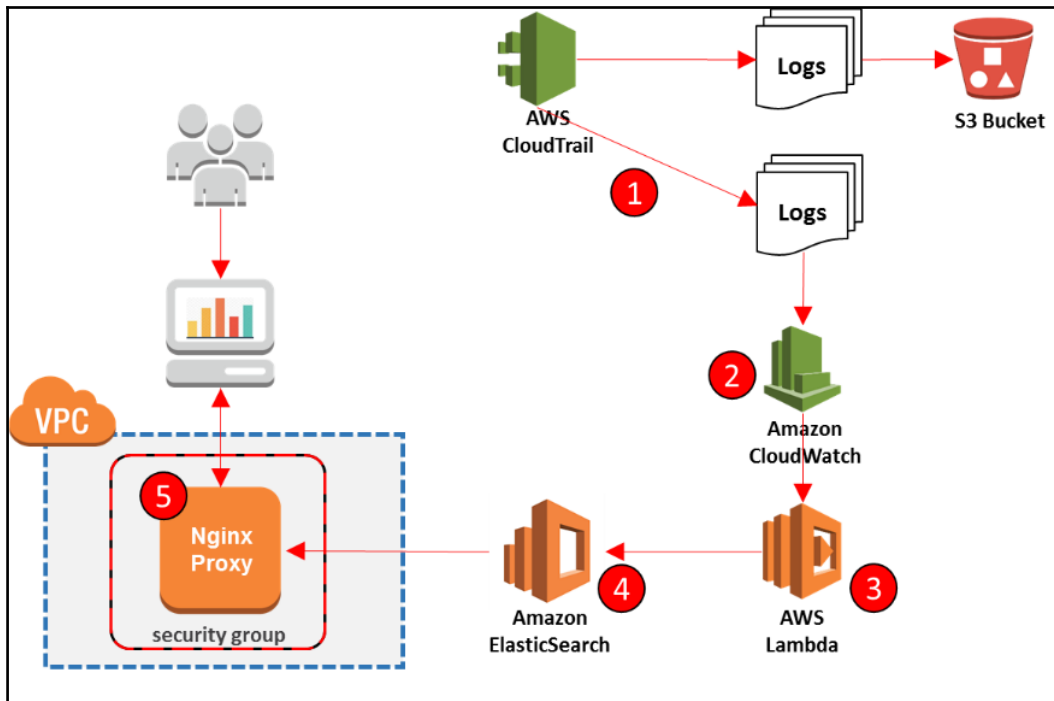
Analyzing CloudTrail Logs using Amazon Elasticsearch

Log management and analysis for many organizations starts and ends with just three letters: *E*, *L*, and *K*, which stands for Elasticsearch, Logstash, and Kibana. These three open-sourced products are essentially used together to aggregate, parse, search, and visualize logs at an enterprise scale:

- **Logstash:** Logstash is primarily used as a log collection tool. It is designed to collect, parse, and store logs originating from multiple sources, such as applications, infrastructure, operating systems, tools, services, and so on.
- **Elasticsearch:** With all the logs collected in one place, you now need a query engine to filter and search through these logs for particular events. That's exactly where Elasticsearch comes into play. Elasticsearch is basically a search server based on the popular information retrieval software library, Lucene. It provides a distributed, full-text search engine along with a RESTful web interface for querying your logs.
- **Kibana:** Kibana is an open source data visualization plugin, used in conjunction with Elasticsearch. It provides you with the ability to create and export your logs into various visual graphs, such as bar charts, scatter graphs, pie charts, and so on.

You can easily download and install each of these components in your AWS environment, and get up and running with your very own ELK stack in a matter of hours! Alternatively, you can also leverage AWS own Elasticsearch service! Amazon Elasticsearch is a managed ELK service that enables you to quickly deploy, operate, and scale an ELK stack as per your requirements. Using Amazon Elasticsearch, you eliminate the need for installing and managing the ELK stack's components on your own, which in the long run can be a painful experience.

For this particular use case, we will leverage a simple CloudFormation template that will essentially set up an Amazon Elasticsearch domain to filter and visualize the captured CloudTrail Log files, as depicted in the following diagram:



1. To get started, log in to the CloudFormation dashboard, at <https://console.aws.amazon.com/cloudformation>.
2. Next, select the option **Create Stack** to bring up the CloudFormation template selector page. Paste <http://s3.amazonaws.com/concurrencylabs-cfn-templates/cloudtrail-es-cluster/cloudtrail-es-cluster.json> in, the **Specify an Amazon S3 template URL** field, and click on **Next** to continue.

3. In the **Specify Details** page, provide a suitable **Stack name** and fill out the following required parameters:
 - **AllowedIPForEsCluster:** Provide the IP address that will have access to the nginx proxy and, in turn, have access to your Elasticsearch cluster. In my case, I've provided my laptop's IP. Note that you can change this IP at a later stage, by visiting the security group of the nginx proxy once it has been created by the CloudFormation template.
 - **CloudTrailName:** Name of the CloudTrail that we set up at the beginning of this chapter.
 - **KeyName:** You can select a key-pair for obtaining SSH to your nginx proxy instance:

Parameters

AllowedIPForEsCluster 80.82.129.191
Please specify one IP address that will have access to your Elasticsearch cluster. You can change this value later.

CloudTrailName useast-prod-CloudTrail-01 Name of the trail that will be used for auditing

KeyName yoyodev
Name of an existing EC2 KeyPair to enable SSH access to the Nginx proxy instance

LogGroupName useast-prod-CloudTrail-LG-01 Name of the CloudWatch Log Group CloudTrail will publish data to

ProxyInstanceTypeParameter t2.micro
EC2 Instance type for Nginx proxy to Elasticsearch cluster: t2 nano, micro, medium or arge. Default is t2.micro.

- **LogGroupName:** The name of the CloudWatch Log Group that will act as the input to our Elasticsearch cluster.
 - **ProxyInstanceTypeParameter:** The EC2 instance type for your proxy instance. Since this is a demonstration, I've opted for the **t2.micro** instance type. Alternatively, you can select a different instance type as well.
4. Once done, click on **Next** to continue. Review the settings of your stack and hit **Create** to complete the process.

The stack takes a good few minutes to deploy as a new Elasticsearch domain is created. You can monitor the progress of the deployment by either viewing the CloudFormation's **Output** tab or, alternatively, by viewing the Elasticsearch dashboard. Note that, for this deployment, a default **t2.micro.elasticsearch** instance type is selected for deploying Elasticsearch. You should change this value to a larger instance type before deploying the stack for production use.



You can view information on Elasticsearch *Supported Instance Types* at <http://docs.aws.amazon.com/elasticsearch-service/latest/developerguide/aes-supported-instance-types.html>.

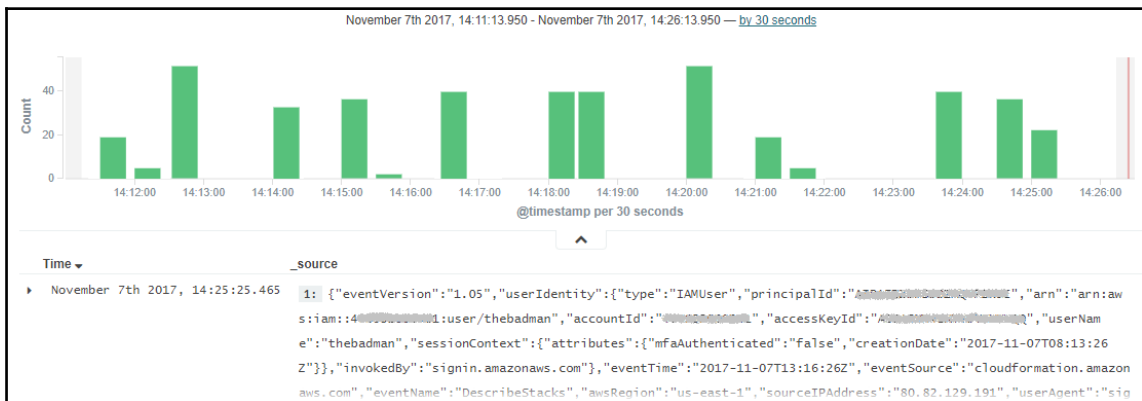
With the stack deployed successfully, copy the Kibana URL from the CloudFormation **Output** tab:

```
"KibanaProxyEndpoint": "http://<NGINX_PROXY>/_plugin/kibana/"
```

The Kibana UI may take a few minutes to load. Once it is up and running, you will need to configure a few essential parameters before you can actually proceed. Select **Settings** and hit the **Indices** option. Here, fill in the following details:

- **Index contains time-based events:** Enable this checkbox to index time-based events
- **Use event times to create index names:** Enable this checkbox as well
- **Index pattern interval:** Set the **Index pattern interval** to **Daily** from the drop-down list
- **Index name of pattern:** Type `[cwl-]YYYY.MM.DD` in to this field
- **Time-field name:** Select the `@timestamp` value from the drop-down list

Once completed, hit **Create** to complete the process. With this, you should now start seeing logs populate on to Kibana's dashboard. Feel free to have a look around and try out the various options and filters provided by Kibana:



Phew! That was definitely a lot to cover! But wait, there's more! AWS provides yet another extremely useful governance and configuration management service that we need to learn about as well, so without any further ado, here's introducing AWS Config!

Introducing AWS Config

AWS Config is yet another managed service, under the security and governance wing of services, that provides a detailed view of the configurational settings of each of your AWS resources. Configurational settings here can be anything, from simple settings made to your EC2 instances or VPC subnets, to how one resource is related to another, such as how an EC2 instance is related with an EBS volume, an ENI, and so on. Using AWS Config, you can actually view and compare such configurational changes that were made to your resource in the past, and take the necessary preventative actions if needed.

Here's a list of things that you can basically achieve by using AWS Config:

- Evaluate your AWS resource configurations against a desired setting
- Retrieve and view historical configurations of one or more resources
- Send notifications whenever a particular resource is created, modified, or deleted
- Obtain a configuration snapshot of your resource that you can later use as a blueprint or template
- View relationships and hierarchies between resources, such as all the instances that are part of a particular network subnet, and so on

Using AWS Config enables you to manage your resources more effectively by setting governing policies and standardizing configurations for your resources. Each time a configuration change is violated, you can trigger off notifications or even perform a remediation against the change. Furthermore, AWS Config also provides out-of-the-box integration capabilities with the likes of AWS CloudTrail, as well to providing you with a complete end-to-end auditing and compliance monitoring solution for your AWS environment.

Before we get started by setting up AWS Config for our own scenario, let's first take a quick look at some of its important concepts and terminologies.

Concepts and terminologies

The following are some of the key concepts and terminologies that you ought to keep in mind when working with AWS Config:

- **Config rules:** Config rules form the heart of operations at AWS Config. These are essentially rules that represent the desired configuration settings for a particular AWS resource. While the service monitors your resources for any changes, these changes get mapped to one or more set of config rules, that in turn flag the resource against any non-compliances. AWS Config provides you with some rules out of the box that you can use as-is or even customize as per your requirements. Alternatively, you can also create custom rules completely from scratch.
- **Configuration items:** Configuration items are basically a point-in-time representation of a particular AWS resource's configuration. The item can include various metadata about your resource, such as its current configuration attributes, and its relationships with other AWS resources, if any, its events, such as when it was created, last updated, and so on. Configuration items are created by AWS Config automatically each time it detects a change in a particular resource's configuration.
- **Configuration history:** A collection of configuration items of a resource over a particular period of time is called its **configuration history**. You can use this feature to compare the changes that a resource may undergo overtime, and then decide to take necessary actions. Configuration history is stored in an Amazon S3 bucket that you specify.
- **Configuration snapshot:** A configuration snapshot is also a collection of configuration items of a particular resource over time. This snapshot acts as a template or benchmark that can then be used to compare and validate your resource's current configurational settings.

With this in mind, let's look at some simple steps which allow you to get started with your own AWS Config setup in a matter of minutes!

Getting started with AWS Config

Getting started with AWS Config is a very simple process, and it usually takes about a minute or two to complete. Overall, you start off by specifying the resources that you want AWS Config to record, configure an Amazon SNS topic, and Amazon S3 bucket for notifications and storing the configuration history, and, finally, add some config rules to evaluate your resources:

1. To begin, access the AWS Config dashboard by filtering the service from the AWS Management Console or by navigating to <https://console.aws.amazon.com/config/>.
2. Since this is our first time configuring this, select the **Get Started** option to commence the Config's creation process.
3. In the **Resource types to record** section, select the type of AWS resource that you wish config to monitor. By default, config will record the activities of all supported AWS resources. You can optionally specify only the services which you want to monitor by typing in the **Specific types** field, as shown in the following screenshot. In this case, I've opted to go for the default values: **Record all resources supported in this region** and **Include global resources**:

Resource types to record

All resources ☒ Record all resources supported in this region ⓘ

☒ Include global resources (e.g., AWS IAM resources) ⓘ

Specific types

Amazon S3 bucket*

Your bucket receives configuration history and configuration snapshot files, which contain details for the resources that AWS Config records.

☒ Create a bucket

☐ Choose a bucket from your account

☐ Choose a bucket from another account ⓘ

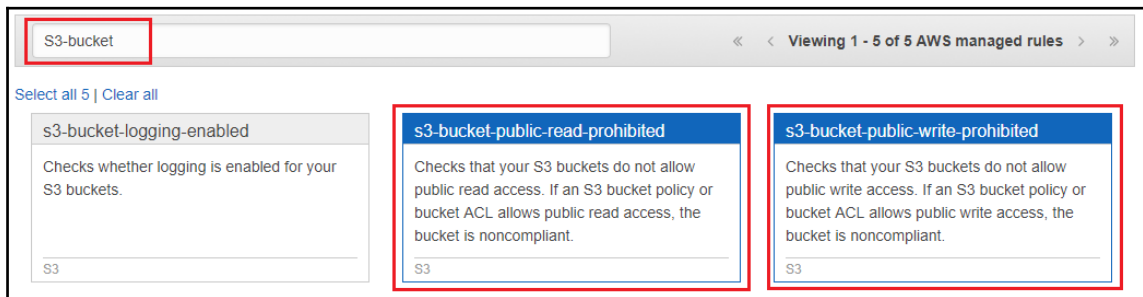
Bucket name* / Prefix (optional) / AWSLogs/██████████/Config/us-east-1

4. Next, select a location to store your configuration history as well as your configuration snapshots. In this case, I've opted to create a new S3 bucket for AWS Config by providing a unique **Bucket name**.
5. Moving on, in the **Amazon SNS topic** section, you can choose to create a new SNS topic that will send email notifications to your specified mailbox, or choose a pre-existing topic from your account.

6. Finally, you will need to provide config with a **Read-only** access role so that it can record the particular configuration information as well as send that over to S3 and SNS. Based on your requirements, you can either **Create a role** or, alternatively, **Choose a role from your account**. Click **Save** to complete the basic configuration for your AWS Config.

With this step completed, we can now go ahead and add Config rules to our setup. To do so, from the AWS Config dashboard's navigation pane, select the **Rules** and click on the **Add rule** option.

7. In the **AWS Config rules** page, you can filter and view predefined rules using the *filter* provided. For this particular scenario, let's go ahead and add two rules for checking whether any of the account's S3 buckets have either public read prohibited or public write prohibited on them or not. To do so, simply type in `S3-bucket` in the filter and select either of the two config rules, as shown in the following screenshot:



- **Resources:** When any resource that matches the evaluation criteria is either created, modified, or deleted
 - **Tags:** When any resource with the specified tag is created, modified, or deleted
 - **All changes:** When any resource recorded by AWS Config is created, modified, or deleted
8. Selecting a particular rule will pop up that rule's configuration page, where you can define the rule's trigger as well as its scope. Let's pick the **s3-bucket-public-read-prohibited** rule for starters and work with that.

9. In the **Configure rule** page, provide a suitable **Name** and **Description** for your new rule. Now, since this is a managed rule, you will not be provided with an option to change the **Trigger type**; however, when you create your own custom rules, you can specify whether you wish to trigger the rule based on a **Configuration change** event or using a **Periodic** check approach that uses a time frequency that you specify to evaluate the rules.
10. Next, you can also specify when you want the rule's evaluations to occur by selecting the appropriate options provided under the **Scope of changes** section. In this case, I've opted for the **Resources** scope and selected **S3: Bucket** as the resource, as depicted in the following screenshot:

Trigger

AWS Config evaluates resources when the trigger occurs.

Trigger type* ☒ Configuration changes ☐ Periodic ⓘ

Scope of changes* ☒ **Resources** ☐ Tags ☐ All changes ⓘ

Resources*

This rule can be triggered only when recorded resources are created, changed, or deleted. Specify which resources are recorded on the Settings page.

11. Optionally, you can also provide the ARN of the resource that you wish config to monitor using the **Resource identifier** field. Click on **Save** once done.

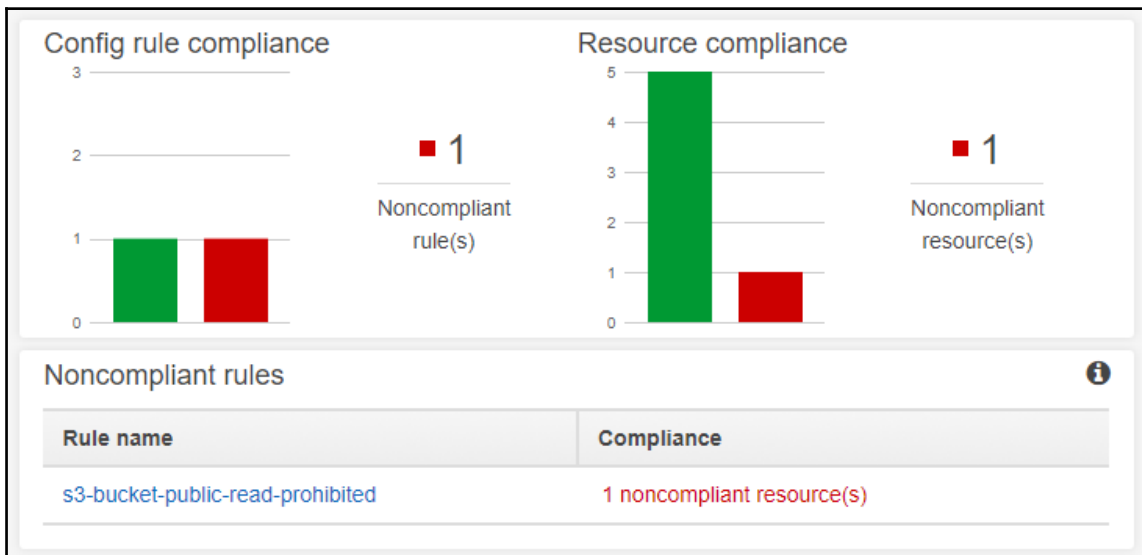
Similarly, using the aforementioned steps, create another managed config rule called **s3-bucket-public-write-prohibited**.

With the rules in place, select the **Resources** option from the config's navigation pane to view the current set of resources that have been evaluated against the set compliance.

In my case, I have two S3 buckets present in my AWS environment: one that has public read enabled on it while the other doesn't. Here's what the **Resources evaluated** dashboard should look like for you:

Resources evaluated					
Click on the  icon to view configuration details for the resource when it was last evaluated with this rule.					
Resource type	Config timeline	Compliance	Last successful invocation	Last successful evaluation	Manage resource
▶ S3 Bucket	yetanothercloudguy.com	Noncompliant	November 07, 2017 2:53:16 PM	November 07, 2017 2:53:16 PM	
S3 Bucket	cloudtrailescuster-s3cloudtrailbucket-13vimgkudb2uf	Compliant	November 07, 2017 2:53:16 PM	November 07, 2017 2:53:16 PM	

Here, you can view the evaluated resources against a **Config timeline** by simply selecting the name of the resource from the column with the same name. This will bring up a time series of your particular resource's configuration state. You can choose between the different time series options to view the state changes, as well as toggle between the time periods using the Calendar icon. The best part of using this feature of config is that you can simultaneously change your resource's configuration by selecting the **Manage resource** option. Doing so will automatically open the S3 buckets configuration page, as in this case. You can alternatively select the **Dashboard** option from AWS Config navigation pane and obtain a visual summary of the current status of your overall compliance, as depicted in the following screenshot:



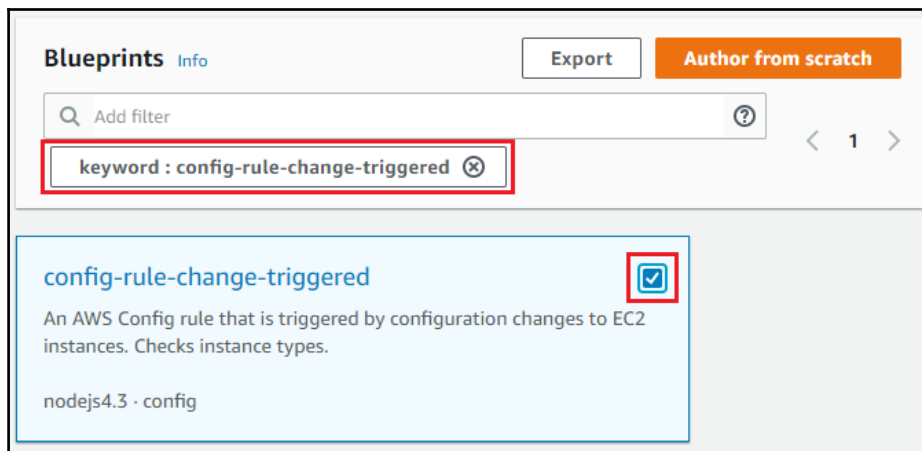
You can use the same concepts to create more such managed config rules for a variety of other AWS services, including EC2, EBS, Auto Scaling, DynamoDB, RDS, Redshift, CloudWatch, IAM, and much more! For a complete list of managed rules, check out <http://docs.aws.amazon.com/config/latest/developerguide/managed-rules-by-aws-config.html>.

With the managed config rules done, the last thing left to do is create a customized config rule, which is exactly what we will be covering in the next section.

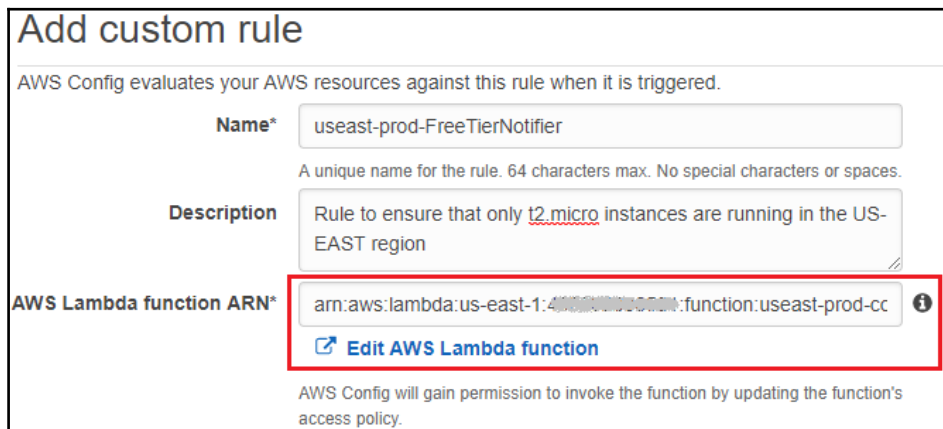
Creating custom config rules

The process for creating a custom config rule remains more or less similar to the earlier process, apart from a few changes here and there. In this section, we will be exploring how to create a simple compliance rule that will essentially trigger a config compliance alert if a user launches an EC2 instance other than the **t2.micro** instance type:

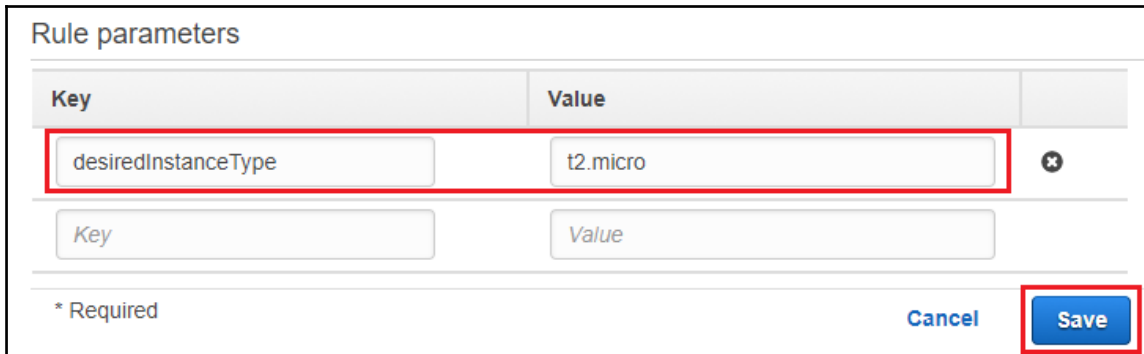
1. To get started, select the **Rules** option from the AWS Config navigation pane, then select the **Add custom rule** button present on the **Add rule** page. The creation of the custom rule starts off like any other, by providing a suitable **Name** and **Description** for the rule. Now, here's where the actual change occurs. Custom config rules rely on AWS Lambda to monitor and trigger the compliance checks. And this is actually perfect, as Lambda functions are event driven and perfect for hosting the business logic for our custom rules.
2. Select the **Create AWS Lambda function** to get things started. Here, I'm going to make use of a pre-defined Lambda blueprint that was essentially created to work in conjunction with AWS Config. Alternatively, you can create your config rule's business logic from scratch, and deploy the same in a fresh function. For now, type in the following text in the **Blueprints** filter, as shown in the following screenshot (**config-rule-change-triggered**):



3. Ensure that the blueprint is selected, and click on **Next** to continue.
4. In the function's **Basic Information** page, provide a **Name** for your function followed by selecting the **Create new role from template(s)** option from the **Role** drop-down list. The role will essentially provide the Lambda function with the necessary permissions to read from EC2 and write the output back to AWS Config as well as to Amazon CloudWatch.
5. Type in a suitable **Role name** and select the **Create function** option to complete the process. Once the function is deployed, make a note of its ARN, as we will be requiring the same in the next step.
6. Return back to the AWS Config **Add custom rule** page and paste the newly created function's ARN in the **AWS Lambda function ARN** file, as shown in the following screenshot:



7. With the function's ARN pasted, the rest of the configuration for the custom rule remains the same. Unlike the managed rules, you can opt to change the **Trigger type** between **Configuration changes** or **Periodic**, as per your requirements. In this case, I've opted to go for the **Condition changes** as my trigger mechanism, followed by **EC2: Instance** as the **Resource type**.
8. Last, but not least, we also need to specify the **Rule parameters**, which is basically a key-value pair that defines an attribute against which your resources will be validated. In this case, **desiredInstanceType** is the **Key** and **t2.micro** is the **Value**. Click **Save** to complete the setup process:



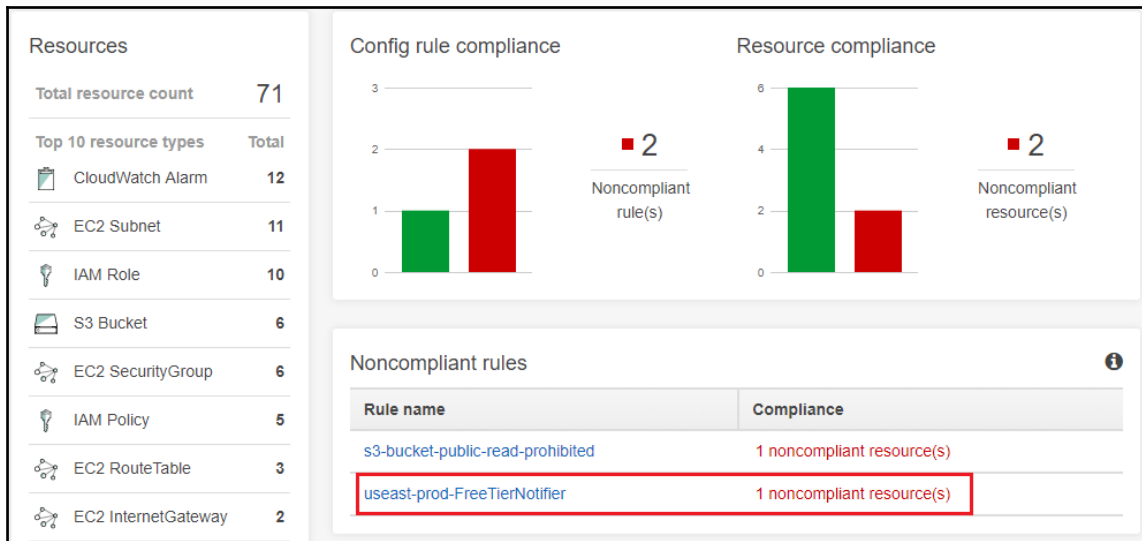
Rule parameters

Key	Value	
desiredInstanceType	t2.micro	✕
Key	Value	

* Required

Cancel Save

9. With the rule in place, all you need to do now is take it for a small test run! Go ahead and launch a new EC2 instance that is other than **t2.micro**. Remember that the instance has to be launched in the same region as that of your Lambda function! Sure enough, once the instance is launched, the change gets immediately reflected in AWS Config's dashboard:



With this, we come towards the end of this section as well as the chapter! However, before we conclude, here's a quick look at some interesting best practices and next steps that you ought to keep in mind when working with AWS CloudTrail and AWS Config!

Tips and best practices

Here's a list of a few essential tips and best practices that you ought to keep in mind when working with AWS CloudTrail, AWS Config, and security in general:

- **Analyze and audit security configurations periodically:** Although AWS provides a variety of services for safeguarding your cloud environment, it is the organization's mandate to ensure that the security rules are enforced and periodically verified against any potential misconfigurations.
- **Complete audit trail for all users:** Ensure that all resource creation, modifications, and terminations are tracked minutely for each user, including root, IAM, and federated users.

- **Enable CloudTrail globally:** By enabling logging at a global level, CloudTrail can essentially capture logs for all AWS services, including the global ones such as IAM, CloudFront, and so on.
- **Enable CloudTrail Log file validation:** An optional setting, however it is always recommended to enable CloudTrail Log file validations for an added layer of integrity and security.
- **Enable access logging for CloudTrail and config buckets:** Since both CloudTrail and config leverage S3 buckets to store the captured logs, it is always recommended that you enable access tracking for them to log unwarranted and unauthorized access. Alternatively, you can also restrict access to the logs and buckets to a specialized group of users as well.
- **Encrypt log files at rest:** Encrypting the log files at rest provides an additional layer of protection from unauthorized viewing or editing of the logged data.

Summary

Well this has surely been a really interesting chapter to cover! Before we move ahead with the next chapter, let's quickly summarize all that we have learned so far!

We started off the chapter with a brief overview of AWS CloudTrail, along with a small step-by-step guide on getting started with your very own CloudTrail Trail. We also learned about AWS CloudTrail Logs, and their integration capabilities with Amazon CloudWatch Logs for better alerting and notifications capabilities. We also leveraged a couple of CloudFormation templates to deploy pre-configured CloudWatch alarms for monitoring our Trail, as well as setting up an entire Amazon Elasticsearch domain for viewing and filtering the CloudTrail Logs. Last, but not least, we also covered AWS Config as a configuration management and compliance service by deploying both managed as well as custom config rules.

In the next chapter, we will be continuing and concluding our security journey with two really amazing services: AWS IAM and AWS Organizations, so stay tuned!

6

Access Control Using AWS IAM and AWS Organizations

In the previous chapter, we learnt and explored about how you can leverage two AWS services, namely AWS Config and AWS CloudTrail, to govern your Cloud environments. In this chapter, we will be continuing on the security journey by revisiting AWS IAM along with a few useful features as well as learning a bit about yet another service in the form of AWS Organizations!

Keeping this in mind, let us have a quick look at the various topics that we will be covering in this chapter:

- What's new with AWS IAM
- Creating policies using the IAM visual editor
- Testing your IAM policies using the IAM Policy Simulator
- Introducing AWS Organizations with a few essential concepts and terminologies
- Creating your own organizations using the AWS Management Console as well as the AWS CLI

What's new with AWS IAM

Before we look at some of the recent enhancements made to IAM, here is a quick crash course on IAM for the uninitiated. AWS Identity and Access Management or IAM is a web service that provides secured access control mechanisms for all AWS services. You can use IAM to create users and groups, assigning users specific permissions and policies, and a lot more. The best part of all this is that IAM is completely FREE. Yup! Not a penny is required to use it.

Let's quickly look at some interesting features provided by AWS IAM:

- **Multi-factor authentication:** IAM allows you to provide two-factor authentications to users for added security. This means that now, along with your password, you will also have to provide a secret key/pin from a special hardware device, such as a hard token, or even from software apps such as Google Authenticator.
- **Integration with other AWS products:** IAM integrates with almost all AWS products and services and can be used to provide granular access rights and permissions to each service as required.
- **Identity federation:** Do you have an on-premise Active Directory already that has users and groups created? Not a problem, as IAM can be integrated with an on-premise AD to provide access to your AWS account using a few simple steps.
- **Access mechanisms:** IAM can be accessed using a variety of different tools, the most common and frequently used being the AWS Management Console. Apart from this, IAM can also be accessed via the AWS CLI, via SDKs that support different platforms and programming languages such as Java, .NET, Python, Ruby, and so on, and programmatically via a secured HTTPS API as well.

With the basics in mind, let us now look at some interesting and useful enhancements made to IAM in recent years.

Using the visual editor to create IAM policies

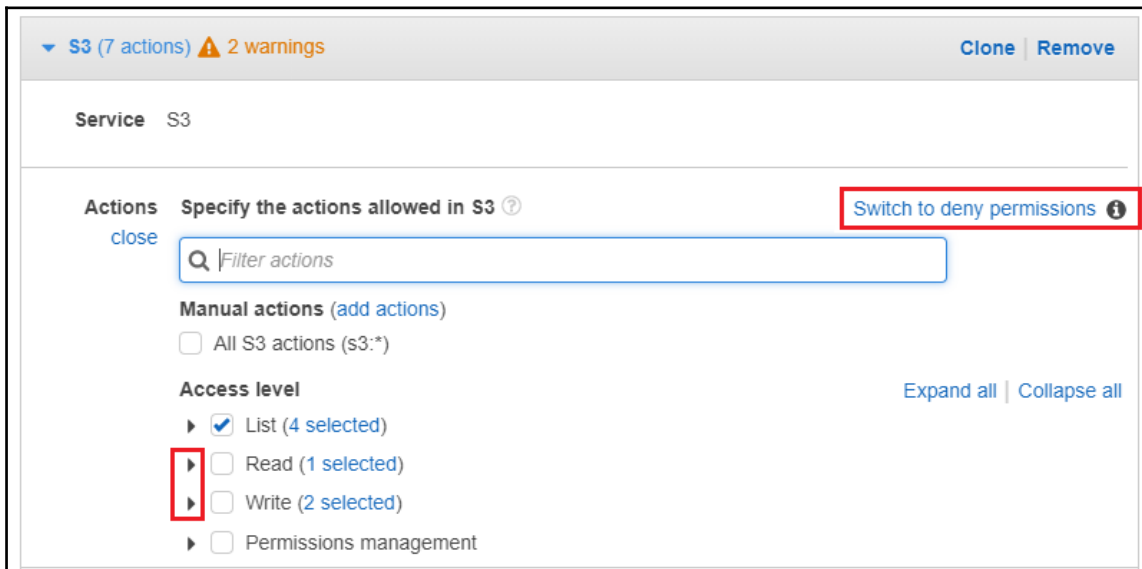
IAM policies are used to define permissions for your IAM entities such as users, groups, and roles. Each policy that you create consists of one or more statements that include the following elements:

- **Effect:** This element determines whether a policy statement allows or explicitly denies access to a particular IAM resource.
- **Action:** Actions are used to define AWS service actions within a policy, for example; you can specify Amazon S3 related actions such as list buckets, read or write to buckets, and so on.
- **Resource:** Resources are the AWS services or individual entities to which the actions apply.
- **Condition:** Conditions are used to define when a particular permission is allowed or denied on a resource. You can leverage one or more conditions to provide additional granular security to your AWS resources.

Once a policy is created, you essentially attach it to your resource which can be an IAM user, group or even a role. However, creating custom and granular IAM policies can prove to be a challenge at times especially if you are just getting started with AWS. To address this, AWS has provided a new and improved visual editor using which you can easily create customized policies on your own:

1. To get started with the visual editor, first log in to the IAM Management Console by selecting this URL <https://console.aws.amazon.com/iam/home>.
2. Once logged in, select the **Policies** option from the navigation pane. This will display a page that lists both the **AWS Managed** as well as the **Customer Managed** policies. To create a policy, simply select the **Create policy** option. For this scenario, let us create a simple S3 policy that grants full access to only a single folder present within an S3 bucket.
3. On the **Create policy** page, ensure that the **Visual editor** tab is selected and click on the **Choose a service** option to filter and pick out Amazon S3.

4. With the **Service** selected, next click on the **Select actions** options to add the appropriate actions for our policy. Here, you can select the appropriate **Access levels** for your resource by either opting to add the actions *manually* or even provide individual permissions by selecting the correct access rights from each individual **Access levels**. In this case, I have opted to select the entire **List** level for permissions and the `s3:GetObject` from the **Read** access level followed by the `s3:PutObject`, and the `s3:DeleteObject` permissions from the **Write** access level. Following is screenshot of the **Actions** selected for your reference:



By default, all actions selected here will be allowed. To deny actions explicitly, select the **Switch to deny permissions** option provided in the **Actions** section.

5. Once completed, you can now select the **Resources** section to add either all or specific resources to your new set of permissions. In this case, we will add the set of permissions to a specific bucket called **useast-prod-stuff-01**. To do so, select the **Add ARN** option adjoining to the bucket field.
6. In the **Add ARN(s)** dialog box, type in the name of the selected bucket in the **Bucket name** field as depicted in the following screenshot. Once done, select **Add** to complete the process:

Add ARN(s)

Amazon Resource Names (ARNs) uniquely identify AWS resources. Resources are unique to each service. [Learn more](#)

Specify ARN for bucket [List ARNs manually](#)

arn:aws:s3:::useast-prod-stuff-01

Bucket name useast-prod-stuff-01 ☐ Any

Cancel Add

7. Once done, you can optionally choose to add permissions to an object level as well. Click on the **Add ARN** option adjoining the **object** field and fill in the required information as we performed in the previous steps. The only addition here is that you can provide an additional *folder name* in the **Object name** field for a more granular permission control. Once done, click on **Add** to complete the process. Here is a screenshot of the completed resource allocation for our policy:

Resources ☒ Specific ☐ All resources [close](#)

bucket ? arn:aws:s3:::useast-prod-stuff-01 [EDIT](#) ☐ Any

[Add ARN](#) to restrict access

object ? arn:aws:s3:::useast-prod-stuff-01/Dummy [EDIT](#) ☐ Any

[Add ARN](#) to restrict access

8. With the permissions and the resources set, you can optionally choose to add **Conditions** as well to your policy. To do so, select the **Specify request conditions** option. Here you can opt to select and edit conditions that match your requirements. For example, for this particular policy, we want the user to access the particular S3 bucket only from within an organization's internal network. To do so, select the **Source IP** option and type in suitable IP range or a single IP address based on your organization's IP range.

9. You can also add additional conditions to your policy by selecting the **Add request condition** option. Here, you can customize and specify a granular condition using a combination of **Condition key**, **Qualifier**, and **Operator** as shown in the following screenshot:

The screenshot shows a dialog box titled "Add request condition" with a close button (X) in the top right corner. The dialog contains the following fields and options:

- Condition key***: A dropdown menu with "aws:username" selected.
- Qualifier**: A dropdown menu with "For all values in request" selected.
- Operator***: A dropdown menu with "StringEquals" selected.
- Value***: A text input field containing "joe".
- If exists**: A checkbox that is checked.
- Buttons**: "Cancel" and "Add" buttons at the bottom right. The "Add" button is highlighted with a red box.
- Footer**: A blue link with a plus icon and the text "Add another condition value".

Here, the condition will check and allow only a particular AWS IAM user with the username **joe** access to the S3 bucket. You can create your own custom conditions using the same approach and once done, click on **Add** to complete the process.

10. With the permissions, resources, and conditions in place, select the **Review policy** option to complete the policy creation process. Provide a suitable **Name** and an optional **Description** for your policy before selecting the **Create policy** option.

There you have it! Simple, wasn't it! You can use the same process to create highly customized and granular policies with relative ease. However, there is still one question that remains unanswered; how do we test and troubleshoot the policies without making any actual requests? That's exactly what the IAM Policy Simulator is all about!

Testing IAM policies using the IAM Policy Simulator

With your new policy created, the next steps would be to attach it to either an IAM user or group and test it out. But how do you effectively test your new policy without having to make any actual calls or requests? That's where the new IAM Policy Simulator comes into play!

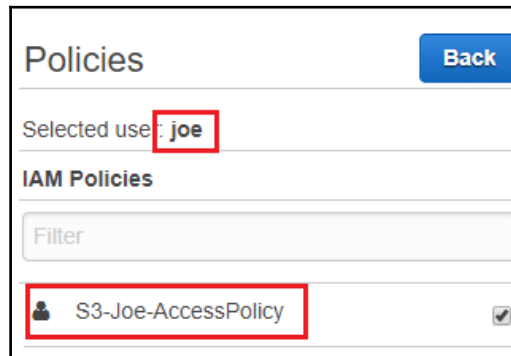
The IAM Policy Simulator is used to evaluate IAM policies in order to determine the most effective set of permissions and actions that you can specify without making any actual resource calls whatsoever. The policy simulator internally leverages the same policy evaluation engine that processes real requests to AWS resources; however, it does not make any actual service request itself. Because of this nature, the policy simulator is unable to report any responses from the generated requests. All you get as a result is whether the policy would allow or deny a particular action. Here are a few ways using which you can leverage the IAM Policy Simulator:

- You can use the IAM Policy Simulator to test policies that are attached to existing users, groups and roles.
- You can also use the simulator to test policies that are not attached yet to your resources by simply copying and executing them against the simulator
- The simulator can also be used to test policies attached to various AWS resources such as Amazon S3 buckets, Amazon EC2 instances, and so on.
- You can even use the simulator to test out real world scenarios by passing various context keys such as *IP addresses* or *usernames* that are passed alongside the conditions of a policy, and much more!

To get started with the IAM Policy Simulator, simply select this URL <https://policysim.aws.amazon.com/>. The policy simulator is a separate entity that runs outside your standard AWS Console. Use your standard AWS IAM credentials to log in to the policy simulator if asked:

1. Once logged in, you can use the simulator to test and validate your existing user, group and even role-based policies. To start with, let us test the policy we created in our earlier section of this chapter! To do so, from the **Users, Groups, and Roles** section, select the **Users** from the drop-down list. You should see a list of users present in your AWS account.

2. Select the appropriate **User** that was used to attach the policy. In my case, the username was `joe`. Once selected, you will be shown all the policies that the user is currently associated with, in this case, we should see the custom S3 access policy that we created in the earlier section as shown in the following screenshot:



3. Select the policy to view its details. You can even choose to modify and test the policy here using the inbuilt *policy editor*, however do note that changes made to the policy here are not reflected in the actual policy.
4. With the policy selected, we are now ready to test it using the **IAM Policy Simulator** section. Here's a snippet of the policy that we are going to test:

```
{
  "Sid": "VisualEditor0",
  "Effect": "Allow",
  "Action": [
    "s3:PutObject",
    "s3:GetObject",
    "s3:ListBucket",
    "s3:DeleteObject"
  ],
  "Resource": [
    "arn:aws:s3:::useast-prod-stuff-01",
    "arn:aws:s3:::useast-prod-stuff-01/Dummy"
  ],
  "Condition": {
    "IpAddress": {
      "aws:SourceIp": "10.0.0.0/24"
    },
    "ForAllValues:StringEqualsIfExists": {
      "aws:username": "joe"
    }
  }
}
```

```

    }
    . . . . .

    "Resource": "*",
    "Condition": {
      "IpAddress": {
        "aws:SourceIp": "10.0.0.0/24"
      },
      "ForAllValues:StringEqualsIfExists": {
        "aws:username": "joe"
      }
    }
  }
}

```

As per our policy, only the user `joe` has the `s3:PutObject`, `s3:GetObject`, `s3:ListBucket`, `s3:DeleteObject` rights to the `useast-prod-stuff-01` bucket and that too if Joe is accessing the bucket from his organization's internal network (`10.0.0.0/24`).

5. To test the same, from the **Select service** drop-down list, select **Amazon S3** option. Next, match either of the actions using the **Select actions** drop-down list. In this case, I've only selected the `s3:PutObject` action.
6. Once completed, from the **Global Settings** section, type in the adjacent values against the *condition keys* that appear in the policy. In this case, type in the **username** and the **sourceip** as depicted in the following screenshot:

Policy Simulator

Amazon S3 | 1 Action(s) sele... | Select All | Deselect All | Reset Contexts | Clear Results | **Run Simulation**

▼ Global Settings ⓘ

The following global AWS condition keys appear in the selected policies:

aws:username | joe

aws:sourceip | 10.0.0.45

Action Settings and Results [1 actions selected. 1 actions not simulated. 0 actions allowed. 0 actions denied.]

Service	Action	Resource Type	Simulation Resource	Permission
▼ Amazon S3	PutObject	object	object	Not simulated

Resource You can specify the resource and context keys used to simulate this action. By default the simulation resource is "".

object | **arn:aws:s3::useast-prod-stuff-01** | ☒ Include Resource Policy

7. Next, from the **Action Settings and Results** section, expand on the **Resource** and type in the ARN of the resource against which the policy needs to be tested. In this case, this has to be the ARN of the S3 bucket **arn:aws:s3:::useast-prod-stuff-01**. Once done, select the **Run Simulation** option.

With the simulation running, you should get either *allowed* or *denied* results based on the values you provide during the simulation. Feel free to change the *actions* as well as the *condition keys* and re-run the simulation. With each attempt, you can fine tune and troubleshoot your policy without having to actually pass any real requests to your resources.

You can also use the same policy simulator to test out new policies that are not yet attached to resources. To do so, you will first need to toggle from the current (default) mode of **Existing policies** to **New Policy** using the **Mode** option provided at the very top of the simulator.

Once the **New Policy** option is selected, you can use the **Policy Sandbox** to create new policies and test them out the same way we did a while back. Remember, however, that policies created or edited here are not reflected back in AWS IAM.

With this, we come towards the end of this section. In the next section, we will be looking at how you can leverage AWS Organizations to effectively manage multiple AWS accounts with relative ease.

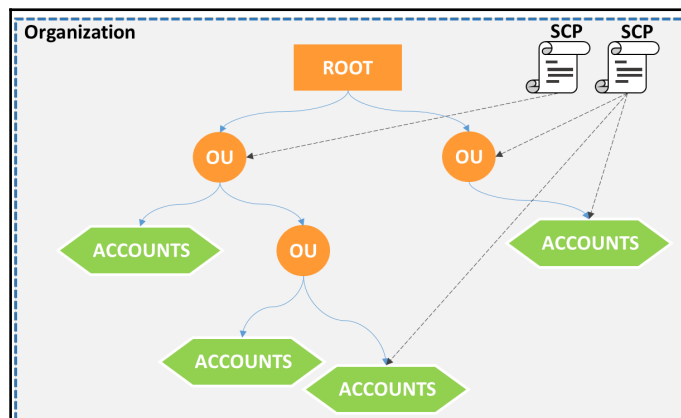
Introducing AWS Organizations

So far we have been working out of a single AWS account that we use for development, testing as well as for production purposes, but this isn't the case with many organizations who end up with multiple AWS accounts for a variety of purposes such as multiple environments, compliance issues, and so on. Each account gets governed and managed in its own way with no centralized ownership or control.

AWS Organizations is a simple service that allows you to consolidate and manage multiple such AWS accounts all under one roof. It enables you to group AWS accounts into one or more collective *organizations* that you can create and manage as a whole.

Here's a quick look at some of AWS Organizations key concepts and terminologies:

- **Organizations:** Organizations are consolidated views of all your AWS accounts in one place. Using organizations, you can centrally view and manage each of your AWS accounts under one roof. Organizations provide an additional functionality using which you can determine the type of organization you wish to create. There are two such feature sets, namely:
- **Consolidated billing:** A key feature provided by AWS Organizations is the ability to view and consolidate each AWS accounts billing under one organization. This feature is selected by default when you first create an organization and only provides you with the consolidated billing views. For leveraging all of the AWS Organizations advanced features, you will have to select the **All features** option.
- **All features:** This feature set provides the full functionality of AWS Organizations, including consolidated billing and many other features that provide you with better control over your individual accounts. Using this feature set, you can restrict certain AWS services from accounts; modify access roles, and much more.
- **Root:** The root is the primary container for all your individual accounts used within AWS. AWS Organizations automatically creates a default root element for you when you first create an organization. Any changes or policies applied at the root level propagate to its subsequent child elements as well.
- **Organizational Unit (OU):** OUs are containers for one or more AWS accounts. You can branch multiple OUs from a single OU as well, however the end of an OU is always an account. Here is a representational diagram depicting the interactions between an organization, the root element, OUs, and various AWS accounts:



- **Accounts:** Accounts are standard AWS accounts that contain your AWS resources. When creating an organization, AWS marks the account from where the organization gets created as the *master account*. Any additional accounts added later to this organization are termed as *member accounts*. The master account is also responsible for overseeing the consolidated billing and payments for the rest member accounts as well as useful for inviting other AWS accounts into the organization, creating OUs, managing policies, and so on.
- **Service Control Policy (SCP):** SCPs are essentially policies that are attached to roots, accounts or OUs for specifying services and actions that the particular account's or OU's user can use. For example, you can use an SCP on an account that is created with HIPAA compliance in mind and you want to restrict all users of this account to use only HIPAA compliant AWS services, and so on.



To know more about the HIPAA compliance and how it works along with AWS visit this URL to know more <https://aws.amazon.com/compliance/hipaa-compliance/>.

An important point to remember here is that SCPs only work when you have enabled the **All features** feature set while creating your organization.

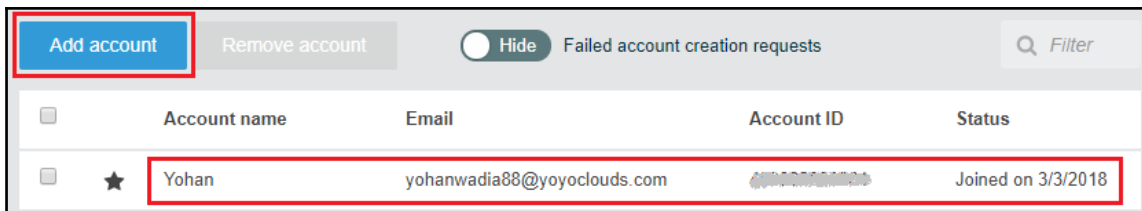
With this basic information in hand, let us look at how you can get started with AWS Organizations using a few simple steps.

Getting started with AWS Organizations

AWS Organizations can be set up using both the AWS Management Console as well as the AWS CLI. In this section, we will be exploring simple steps using which you can get started with your own organization in a matter of minutes:

1. From the AWS Management Console, filter out **AWS Organizations** using the **Filter** option or alternatively navigating to this URL <https://console.aws.amazon.com/organizations/>.
2. Since this is the first time we are setting up an AWS Organization here, this particular account will now be transformed into the master account. Any other AWS accounts added or created to this master account will be termed as member accounts. Click on **Create organization** to get started.

3. At the time of creating an organization, you can opt to select either to **Enable all features** or **Enable only Consolidated Billing** based on your requirements. For this scenario, select **Enable all features** and click on **Create organization** once completed.
4. With the organization created, you should see your existing account listed on the accounts page as shown in the following screenshot. Select the **Add account** option to add a new AWS account to our organization:



5. As mentioned before, AWS Organizations allows you to add existing AWS accounts into a new organization as well as create new accounts as a part of your master account itself. For this particular scenario, let us go ahead and create a new account called **sandbox**. Select the **Create account** option on the **Add account** page.
6. Provide a suitable name and email address for your new account in the **Full name** and **Email** fields respectively. The email that you provide has to be globally unique so provide an email address that has not been used so far with AWS here.
7. Next, in the **IAM role name** field, provide a suitable role name for your account. This role will enable you to access the new member account when signed in as an IAM user in the master account. Once completed, click on **Create** to complete the process.

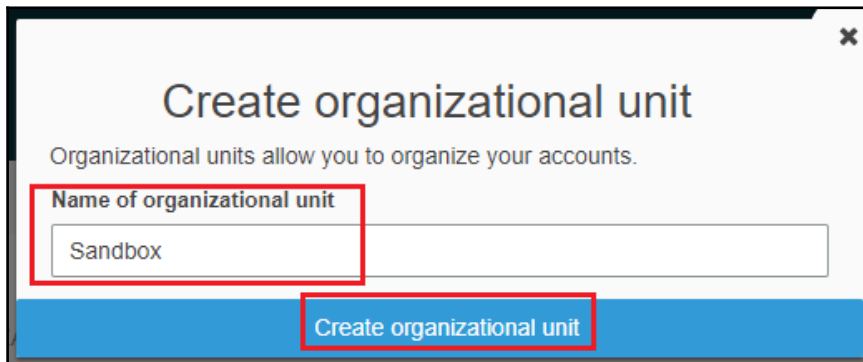
Here is a snippet of the IAM Role that is created by AWS. The role grants full access to all AWS services and resources present in the new account:

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

The account creation process takes a few minutes to complete. Once done, you should see a new account created with the name **Sandbox** and a new **Account ID** as well in the **Accounts** page of AWS Organizations. Select the sandbox account to view its associated ARN and ID in the adjoining pane.

With your account created, the next step involves creation of one or more OUs and moving the newly created account into an OU. For this particular scenario, let us go ahead and create an OU called **Sandbox**:

1. Select the **Organize accounts** tab from the AWS Organizations dashboard to view the current hierarchical setup of the organization. From the **Organizational Units** section, select the + **New Organization Unit** option to get started.
2. In the **Create organizational unit** dialog box, provide a suitable name for the new OU in the **Name of organization unit** field as shown in the following screenshot. Click on **Create organizational unit** once done:



3. With the OU created, the final step in the process is to move the account into the newly created OU. To do so, from the same **Organize accounts** page, select the **Sandbox** account and click on **Move**.
4. This will bring up a simple interface using which you will need to select the new OU to which you want to move the **Sandbox** account. Select the **Sandbox** OU and click on **Move** to complete the process.

With this step, you should have a new OU and a new account listed within it. You can use the same steps to create multiple OUs and accounts based on your requirements and needs. You can additionally provide restricted access to the services present in your new account by creating and assigning a new SCP to it as well:

1. To create a new SCP, select the **Policies** tab on the AWS Organization dashboard. You should see a default policy with the name `FullAWSAccess` already present there. This is a default policy that is created by AWS the first time you create an AWS Organization. The sandbox environment too is currently referenced to the same policy.
2. To create a new SCP, select the **Create policy** option. AWS Organization provides you with two options when it comes to creating new SCPs. You can choose to leverage the **Policy generator** to select specific services and actions from a list and build your custom policy or alternatively, choose to **Copy an existing SCP** and edit the same manually. For this case, select the **Policy generator** option.
3. Next, provide a **Policy name** and **Description** followed by selecting the **Effect** the policy should have on the applied AWS account.

AWS Organizations allows you to either whitelist (allow) or blacklist (deny) services based on your requirements. Blacklisting services will cause all services listed in the policy to be blocked by default, whereas whitelisting services will block all service APIs that are not listed in the policy. Let us go ahead and create a simple policy for our sandbox OU that allows EC2, S3, and RDS services while explicitly blocking AWS CloudTrail.

4. In the **Choose Overall Effect** section, select **Allow** to first create the whitelist of services. Use the **Statement builder** to select the correct service and its appropriate action as well. Once done, click on **Add statement** to add further services and actions as shown in the following screenshot:

Statement builder

Select Service ▼ Select action ▼ + Add statement

Service	Actions	Effect
Amazon EC2	*	Allow
Amazon S3	*	Allow
Amazon RDS	*	Allow
AWS CloudTrail	CreateTrail DeleteTrail	Deny

Cancel Create policy

5. Once you are done with adding the required statements, simply select the **Create policy** option to complete the process.
6. With the policy created, you can attach the same to an existing account, OU, or even root by simply selecting the policy from the **Policies** page and selecting either of the root, accounts, or organization units options as required. However, before you proceed further, you will first need to enable SCP policy types for your root account. To do so, select the **Root** option from the **Organize accounts** tab. In the adjoining details pane, select enable next to the **Service control policies** section.
7. With this completed, you can now proceed with attaching the newly created policy to an entity within your organization. Remember, by attaching the policy to the root domain, you effectively propagate the policy down to its members as well. This includes the sub OUs and accounts that you may have created. For this particular case however, select the **Accounts** option and click the **Attach** option below the listed **Sandbox** environment.

There you have it! A simple way using which you can create and manage your AWS accounts with utmost ease. Next, we will look at how you can achieve the same results by leveraging the AWS CLI as well.

Creating and managing AWS accounts, OUs and SCPs using the CLI has its own bit of advantages. For example, you can easily automate the entire account creation process and make it faster and easier to on-board new teams within your organization, and so on:

1. The first step involved in this exercise is to create the organization and set it to use all features, just as we performed with the AWS Management Console. To do so, type in the following command as shown:

```
# aws organizations create-organization --feature-set ALL
```



However this command may result in an error message for those of you who already have associated your AWS account with an organization.

You can alternatively pass the `CONSOLIDATED_BILLING` value for the `--feature-set` parameter depending on your organization's requirements.

2. With the organization in place, let us go ahead and create an AWS account for our organization, but before we do that, we need to execute just one command that will provide us with the organization's root's ID. The ID will be in the form of `r-<XY00>`. Make a note of the same for later steps:

```
# aws organizations list-roots
```

3. With the root ID noted type in the following command to create a new account. In this case, we are going to create a new account with the name of `prod`. Remember to substitute the `<EMAIL_ID>` field with a globally unique value:

```
# aws organizations create-account  
--email <EMAIL_ID>  
--account-name prod
```

Here is a snapshot of the command's output. Make a note of the new account's status ID in the format `car-<UNIQUE_ID>` as shown in the following screenshot:

```
yoyo@YoYoNux:~$  
yoyo@YoYoNux:~$ aws organizations create-account \  
> --email y... \  
> --account-name prod  
{  
  "CreateAccountStatus": {  
    "RequestedTimestamp": 1520255207.659,  
    "State": "IN PROGRESS",  
    "Id": "car-0f83b640207611e8bc03500c66ce5229",  
    "AccountName": "prod"  
  }  
}
```

4. You can use this status ID to check whether the account creation has completed successfully or not by typing in the following command. Replace the `car-<UNIQUE_ID>` with the value copied from the earlier step:

```
# aws organizations describe-create-account-status  
--create-account-request-id car-<UNIQUE_ID>
```



You can view the newly created accounts ID by using the following command:

```
aws organizations list-accounts
```

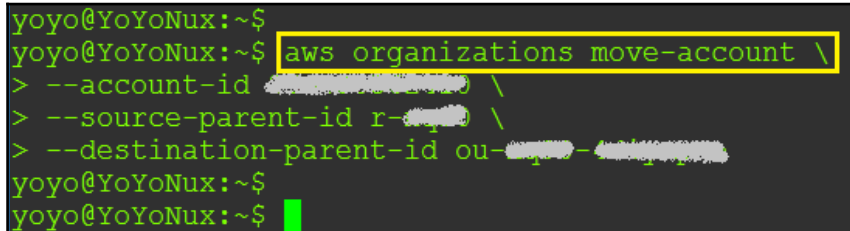
5. Once the new account is created, we can proceed to create a new OU and move the account over to the new OU. In this case, we are naming the new OU as `production`. Substitute the value of `r-<XY00>` with the root ID that we made a note of from our earlier steps:

```
# aws organizations create-organizational-unit  
--parent-id r-<XY00>  
--name production
```

The output of this command yields two important values, first is the ARN of the new OU and the second is the OU's ID which is in the form of `ou-<XY00>-<UNIQUE_ID>`. Make a note of the same for the next steps.

6. Now that the account and OU are created, we simply have to move the account into the new OU. To do so, type in the following command while substituting the correct values for the `account-id`, `parent-id`, and the `OU-id`:

```
# aws organizations move-account
--account-id <NEW_ACCOUNT_ID>
--source-parent-id r-<XY00>
--destination-parent-id ou-<XY00>-<UNIQUE_ID>
```



A terminal window screenshot showing the execution of the `aws organizations move-account` command. The prompt is `yoyo@YoYoNux:~$`. The command is entered as `aws organizations move-account \` and is highlighted with a yellow box. The subsequent lines show the command being completed with flags: `> --account-id [redacted] \`, `> --source-parent-id r-[redacted] \`, and `> --destination-parent-id ou-[redacted]-[redacted]`. The prompt returns to `yoyo@YoYoNux:~$` after the command is executed.

That's it! You have just created a brand new AWS account and moved it into a new OU as well with a few simple commands! It's that easy! However, you can take things a bit further by creating a new SCP and attaching the same to our newly created OU. Let us assume for a moment that the particular OU that we have created can only allow certain AWS services to run as a part of the production environment. To do so, we first need to create a new SCP as shown in the following code block:

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": [
      "ec2:*",
      "rds:*",
      "dynamodb:*"
    ],
    "Resource": "*"
  }]
}
```

In this case, the SCP is whitelisting EC2, RDS, and DynamoDB however feel free to modify to suit your own requirements:

1. Paste the SQP into a new file named as `policy.json` and save it. Next, type in the following command to create it:

```
# aws organizations create-policy
--content file://policy.json
--name AllowProductionServices
--type SERVICE_CONTROL_POLICY
--description "This policy allows only certain production services"
```

2. With the policy created, you should receive a policy ID in the form of `p-<UNIQUE_ID>`. Make a note of the same. Next, use the following command to attach the newly created policy to our OU:

```
aws organizations attach-policy
--policy-id p-<UNIQUE_ID>
--target-id ou-<XY00>-<UNIQUE_ID>
```

That's it! You have just successfully attached a new policy to your OU. You can use the same syntax to attach the policy directly to each individual account as well.

Planning your next steps

There are still plenty of things worth trying out when working with AWS Organizations. Here are a few recommendations for the same:

First up, as a best practice, you should always monitor your individual organizations and make sure that each change is tracked and reported. You can leverage both AWS CloudTrail as well as AWS CloudWatch events to accomplish the same. Monitoring the organizations is essential as it helps you to ensure that no unwanted changes affect the compliance of your accounts and environments. You can read up more on how you can leverage AWS CloudTrail and AWS CloudWatch events for monitoring organizations at https://docs.aws.amazon.com/organizations/latest/userguide/orgs_monitoring.html.

Next up, I would also recommend that you try out the AWS provided end-to-end account creation process here:

<https://aws.amazon.com/blogs/security/how-to-use-aws-organizations-to-automate-end-to-end-account-creation/>

The overall process for automating the account creation is very similar to the CLI commands that we executed a while back. The script accepts a few variables that are required to be defined first along with a few customizations of role names. Once done, you simply execute the shell script, which in turn calls a CloudFormation template to create and configure the new member account.

Summary

With this, we come toward the end of yet another chapter but before we move on to the next, let us quickly summarize all that we have learnt so far!

We first started with a brief recap of AWS IAM and its core building blocks followed by learning a bit about two really useful enhancements made to IAM in recent times. The first was a visual editor using which you can create customized and granular IAM policies with relative ease, followed by an IAM Policy Simulator tool that helps you to create and test your policies without affecting any running workloads on the cloud. Post this we also learnt about AWS Organizations and how you can leverage it for creating and managing multiple AWS accounts under one roof. We also saw how easily you can create and work with root, organizations, and accounts using both the AWS Organizations dashboard as well as the CLI.

In the next chapter, we will be learning and exploring how you can leverage the AWS Code Suite of services to build an entire end-to-end CICD pipeline with the utmost of ease.

7

Transforming Application Development Using the AWS Code Suite

In the previous chapter, we explored a few interesting and really useful enhancements made to the AWS **Identity and Access Management Service (IAM)** along with a quick deep dive into AWS Organizations as well.

In this chapter, we will be learning and exploring three extremely useful and powerful services provided by AWS that are specially catered toward enhancing a developer's experience with continuous code deployments: AWS CodeCommit, AWS CodeDeploy, and AWS CodePipeline!

Keeping this in mind, let's have a quick look at the various topics that we will be covering in this chapter:

- Introducing AWS CodeCommit, AWS CodeDeploy, and AWS CodePipeline along with each service's concepts and internal workings
- Creating your first CodeCommit repository and uploading an application to it
- Running basic Git commands against your new code repository
- Configuring the CodeDeploy agent on an EC2 instance
- Leveraging the AppSpec file for configuring application life cycle deployment
- Creating your own continuous delivery system using CodePipeline

So without any further ado, let's get started right away!

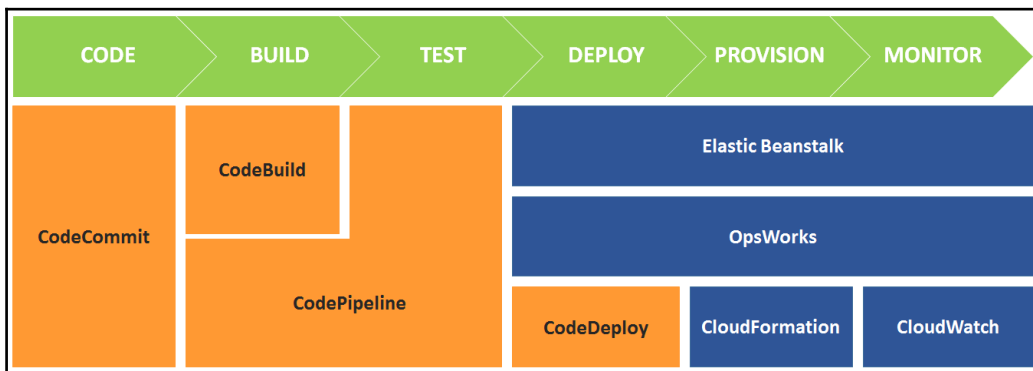
Understanding the AWS Code Suite

Besides providing a plethora of infrastructure-related services, AWS also provides a few services that are designed to help developers quickly design, develop, build, and deploy their applications on the AWS cloud platform. In this section, we will have a quick look at these services and how you can leverage them together to build your very own continuous integration and delivery pipelines:

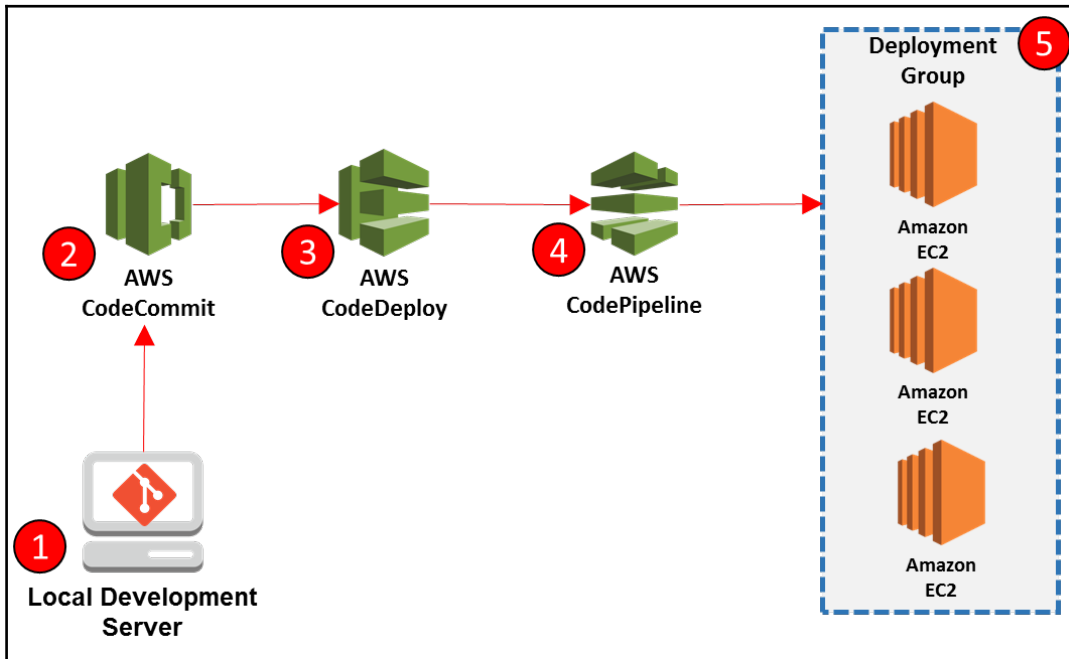
- **AWS CodeCommit:** An important starting point for any CI/ CD pipeline is a simple yet functional source control repository. Traditionally, this would be set up on one or more physical servers in the form a Git or SVN repository that developers would use to push their code and updates to; however, maintaining such code repositories and scaling them was always going to be a challenge. That's where AWS CodeCommit comes into play! AWS CodeCommit is a managed source control service that enables developers to securely store their code on the AWS cloud. It offers many of the features that you would require and use while working with different source control repositories, such as branching, commits, rollbacks, and much more.
- **AWS CodeBuild:** AWS CodeBuild is a code build service that developers can leverage to automate their source code compilations, tests, executions, and code packaging for deployments. Similar to its other counterpart services in the Code Suite, CodeBuild too is managed completely by AWS, thus eliminating any unnecessary administrative overheads, such as patching or scaling the code build software. CodeBuild is highly extensible and it also easily integrates with your existing CI/CD workflows as well.

- **AWS CodeDeploy:** With your application code stored securely and compiled, the final step requires the code to be deployed across your fleet of EC2 instances. This can be easily achieved with the help of our next Code Suite service, called AWS CodeDeploy. Using CodeDeploy, a developer can automate code deployments to any environment that runs off of either EC2 instances as well as servers that are running in an on-premise datacenter. CodeDeploy essentially eliminates deployment complexities by allowing you to automate the delivery of your code across thousands of instances without having to undergo any major downtimes.
- **AWS CodePipeline:** AWS CodePipeline is a full fledged CI/CD service provided by AWS that developers can leverage to build end-to-end CI/CD pipelines either by using the AWS Code Suite of services or even with other popular third-party tools, such as GitHub, Jenkins, and so on. Using CodePipeline, you can also create and define custom software release models using which your application gets updated with the latest set of updates, tested, and packaged as well for the next iterative set of deployments.

Here is a brief representation of how these services map together collaboratively to create a comprehensive CI/CD pipeline:



For the purpose of this chapter, however, we will be working with only three AWS Code Suite services, namely AWS **CodeCommit**, AWS **CodeDeploy**, and AWS **CodePipeline**. The chapter will showcase how these three services can be leveraged together to build your very own CI/CD pipelines for our sample WordPress application. Here is a high-level depiction of our overall use case:



With the basic understanding of the Code Suite services out of the way, let's learn a bit more about AWS CodeCommit and how you can leverage it as your very own source code repository!

Getting Started with AWS CodeCommit

As discussed earlier, AWS CodeCommit is a secure and highly scalable source control service which allows you to create multiple private Git repositories without having to bother about any of the underlying management overheads. You can use it to store anything, from code, to application binaries, to even code packages, all using the standard Git-like functionality. This makes CodeCommit extremely easy to work with even if you have not used it before. Here is the gist of some of the most commonly used Git commands and how you can leverage them with CodeCommit:

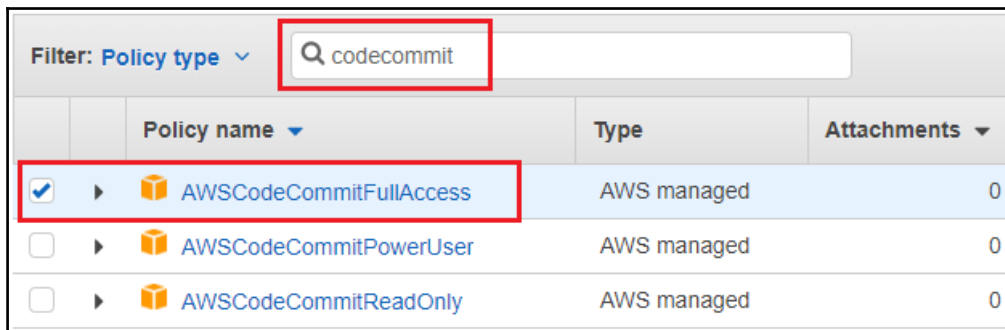
- `git clone`: Used to clone and connect the AWS CodeCommit repository over to your local development server.
- `git add`: Once the repository is cloned locally, you can use it to add, edit, or delete files as you see fit. Once done, use the `git add` command to stage the modifications in your local Git repository.
- `git commit`: Used to commit the modifications made to the files to the local Git repository.
- `git push`: Used to push the committed files and changes over to the AWS CodeCommit repository.
- `git pull`: Used to ensure that the files you are working on are synced and are of the latest version from the AWS CodeCommit repository.

In this section, we will be looking at a few simple steps to enable you to create your very own source code repository using the AWS Management Console. However, before we move on to that, it is important to understand some of the different connections that you can use to connect to your CodeCommit repository. This can vary based on your development environments as well as security requirements:

- **Using the HTTPS connections:** Configuring Git credentials using HTTPS connections is by far the simplest and most widely used method for connecting to your Git repository. With this set up, you simply generate a static username and password using AWS IAM. Once the credentials are created, you can then use them with Git and any third-party tool, such as an IDE, for authentication.
- **Using the SSH connections:** In this case, a user will be required to create public and private key files on your local development server that Git and AWS CodeCommit can use for SSH authentication. The public key generated in this process gets associated with your IAM user, whereas the private key remains on the local development server. The generation of the keys varies from operating system to operating system and can be a tedious process at times to manage.

For this section, however, we will be leveraging the SSH connections method itself for connecting to our AWS CodeCommit repository:

1. To get started, first log in to your AWS Management Console and filter the IAM service using the **Filter** option provided. Alternatively, you can also select URL `https://console.aws.amazon.com/iam/` to view the IAM dashboard.
2. Here, we will start off by creating a dedicated user that will have full management rights to our CodeCommit repository. Select the **Users** option from the IAM dashboard's navigation pane to bring up the list of currently created IAM users.
3. Next, select the **Add user** option. This will bring up the **Add user** page where you can provide a suitable **User name** as well as opt for the user's **Access type**. In this case, the CodeCommit user will only require **Programmatic access**. Click **Next** to proceed.
4. Moving on, in the **Permissions** page, we are required to filter and attach the `AWSCodeCommitFullAccess` policy to our newly created user. To do so, select the **Attach existing policies directly** option and select the `AWSCodeCommitFullAccess` policy, as shown in the following screenshot. Alternatively, you can also provide a customized access policy here based on your requirements:



5. Complete the user creation process by reviewing the changes and making a note of the user's new access and secret keys as well.

At this point, with your CodeCommit IAM user created, we now move on to the next part of this section where we create and configure a set of public and private keys for the IAM user, using a simple Linux-based development server. Follow URL <http://docs.aws.amazon.com/codecommit/latest/userguide/setting-up-ssh-windows.html> if you are using a Windows operating system as your development server:

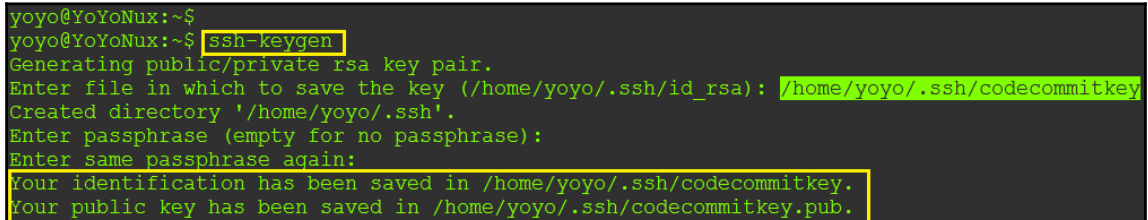
1. Log in to your development server and run the following command to generate the new set of keys:

```
# ssh-keygen
```

2. When prompted, save the keys in the following directory structure:

```
/home/<USER_NAME>/.ssh/<KEY_NAME>
```

Make a note of the public and private keys' locations, as depicted in the following screenshot:



```
yoyo@YoYoNux:~$  
yoyo@YoYoNux:~$ ssh-keygen  
Generating public/private rsa key pair.  
Enter file in which to save the key (/home/yoyo/.ssh/id_rsa): /home/yoyo/.ssh/codecommitkey  
Created directory '/home/yoyo/.ssh'.  
Enter passphrase (empty for no passphrase):  
Enter same passphrase again:  
Your identification has been saved in /home/yoyo/.ssh/codecommitkey.  
Your public key has been saved in /home/yoyo/.ssh/codecommitkey.pub.
```

3. Next, display and copy the public key's contents, using the following command. Note the public key will be saved in the file with a .pub extension:

```
# cat /home/<USER_NAME>/.ssh/<KEY_NAME>
```

4. Log in to your IAM dashboard once again and select the newly created user from the **Users** page. Select the user's **Security Credentials** tab. Here, under the **SSH keys for AWS CodeCommit** section, click on **Upload SSH public key** to paste the entire copied text from the earlier step.

- Once completed, you should now see a unique key auto-generated under the **SSH key ID** column, as shown in the following screenshot. Copy this SSH key ID as we will be requiring it in the next steps:

SSH keys for AWS CodeCommit			
Upload SSH public key			
SSH key ID		Uploaded	Status
APKAI2HR7DUK2CPNFLRA	Show SSH key	2017-12-05 10:04 UTC+0100	Active Make inactive

- With the public key uploaded to IAM and the new SSH key ID generated, the final step is to create a simple `config` file in your local development server with the following contents pasted into it:

```
# vi ~/.ssh/config
##### SUBSTITUTE THE <VALUES> WITH YOUR ACTUAL ONES #####
Host git-codecommit.*.amazonaws.com
  User <SSH_KEY_ID>
  IdentityFile ~/.ssh/<PRIVATE_KEY_FILENAME>
```

- Save the file once done. Remember to modify the permissions of your `config` file before moving on to the verification step:

```
# chmod 600 config
```

- To verify the connectivity, simply use the following command to SSH to the AWS CodeCommit endpoint. Since this will be a first connect, you will be prompted to verify the connection for authenticity. Type in `yes` when prompted:

```
# ssh git-codecommit.us-east-1.amazonaws.com
```



The endpoint you use will be specific to the AWS region that you operate out of. You can view the list of region-specific CodeCommit URLs along with the availability of the CodeCommit service at <http://docs.aws.amazon.com/codecommit/latest/userguide/regions.html>.

With this step, we have successfully validated and connected our development server with the AWS CodeCommit service! But where is our CodeCommit repository?

1. To create the repository, log in to the AWS CodeCommit service using URL `https://console.aws.amazon.com/codecommit`. Remember to change the **Region** based on what you selected during the key verification state.
2. Since this is our first time working with CodeCommit, select the **Get Started** option to begin with. This will display the **Create repository** page, as shown in the following screenshot:

3. Provide a suitable **Repository name** and an optional **Description**. Click on **Create repository** once done.
4. You can additionally configure notifications for specific **Event types**, such as pull requests and commits made to your repo in the **Configure email notifications** page. Simply select an existing **SNS topic** or opt to **Create a new topic** based on your requirements. Once done, click on **Save** to complete the repository creation process.

With the repository created, you can now use the development server and connect to it using a simple `git clone` command. You can obtain your repository's connection URL anytime by simply selecting the **Connect** option present on the **Code** page:

```
# git clone
https://git-codecommit.us-east-1.amazonaws.com/v1/repos/<YOUR_CODECOMMIT_REPO>
```

Here's a snapshot of the first `git clone` command output:

```
yoyo@YoYoNux:~$  
yoyo@YoYoNux:~$ git clone ssh://git-codecommit.us-east-1.amazonaws.com/v1/repos  
/wordpress-repo-01  
Cloning into 'wordpress-repo-01'...  
Warning: Permanently added the RSA host key for IP address '54.239.20.155' to t  
he list of known hosts.  
warning: You appear to have cloned an empty repository.  
Checking connectivity... done.  
yoyo@YoYoNux:~$  
yoyo@YoYoNux:~$
```

Since the repository is empty, the cloning process simply creates a folder with your repository's name on your development server. You can now use this folder as a code source control by simply adding your program files, binaries, and other application-specific data to it.

In the next section, we will be using this repository as our WordPress application source control and explore a few simple commands and features that are provided as a part of AWS CodeCommit.

Working with branches, commits, and triggers

With your CodeCommit repo created, it's now time to go ahead and use this repository as our source control repository. To do so, we will first push a standard WordPress application from our local development server to the AWS CodeCommit repository using simple Git commands and later run a few more Git commands as well as AWS actions to branch and commit our code.



You can obtain a WordPress application ZIP file by downloading it from here:

<https://github.com/WordPress/WordPress>

To begin with, first we will need to copy the WordPress application contents over to our local repository that was cloned earlier:

1. Once the WordPress application is downloaded to your local development server, simply unzip and copy its contents over to the local repository. Your local repository should now show a folder structure similar to the following screenshot:

```
yoyo@YoYoNux:~/wordpress-repo-01$ tree -L 1
.
├── index.php
├── license.txt
├── readme.html
├── wp-activate.php
├── wp-admin
├── wp-blog-header.php
├── wp-comments-post.php
├── wp-config-sample.php
├── wp-content
├── wp-cron.php
├── wp-includes
├── wp-links-opml.php
├── wp-load.php
├── wp-login.php
├── wp-mail.php
├── wp-settings.php
├── wp-signup.php
├── wp-trackback.php
└── xmlrpc.php
```

2. With the code in place, simply use the following Git commands to commit and push the code over to your AWS CodeCommit repository. First up, stage the files using the `git add` command:

```
# git add *
```

3. Next, commit the changes using the `git commit` command:

```
# git commit -m "First Commit!!"
```

4. And, finally, push the commit over to the AWS CodeCommit repository. Here, the keyword `origin` is the default remote name used by Git for your AWS CodeCommit repository, whereas `master` is the default branch name:

```
# git push -u origin master
```

5. You should see the code get uploaded to your AWS CodeCommit repository, as shown in the following image. You can cross-verify this by refreshing and checking the **Code** page on your AWS CodeCommit dashboard as well:

```
yoyo@YoYoNux:~/wordpress-repo-01$
yoyo@YoYoNux:~/wordpress-repo-01$ git push -u origin master
Counting objects: 1949, done.
Compressing objects: 100% (1921/1921), done.
Writing objects: 100% (1949/1949), 12.90 MiB | 471.00 KiB/s, done.0.00 KiB/s
Total 1949 (delta 248), reused 0 (delta 0)
To ssh://git-codecommit.us-east-1.amazonaws.com/v1/repos/wordpress-repo-01
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
yoyo@YoYoNux:~/wordpress-repo-01$
yoyo@YoYoNux:~/wordpress-repo-01$
```

Similarly, you and your fellow developers can edit and commit the code back to the AWS CodeCommit repository. You can also create multiple branches of your repository so that developers can work independently on the code without affecting the `master` branch. Once the features are all thoroughly tested and verified, the individual developer branches can be merged into a more stable `master` branch of the software.

Creating a branch in CodeCommit is an extremely easy process! You can use the CodeCommit dashboard, the Git command line, or even the AWS CLI to create one of your own:

1. To create a branch using the AWS CodeCommit dashboard, simply select the **Branches** option from the navigation pane.
2. Next, select the **Create branch** option to bring up the **Create** branch page. Here, provide a suitable **Branch name** and also select where you would like this new branch to **Branch from**. In this case, since you only have the `master` branch created, you can select that for now. Click on **Create** once done.
3. You can also use the Git command line itself to achieve the same result. In this case, from the development server, type in the following command to create a new branch:

```
# git checkout -b <NEW_BRANCH_NAME>
```

```
yoyo@YoYoNux:~/wordpress-repo-01$
yoyo@YoYoNux:~/wordpress-repo-01$ git checkout -b dev-cli-09-12-2017
Switched to a new branch 'dev-cli-09-12-2017'
yoyo@YoYoNux:~/wordpress-repo-01$
```

With the new branch created, you can also use the **Compare** functionality provided by CodeCommit to compare the changes made to the branch against another branch. To do so, we first need to perform some changes in the application so that it can get reflected as a change.

Without changing the current branch of the repository, simply update any one of the WordPress files by adding or removing a comment. In my case, I simply made a few comment changes in the WordPress application's `index.php` file; however, feel free to modify any other file as you see fit. Once the changes are made, we once again need to stage, commit, and push the changes over to the new branch of our repository:

1. Stage the changes by using the `git add` command. You can either add all the files for staging by using `*` or even specify the filename you wish to stage as well:

```
# git add *
```

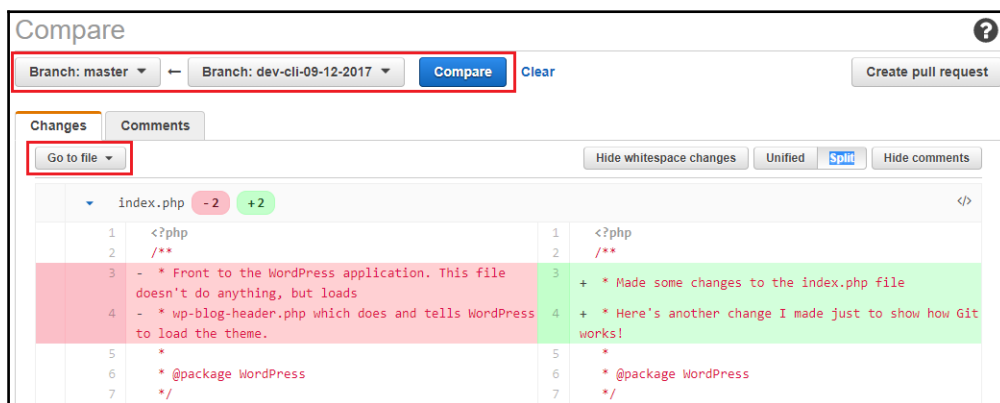
2. Next, commit the changes using the `git commit` command:

```
# git commit -m "<SOME_NEW_COMMIT_MESSAGE>"
```

3. And, finally, push the changes over to the branch using the following command:

```
# git push origin <NEW_BRANCH_NAME>
```

4. With the changes pushed, use the **Compare** option provided under the **Commits** section in the CodeCommit dashboard. Here, select the **master** as the *source* branch and the branch that you created using the Git command line as the **Destination** branch. Click on **Compare** once done. You should see the changes compared, as shown in the following screenshot:



You can use the **Go to file** drop-down list to toggle between different files, if you have made changes in them. Alternatively, you can also use the **Unified** and **Split** views to change the visual comparison as you see fit.

CodeCommit also provides an additional feature called **triggers** that you can use to either send notifications to or run some other external code build or process. You can assign up to 10 triggers per repository that you create, however, at the time of writing this book, CodeCommit only supports AWS SNS and AWS Lambda as its trigger mechanisms:

1. To create a simple trigger, using the CodeCommit dashboard, select the **Settings** tab from the navigation pane. Here, select the **Triggers** tab to create as well as view the list of existing triggers, if any.
2. Select the **Create trigger** option to bring up the **Create trigger** page. Here, you can configure triggers in response to certain repository events, such as **Push to an existing branch**, **Create branch or tag**, **Delete branch or tag**, or **All repository events**.
3. Provide a **Trigger name** and select the appropriate **Events** and **Branch name** that you wish to associate the trigger with. Once done, you can configure the trigger to either use an existing SNS topic or a Lambda function as its **Service**. You can even test the functioning of the trigger by selecting the **Test trigger** option. This will simulate a trigger based on the *event* that you would have selected earlier.

In this way, you can configure triggers for sending notifications to your developers as well as trigger-specific Lambda functions based on your repository's requirements.

Introducing AWS CodeDeploy

With CodeCommit configured and ready to use for our WordPress application, we can now move on to yet another Code Suite service that can actually be used to deploy the code across thousands of EC2 instances! Here's introducing AWS CodeDeploy!

AWS CodeDeploy is basically a deployment service that allows you to automate the deployment of your applications to Amazon EC2 instances, Lambda functions, or even to on-premise instances. There is no limit to what an AWS CodeDeploy service can deploy. You can use it for deploying virtually anything from code, packages, binaries, scripts, files, and so on. As of writing this book, CodeDeploy only supports GitHub repositories and Amazon S3 buckets as the default application content repositories. Yes, you heard it right, CodeDeploy does not support CodeCommit as a repository source as of now.

Besides the automation, CodeDeploy also provides you with the following set of useful benefits. It allows you to:

- Quickly create new prototype software and deploy at scale without manual interventions
- Easily update to your application code without any downtime
- Rollback deployments in case of any errors
- Scale your deployment from one to a thousand instances, all without disrupting to your existing applications

In this section of the chapter, we will be looking at how to set up CodeDeploy for our own application deployments, but before we get into that, here's a quick look at some of CodeDeploy's essential concepts.

Concepts and terminologies

CodeDeploy essentially comprises two main configurable sections that can be broadly classified as deployments and applications. Here's a look at each of these concepts:

- **Applications:** Applications here imply simple names that are used by CodeDeploy to identify individual application codes targeted for specific deployments. An application can be deployed either on an EC2 instance, an on-premise instance, as well as on a serverless compute platform, such as AWS Lambda.
- **Deployments:** Deployments are a collection of deployment configurations and deployment types, including:
 - **Deployment configurations:** Deployment configurations are a set of simple rules that determine how fast an application will be deployed and the success or failure conditions for that particular deployment. For example, for an EC2 deployment, the configuration rules can dictate the required minimum number of healthy instances, whereas with a Lambda function deployment, these rules can be used to specify how the traffic is routed to the functions during a deployment.
 - **Deployment group:** This is a group of EC2 or on-premise instances that are either standalone or a part of an auto-scaling group. Since AWS Lambda is a managed service, it does not provide any deployment groups.

- **Deployment types:** Deployment types indicate the type of method used to get the latest version of your application deployed on a particular deployment group. There are two deployment types supported:
 - **In-place deployments:** In this case, the application running on each EC2 instance is stopped, updated, started, and verified. This form of deployment is only supported for EC2 and on-premise instances.
 - **Blue/green deployments:** In this scenario, the underlying instances are replaced by newer instances with the updated piece of code. The instances are registered to an **Elastic Load Balancer (ELB)** that routes traffic to the newer instances while the older instances can then be terminated. With the serverless platform, the traffic here too is shifted automatically by AWS from the current Lambda functions to the current updated ones. Note that all Lambda deployments are in fact blue/green deployments only.
- **CodeDeploy agent:** CodeDeploy agent is a simple software package that gets installed on either an EC2 or an on-premise instance and is used by CodeDeploy for setting up and working with application deployments. Once the agent is installed on an instance, an associated configuration file is created. This file contains application-specific directory paths and other settings that CodeDeploy uses to interact with the instances. The file is a simple YAML file and can be located in the following directories based on the instance's operating system:
 - **Amazon Linux, Ubuntu, RedHat Enterprise Linux:**
`/etc/codedeploy-agent/conf/codedeployagent.yml`
 - **Windows Servers:** `C:\ProgramData\AmazonCodeDeploy\conf.yml`
- **Application specification files:** Application specification files, or AppSpec files, are used to define and manage individual deployments as a series of life cycle event hooks. Each hook itself can be another file, such as a simple script to start or stop services, install dependencies, and so on. AppSpec files are supported in both JSON as well as YAML formats. At the time of the deployment, the AWS CodeDeploy agent looks up the name of the current event in the hooks section of the AppSpec file. If an event is found, the agent retrieves the list of scripts to execute and runs them sequentially in the order in which they were written in the AppSpec file.

With the basics out of the way, let's quickly look at how we can set up an EC2 instance to be used with CodeDeploy.

Installing and configuring the CodeDeploy agent

Before we begin with the actual launch of our EC2 instance with the CodeDeploy agent installed on it, we need to set up an EC2 instance profile as well as an instance role that will grant our EC2 instances the necessary permissions to interact with both CodeCommit as well as with CodeDeploy:

1. To get started, first log in to the AWS Management Console and select the **IAM** service from the services **Filter**. Alternatively, you can launch the IAM dashboard by selecting URL <https://console.aws.amazon.com/iam/>.
2. From the IAM dashboard, select the **Policies** option from the navigation pane to bring up the IAM **Policies** page. Here, click on **Create policy** to get started.
3. In the **Create policy** page, select the **JSON** tab and paste the following lines of the policy document:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ec2:Describe*",
        "sns:*",
        "codecommit:*",
        "codedeploy:*",
        "codepipeline:*",
        "codecommit:GetBranch",
        "codecommit:GetCommit",
        "codecommit:UploadArchive",
        "codecommit:GetUploadArchiveStatus",
        "codecommit:CancelUploadArchive",
        "s3:*"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

The policy document essentially provides the EC2 instance with the required set of permissions to interact with the likes of AWS services, such as CodeDeploy, CodeCommit, and CodePipeline.

4. Click on **Review policy** once done. In the final **Review policy** page, provide a suitable **Name** for the policy and click **Create policy** to complete the process.
5. With the policy created, we now simply assign this to a new IAM Role. To do so, select the **Roles** option from the navigation pane to bring up the IAM **Roles** page.
6. Click on **Create role** to start the wizard. From the **Select type of trusted entity** section, make sure you select **AWS service** and filter out **EC2** from there. Click on **Next: Permissions** to proceed.
7. In the **Attach permissions policy** page, filter the earlier created policy and attach it to our new role, as depicted in the following screenshot:

The screenshot shows the 'Attach permissions policies' interface. At the top, there are buttons for 'Create policy' and 'Refresh'. Below them is a filter section with 'Filter: Policy type' and a search box containing 'IAM-'. To the right of the search box, it says 'Showing 1 result'. Below this is a table with columns: 'Policy name', 'Attachments', and 'Description'. The table has one row with the policy name 'IAM-Instance-Profile-CodeDeploy', 0 attachments, and the description 'Instance Profile for CodeDeploy'. A checkbox in the first column of this row is checked. At the bottom of the table, there are buttons for 'Cancel', 'Previous', and 'Next: Review'. The 'Next: Review' button is highlighted with a red box.

8. Finally, at the **Review** page, provide your role with a suitable **Role Name** and click on **Create role** to complete the process.
9. Before launching your EC2 instance with this newly created Role, ensure that the role's **Trust Relationship** has the following set of AWS services added in its policy document:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "codecommit.us-east-1.amazonaws.com",
          "ec2.amazonaws.com",
```

```

        "codedeploy.us-east-1.amazonaws.com",
        "codepipeline.us-east-1.amazonaws.com"
    ]
},
"Action": "sts:AssumeRole"
}
]
}

```

With this step completed, we are now ready to launch a simple EC2 instance and assign the newly created role:

1. From the EC2 Management Console, select the **Launch Instance** option to get started. For this particular use case, I've opted to go for the standard **Amazon Linux** AMI (amzn-ami-hvm-2017.09.1.20171120-x86_64-gp2 - ami-55ef662f); however, you can very well opt for a different Linux OS distribution as per your requirements.
2. Select an appropriate **Instance type** for hosting our simple WordPress application. For now, I've selected the **t2.micro** instance type itself.
3. Next, in the **Configure Instance Details** page, select the appropriate **Network**, **Subnet**, and **IAM Role** for our new EC2 instance. Paste the following set of lines as **User data** under the **Advanced Details** section, as shown in the following code. This simple user data script will copy and install the CodeDeploy agent along with a few other essential dependencies. You can find the complete copy of the following code at <https://github.com/yoyoclouds/Administering-AWS-Volume2>:

```

#!/bin/bash
yum -y update
yum install -y ruby
yum install -y aws-cli
cd /home/ec2-user
aws s3 cp s3://aws-codedeploy-us-east-1/latest/install . --region
us-east-1
chmod +x ./install
./install auto

```



Remember to change the `region` parameter as per your current operational region value.

4. Once the required storage is assigned to the instance, move on and assign a few essential tags for our EC2 instance. These tags will be used later in CodeDeploy to reference our EC2 instances, so make a note of the same.
5. Finally, create a new security group and make sure that the ports 22 (SSH) and 80 (HTTP) are open for internet traffic.
6. Review the settings of your instance and launch it. Additionally, remember to associate your instance with a key pair as well before you launch it, as it can be useful to verify or troubleshoot the AWS CodeDeploy agent.

With this, you now have successfully launched and set up a CodeDeploy agent on an EC2 instance. In the next section of this chapter, we will look at how you can take this installation further by configuring the AppSpec file for the final CodeDeploy deployment.

Setting up the AppSpec file

As mentioned earlier, the AppSpec, or the application specifications file, is basically a YAML or JSON backed file used to define life cycle hooks for a particular deployment. In this case, with our EC2 instance prepped and ready with the CodeDeploy agent, we still need the AppSpec file to define a set of dependencies that will essentially install the necessary packages on the EC2 instance, start or stop the services, change permissions, and much more.

The AppSpec file comprises the following sections:

- **Version:** The version of the AppSpec file. Currently, the version number supported with CodeDeploy is 0.0. Do not change this value.
- **OS:** Specifies the operating system of the underlying EC2/on-premise instance.
- **Files:** Specifies the files that need to be copied on the instance at the time of the deployment process. You can additionally specify a source and a destination folder as well for your applications here.
- **Hooks:** Hooks essentially specify when a particular deployment life cycle has to be triggered. There are four main types of hooks: `BeforeInstall`, `AfterInstall`, `ApplicationStart`, and `ApplicationStop`. Each hook further requires additional parameters, such as the location of the scripts to execute, a timeout value, and how the scripts should be run.

Here is a representation of a simple AppSpec file for our WordPress deployment. Since we are using the Amazon Linux-backed instance, the `os` value is set to `linux` along with other essential parameters, such as `files` and `hooks`. Note in our case, we are configuring the WordPress application files to be copied from their default location over to `/var/www/html/WordPress` as well:

```
version: 0.0
os: linux
files:
  - source: /
    destination: /var/www/html/WordPress
hooks:
  BeforeInstall:
    - location: scripts/install_dependencies.sh
      timeout: 300
      runas: root
  AfterInstall:
    - location: scripts/change_permissions.sh
      timeout: 300
      runas: root
  ApplicationStart:
    - location: scripts/start_server.sh
      timeout: 300
      runas: root
  ApplicationStop:
    - location: scripts/stop_server.sh
      timeout: 300
      runas: root
```

To create this file, login to your development server and not your CodeDeploy instance. Once logged into the development server, open the WordPress application directory that we used during the CodeCommit sections of this chapter. This is the same directory that we used to sync with our master CodeCommit repository as well:

1. Here, at the application's root directory, create a blank `appspec.yml`, and copy-paste the `appspec` contents explained earlier:

```
# vi appspec.yml
```

2. With the AppSpec file created, we now move on to create the individual files for the life cycle hooks. To do so, create a folder named `scripts` within the WordPress application directory and create each of these individual shell scripts within it:

```
# mkdir scripts
```

3. Create the `install_dependencies.sh` script that will essentially install the necessary packages required to run WordPress on an instance:

```
# vi scripts/install_dependencies.sh
#!/bin/bash
sudo yum install -y httpd mysql mysql-server php
sudo yum install -y php-mysql
```

4. Similarly, create the `change_permissions.sh` script that modifies the permissions of the files present in the scripts folder to *executable*:

```
# vi scripts/change_permissions.sh
#!/bin/bash
chmod -R 755 /var/www/html/WordPress
```

5. And, finally, create the `start_server.sh` and the `stop_server.sh` scripts that will start and stop the `httpd` and `mysql` services on the deployment EC2 instances:

```
# vi scripts/start_server.sh
#!/bin/bash
service httpd start
service mysqld start

# vi scripts/stop_server.sh
#!/bin/bash
isExistApp=`pgrep httpd`
if [[ -n $isExistApp ]]; then
    service httpd stop
fi
isExistApp=`pgrep mysqld`
if [[ -n $isExistApp ]]; then
    service mysqld stop
fi
```

Got this far? Awesome! We are almost done with the AppSpec files, with just one small step left: uploading these changes to our CodeCommit repository!

To do so, run the following set of commands from the deployment server:

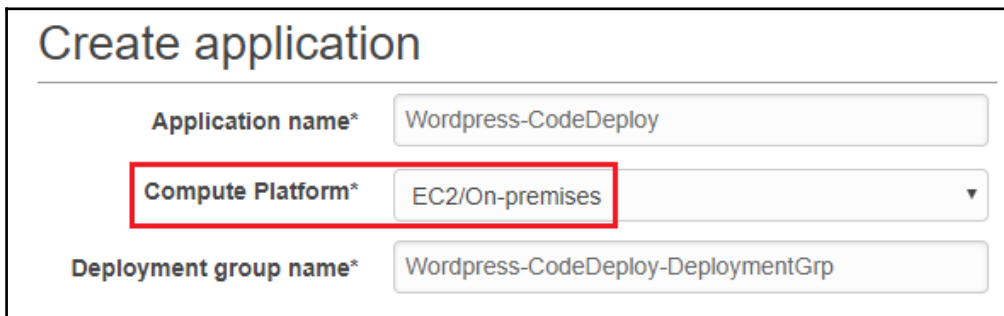
```
# git add *
# git commit -m "Added scripts directory with AppSpec file!"
# git push -u origin <NEW_BRANCH_NAME>
```

Et voila! The WordPress application and our deployment scripts are all uploaded to our CodeCommit branch and ready for deployment! In the next section, we will create and configure an application and deployment group for our CodeDeploy.

Creating a CodeDeploy application and deployment group

With our AppSpec file and scripts in place and the EC2 instance set up with the CodeDeploy agent as well, the final thing left is to configure AWS CodeDeploy and tie these things together:

1. Start off by logging into the AWS CodeDeploy service by selecting URL `https://console.aws.amazon.com/codedeploy/home`.
2. Since this is the first time we are using CodeDeploy, select the **Get Started Now** option. Here, you will be prompted to select either a **Sample deployment** that deploys a sample application on the EC2 instances or, alternatively, go for a **Custom deployment** if you have your code and EC2 instances up and running. In this case, we will select the **Custom deployment** option itself.
3. In the **Create application** page, start off by providing a suitable **Application name**, **Compute Platform**, and a **Deployment group name**. Remember to select the **EC2/On-premises** option from the **Compute Platform**, as shown in the following screenshot:



The screenshot shows the 'Create application' form in the AWS CodeDeploy console. It contains three input fields: 'Application name*' with the value 'Wordpress-CodeDeploy', 'Compute Platform*' with a dropdown menu showing 'EC2/On-premises' (highlighted by a red rectangle), and 'Deployment group name*' with the value 'Wordpress-CodeDeploy-DeploymentGrp'.

4. Next, in the **Deployment type** section, choose the **In-place deployment** option for now. This will enable CodeDeploy to update the existing instances with the revised set of application code with some amount of downtime.

5. In the **Environment configuration** section, you can specify any combination of **Auto Scaling groups**, **Amazon EC2 instances**, and **On-premises instances** to add instances to this deployment group. Since we have created an EC2 instance in our earlier steps with the CodeDeploy agent installed in it, select the **Amazon EC2 instances** tab. From the **Tag group** drop-down, select the instance's **Key** and **Value**, as shown.

Note that these are the same tags that you would have configured to your instance before its launch in our earlier sections:

Environment configuration

Specify any combination of Auto Scaling groups, Amazon EC2 instances, and on-premises instances to add instances to this deployment group.

Auto Scaling groups **Amazon EC2 instances** On-premises instances

Tag group 1

	Key	Value	Instances	
1	name	wordpress-CodeDeploy-De	1	✕
2				✕

Matching instances

« < 1 to 1 of 1 instances > »

Instance ID	Status	Filter types	Association
i-0efcd347379c7c1fa4	Running	name:wordpress-CodeDeplo...	EC2

6. Moving on, in the **Deployment configuration** section, you can choose from a list of default and custom deployment configurations. As discussed earlier, a deployment configuration is a set of rules that determines how fast an application will be deployed along with the definition of success or failure conditions for a particular deployment.

There are three default configurations provided by AWS CodeDeploy itself:

- **OneAtATime**: Routes traffic to one instance in the replacement environment at a time
- **HalfAtATime**: Routes traffic to up to half the instances in the replacement environment at a time

- **AllAtOnce**: Routes traffic to all instances in the replacement environment all at once

Since we are working with just a single EC2 instance as of now, go ahead and select the **OneAtATime** deployment configuration option.

7. Finally, select the IAM Role that we created and assigned our EC2 instance to at the time of launching, using the **Service Role** drop-down list. Once done, select the **Create application** option to complete the process!

There you have it! If you made it this far then you have successfully configured both AWS CodeCommit and AWS CodeDeploy for our WordPress application's deployment! But we are still missing the glue that ties all these services together, and that precisely is what we will be talking about in the next section with the introduction of AWS CodePipeline!

Introducing AWS CodePipeline

AWS CodePipeline is a continuous delivery service that you can use to model, visualize, and automate the steps required to release your application software. This is made possible by building *pipelines* that contain one or more *stages*. The stages can be broadly classified as *build*, where the code is compiled and built using, say, AWS CodeBuild or some other third-party tool, *staging*, and *deployment*, where the code is pushed on to compute instances using AWS CodeDeploy, and so on. Each stage internally describes a set of actions that it needs to perform in order to prepare the software for its release. This action can be anything from building your source code from a Git repository, to making changes to a file, or deploying packages, and so on. Every change made to either your code or some configurational setting within CodePipeline is considered as a *revision* and you can have multiple such revisions created within a single stage of a pipeline.



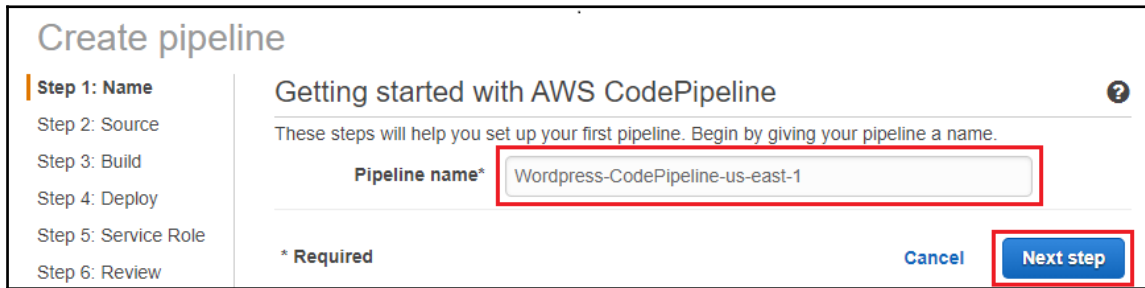
Even changes made to a single stage within the pipeline results in all actions across all stages being re-executed.

You can use these features provided by CodePipeline to effectively manage and monitor the release of your software. In this section, we will be continuing with our use case set up earlier using CodeCommit and CodeDeploy and see how we can truly build an end-to-end continuous delivery cycle using AWS CodePipeline.

Creating your own continuous delivery pipeline

Getting started with CodePipeline is extremely easy provided you have all the necessary prerequisites met, which include setting up the CodeCommit repository with your latest piece of application code (in this case, the WordPress application), as well as configuring the application and the AppSpec file using CodeDeploy:

1. To begin with, launch the CodePipeline Management dashboard by selecting URL `https://console.aws.amazon.com/codepipeline/home`.
2. Since this is our first setup, click on the **Get Started** option to get going. This will bring up the **Getting started with AWS CodePipeline** wizard, as shown in the following screenshot. Start off by providing a suitable **Pipeline name** and click **Next step** to continue:



3. Next, in the **Source** page, we need to select and configure the source for our new pipeline. At the time of writing this book, CodePipeline supports three source code providers, namely Amazon S3, AWS CodeCommit, and GitHub. For the purpose of this use case, go ahead and select **AWS CodeCommit** from the **Source** drop-down list.
4. This will automatically prompt you to enter the subsequent CodeCommit **Repository name** as well as its corresponding **Branch name**. Make sure you provide the same branch name that contains the latest WordPress code as well as the AppSpec file. Click on **Next step** to continue.
5. The third stage of the Pipeline setup is the **Build** stage where you can specify the build provider. CodePipeline supports three build providers, namely AWS CodeBuild, Jenkins, and Solano CI. Since our WordPress installation doesn't require any compilations or build procedures, simply select the **No Build** option from the drop-down list and click on **Next step** to continue.

6. The fourth state requires the **Deployment** configurations to be set up for the pipeline. Here too you are provided with various options that you can choose to leverage based on your needs. At present, CodePipeline supports AWS Opsworks, AWS CodeDeploy, AWS CloudFormation, and AWS Elastic Beanstalk as the **Deployment providers**. Since we have already configured AWS CodeDeploy for our use case, select the same from the drop-down list.
7. Next, fill in the correct **Application name** as well as the **Deployment group** that we configured during the setup of CodeDeploy. Click on **Next step** once done:

The screenshot shows the 'Create pipeline' wizard in the AWS CodePipeline console, specifically Step 4: Deploy. On the left, a sidebar lists the steps: Step 1: Name, Step 2: Source, Step 3: Build, Step 4: Deploy (highlighted), Step 5: Service Role, and Step 6: Review. The main area is titled 'Deploy' and contains instructions: 'Choose how you deploy to instances. Choose the provider, and then provide the configuration details for that provider.' Below this, the 'Deployment provider' dropdown is set to 'AWS CodeDeploy'. Under the 'AWS CodeDeploy' section, there are two instructions: 'Choose one of your existing applications, or create a new one in AWS CodeDeploy.' and 'Choose one of your existing deployment groups, or create a new one in AWS CodeDeploy.' The 'Application name' field is filled with 'Wordpress-CodeDeploy' and the 'Deployment group' field is filled with 'Wordpress-CodeDeploy-DeploymentGrp'. At the bottom, there are three buttons: 'Cancel', 'Previous', and 'Next step' (highlighted with a red box). A red box also highlights the 'Application name' and 'Deployment group' fields. A red box highlights the 'Next step' button.

8. The final step required is to configure the **Service Role**. The service role essentially grants CodePipeline permissions to use resources in your AWS account. Provide a suitable **Role name** and click on **Next step** to review the pipeline's configuration.
9. On the **Review your pipeline** page, ensure that all the fields are correctly configured and click on **Create pipeline** when done.

Selecting this option first creates a unique S3 bucket within your environment that will contain and store all the necessary artifacts for this particular pipeline. Once the pipeline is created, you can view it on the AWS CodePipeline dashboard.

Putting it all together

With the pipeline all set up, you can now test the entire setup as one cohesive solution!

First up, ensure that there are no errors in either of the stages during the deployments. In case there are any errors, simply select the particular error link provided in the stage and follow it back to its source, which can be anywhere from issues in CodeCommit to even the setting up of CodeDeploy. Here's a screenshot of the pipeline that we created using an accumulation of all of the preceding sections:

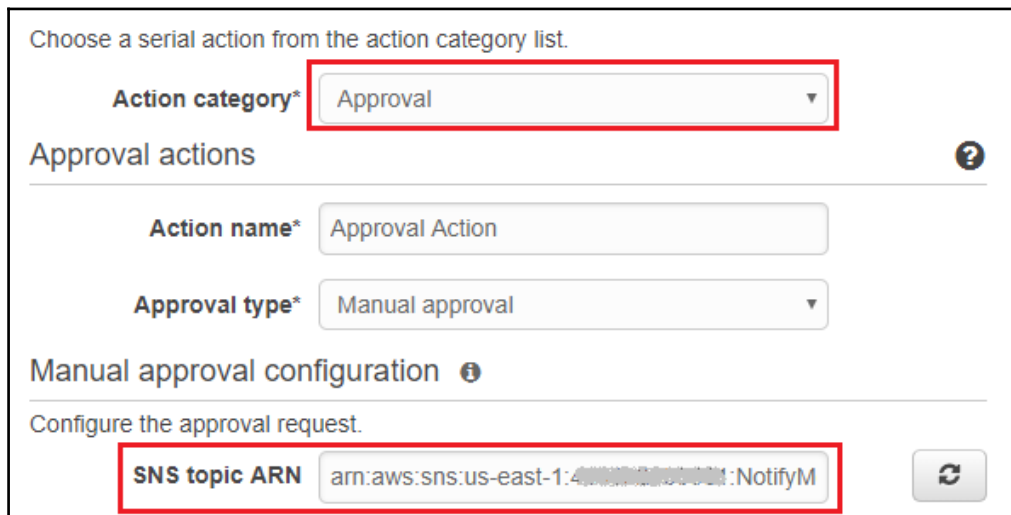
The screenshot displays the AWS CodePipeline console for a pipeline named "Wordpress-CodePipeline-us-east-1". At the top right, there is a link "View pipeline history" enclosed in a red box. Below the pipeline name, the text "View progress and manage your pipeline." is visible. On the left, there are two buttons: "Edit" (highlighted with a red box) and "Release change". The pipeline consists of two stages: "Source" and "Staging". The "Source" stage uses the "AWS CodeCommit" provider and shows a successful execution "1 min ago" with commit ID "c0508ea". Below this, a message indicates "Source: Updated the insta...". The "Staging" stage uses the "Wordpress-CodeDeplo." provider (AWS CodeDeploy) and shows a successful execution "just now". Below this, another message indicates "Source: Updated the insta...".

Here, you can choose to add more stages to your pipeline by simply selecting the **Edit** option, as highlighted earlier. Additionally, you can also view your pipeline's execution history by selecting the **View pipeline history** option.

In the **Edit** pipeline page, you can choose to add one or more stages to your pipeline as you see fit. Simply select the **+ Stage** option provided at the end of each existing stage. This will bring up a new dialog where you can specify the stage's **Name** as well as define one or more actions.

Consider the following use case where we need to add an approval step before the code actually gets pushed into the staging area. In that case, we need to add a new stage between the existing **Source** and **Staging** stages:

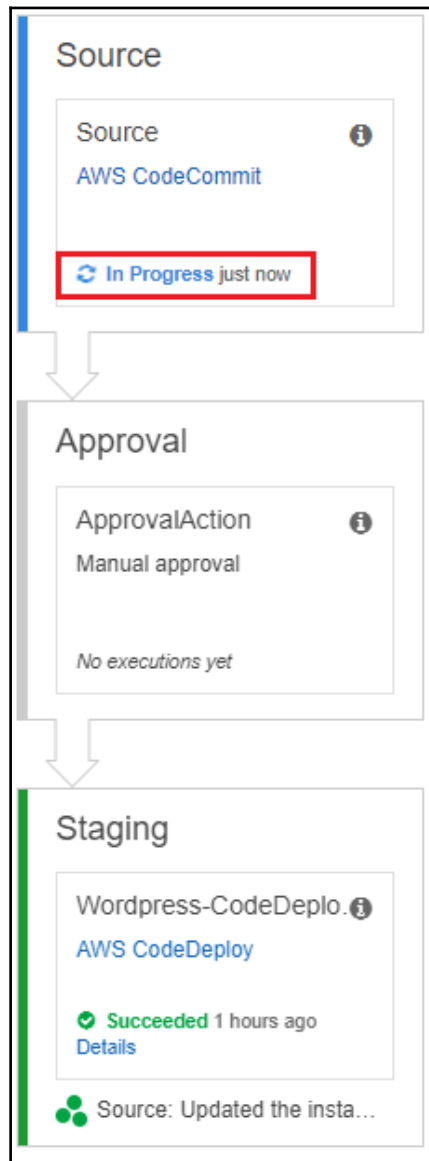
1. Click on the **+ Stage** option and provide a suitable name for this new stage. Next, select the **+ Action** option to add the rules for setting up the approval process.
2. In the **Add action** dialog box, start by selecting the type of action from the **Action category** drop-down list. The following list of actions can be added to a stage: **Approval**, **Source**, **Build**, **Test**, **Deploy**, and **Invoke**. For this use case, select **Approval**:



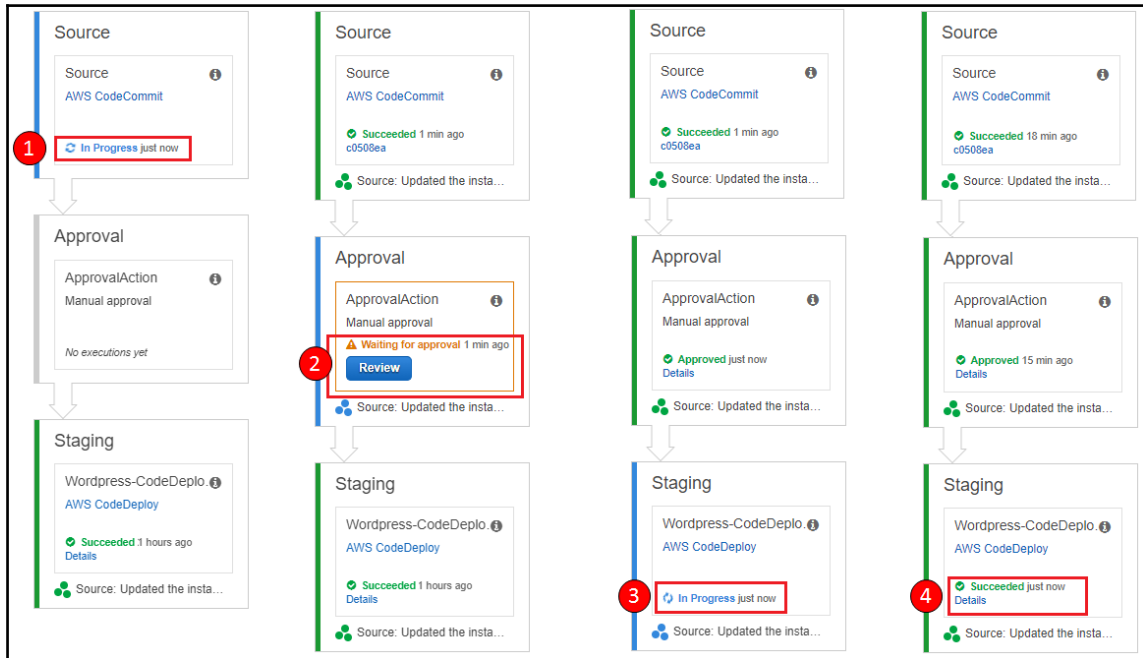
The screenshot shows the 'Add action' dialog box in AWS CodePipeline. At the top, it says 'Choose a serial action from the action category list.' Below this, the 'Action category*' dropdown is set to 'Approval' and is highlighted with a red box. Under the 'Approval actions' section, the 'Action name*' is 'Approval Action' and the 'Approval type*' is 'Manual approval'. Below this is the 'Manual approval configuration' section, which includes the instruction 'Configure the approval request.' and an 'SNS topic ARN' field. The 'SNS topic ARN' field contains the value 'arn:aws:sns:us-east-1:4[redacted]:NotifyM' and is also highlighted with a red box. A refresh button is located to the right of the ARN field.

3. Fill in a suitable **Action name** and select an appropriate **Action type** as well. At present, only a **Manual approval configuration** option is provided by CodePipeline.

4. Finally, select either one of a pre-existing SNS topics using the **SNS topic ARN** field and click on **Add action** once completed.
5. With the new stage added, click on **Save pipeline changes** to commit the change. You should have a new stage added and visible in your pipeline, as shown in the following screenshot:



6. Once the **Source** stage is prepped, the pipeline automatically halts at the **Approval** stage, as depicted in the following screenshot. Here, you can approve the process by simply selecting the **Review** option and then selecting the approval command. Doing so initiates the final **Staging** stage which invokes the CodeDeploy service to deploy the WordPress code over to our awaiting EC2 instance:



After a few minutes, the application is successfully uploaded on the instance and is up and running as well. You can verify this by making a note of your instances public IP address and typing in the following URL on a browser:

```
http://WordPress/wp-admin/index.php
```

Remember to prefix **WordPress** in your URL since the AppSpec file clearly pointed to the root of the application at the `/var/www/html/WordPress` directory and not at `/var/www/html` itself.

In this way, you can easily leverage and automate the deployments of your application code using the AWS Code Suite of services! Amazing isn't it? But that's not all folks! There is a ton of other fascinating things that I would like you to learn and explore, all covered in the next section itself.

Planning your next steps

Well, we have covered a lot of new features and services in this chapter. However, there are still a few things that I would recommend you need to read up on your own as well. First up is the AWS CodeStar!

AWS CodeStar is an amazing service that can help developers create, manage, and work on developing applications, all one place. The integral part of CodeStar that enables this is called **projects**. Developers can create projects based on predefined templates that contain a supported programming model which is ready to use for developing. You can additionally select hosting for your application from a variety of options that include Amazon EC2, Elastic Beanstalk, and AWS Lambda as well! CodeStar comes with easy connectivity with various IDEs as well: Eclipse, Visual Studio, just to name a few. You can read more about *AWS CodeStar* at <http://docs.aws.amazon.com/codestar/latest/userguide/welcome.html>.

The second awesome service worth trying out is AWS Cloud9! AWS Cloud9 is a cloud-based IDE that you can use to write, debug, and run code, all with a standard web browser! Cloud9 comes pre-packaged with essential tools for the most popular programming languages, including JavaScript, Python, PHP, Node.js, C++, and much more, and the best part is you can easily integrate Cloud9 with the likes of other AWS services, such as CodeCommit, CodeStar, and so on. You can read more about AWS Cloud9 at <https://docs.aws.amazon.com/cloud9/latest/user-guide/welcome.html>.

Last, but not least, I would also recommend that you try out a complete, end-to-end CI/CD solution provided by the **AWS Partner Network (APN)** blog that leverages a CloudFormation stack to deploy a full-fledged AWS Code Suite, including CodeBuild, CodeCommit, CodeDeploy, and CodePipeline! The stack gives you a feel for using AWS CodeBuild as a service and also provides you with a template to automate your own Code deployments using CloudFormation. To view the blog visit <https://aws.amazon.com/blogs/apn/deploy-to-production-using-aws-codebuild-and-the-aws-developer-tools-suite/>.

Summary

Here's a quick round up of the topics that we have covered so far in this chapter!

We started off by learning and understanding a bit about the AWS Code Suite of services and how they tie into the continuous integration and continuous delivery life cycle. We then learned a bit about AWS CodeDeploy and how easy it is to get started with it. We created a simple CodeCommit repository and used that to store our sample WordPress application as well. Later, we looked at AWS CodeDeploy and how it works. We also configured a CodeDeploy agent and leveraged the AppSpec file for our WordPress application's deployment life cycle. Finally, we tied it all together by integrating the work done so far with AWS CodePipeline and ended the chapter with a few essential things to read and try out on your own!

In the next chapter, we will be exploring two application-specific services and look at how you can leverage them for your requirements: Amazon SQS and Amazon SNS. So stay tuned!

8

Messaging in the Cloud Using Amazon SNS and Amazon SQS

In the previous chapter, we briefly explored the AWS code suite of services, namely AWS CodeCommit, AWS CodeDeploy, and AWS CodePipeline, and how they tie into the continuous integration and continuous delivery life cycle of an application.

In this chapter, we will be learning and exploring yet another group of AWS services that are extremely useful when it comes to developing modern cloud-ready applications, as well as for the general housekeeping of your AWS accounts: Amazon **Simple Notification Services**, or **SNS**, and Amazon **Simple Queue Service**, or **SQS**.

Keeping this in mind, let's have a quick look at the various topics that we will be covering in this chapter:

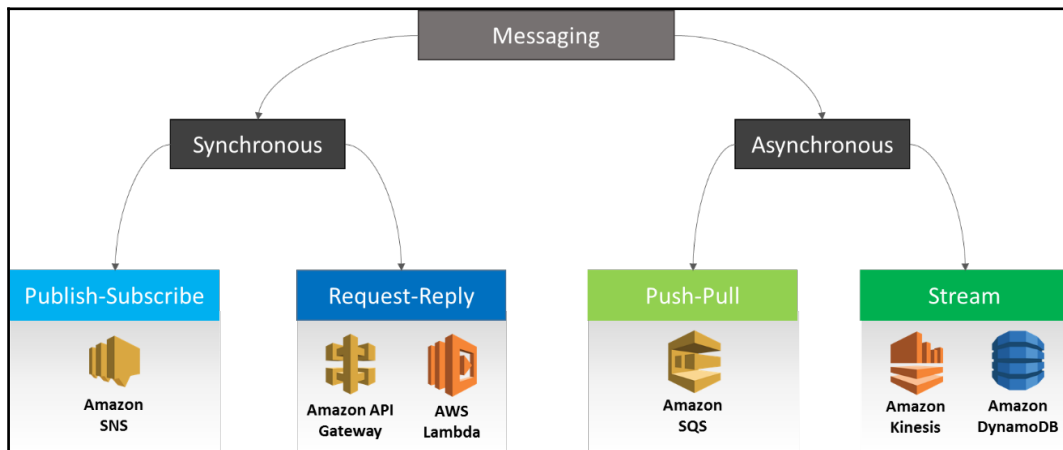
- Introducing the various messaging services provided by AWS, along with an insight into which service to use for what purpose
- Introducing Amazon SNS and Amazon SQS, along with their core concepts and terminologies
- Creating your own SNS topics and subscriptions, and leveraging them for your AWS account
- Monitoring SNS notifications using Amazon CloudWatch

- Integrating Amazon SNS with Slack for a richer, user notification experience
- Getting started with standard and FIFO queues, and integrating Amazon SNS with Amazon SQS

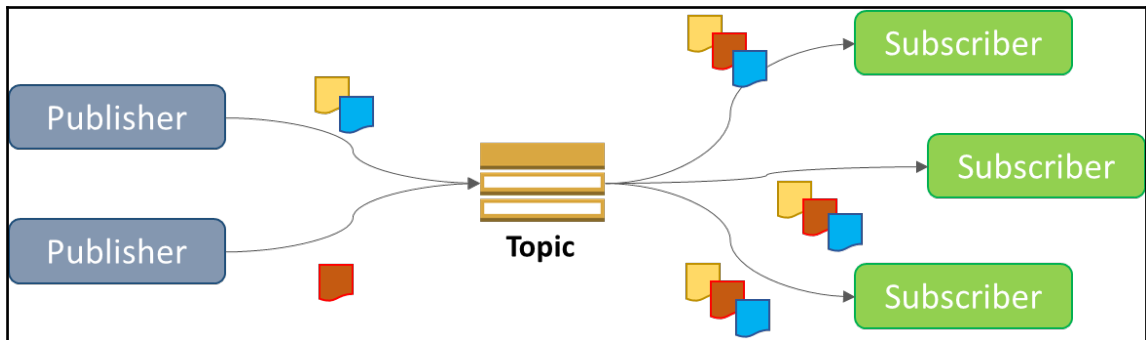
So, without any further ado, let's get started!

Understanding the AWS messaging services

We all know by now that AWS provides a plethora of services designed to help you with developing a rich set of cloud-ready applications; but with so many different services to choose from, how do you make the right set of choices to begin with? That's exactly what we will be learning and exploring in this section, starting with a brief understanding and comparison of a few commonly used AWS messaging services, as depicted in the following diagram:

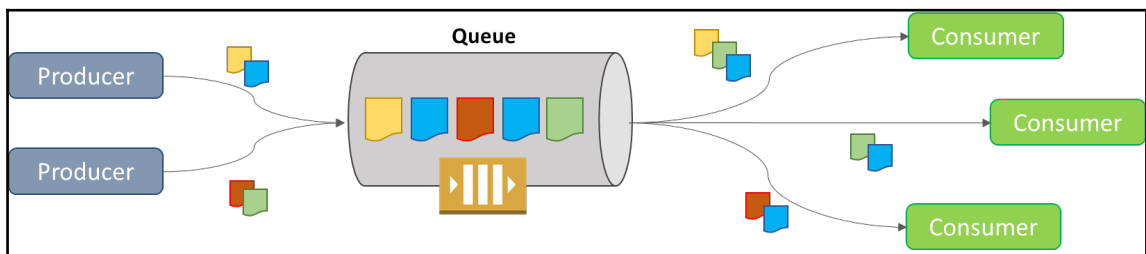


- **Amazon SNS:** Amazon SNS, or **Simple Notification Service**, is a synchronous, managed service that provides the end user with the ability to deliver or send messages to one or more endpoints or clients. This works by using a **Publisher-Subscriber**-like model, as depicted in the following diagram:

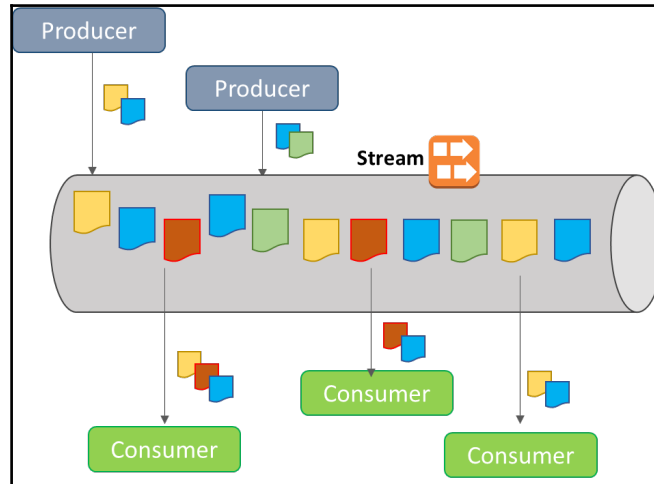


One or more publishers or producers post a message to a corresponding SNS **topic** without knowing which subscribers or consumers will ultimately consume the message. The producer also doesn't wait for a response back from the consumers, thus making SNS a loosely-coupled service. It is the consumer's task to subscribe to the topic and get notified of the incoming messages. SNS supports a variety of consumer implementation options, such as email, mobile push notifications or SMS, HTTP/HTTPS notifications, and even Lambda functions.

- **Amazon SQS:** Amazon SQS, or **Simple Queue Service**, on the other hand, is an asynchronous managed service that provides users with the ability to push and pull messages from a queue. Here too, one or more producers can be used to push messages into the queue, which a corresponding set of consumers on the other end consume and process the messages one at a time. An important point to note here is that, unlike its counterpart, SNS, where the consumers are notified of a new message, here, the consumers have to poll the queue in short intervals of time for newer messages. Once a message is found, the consumer has to process it and then delete it from the queue. The process is shown here:



- **Amazon Kinesis:** Amazon Kinesis functions a lot like Amazon SQS; however, it is fundamentally designed and optimized for high-throughput data writes and reads. Here, instead of a queue, you are provided with a stream that consumers can use to read from multiple times. The stream is automatically trimmed after a span of 24 hours, so, unlike your consumers from the queue, here you are not required to delete the messages once they are processed:



Similar to Amazon Kinesis, AWS also provides a streaming functionality with DynamoDB as well, called DynamoDB streams. Using this feature, you can basically enable real-time changes to certain items within your tables in the form of a stream. And, finally, you also get the standard request–reply model of messaging using a combination of Amazon API Gateway, ELBs, AWS Lambda, and other services. This mode of communication is also synchronous in nature and can be used to fit a variety of use cases, as per your requirements.

Keeping these basic differences in mind, let's now move forward and learn more about SNS.

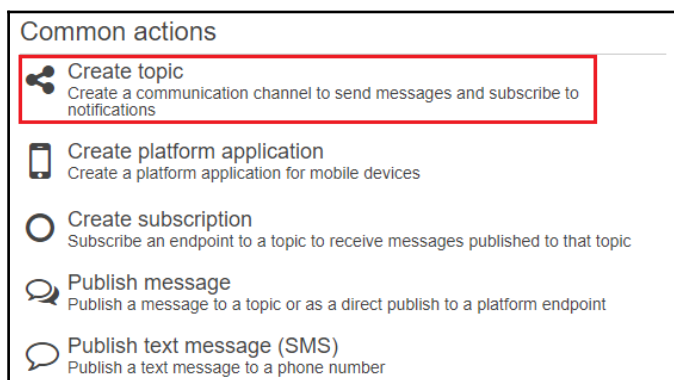
Getting started with Amazon Simple Notification Service

As discussed briefly earlier, SNS is a managed web service that you, as an end user, can leverage to send messages to various subscribing endpoints. SNS works in a publisher–subscriber or producer and consumer model, where producers create and send messages to a particular topic, which is in turn consumed by one or more subscribers over a supported set of protocols. At the time of writing this book, SNS supports HTTP, HTTPS, email, push notifications in the form of SMS, as well as AWS Lambda and Amazon SQS, as the preferred modes of subscribers.

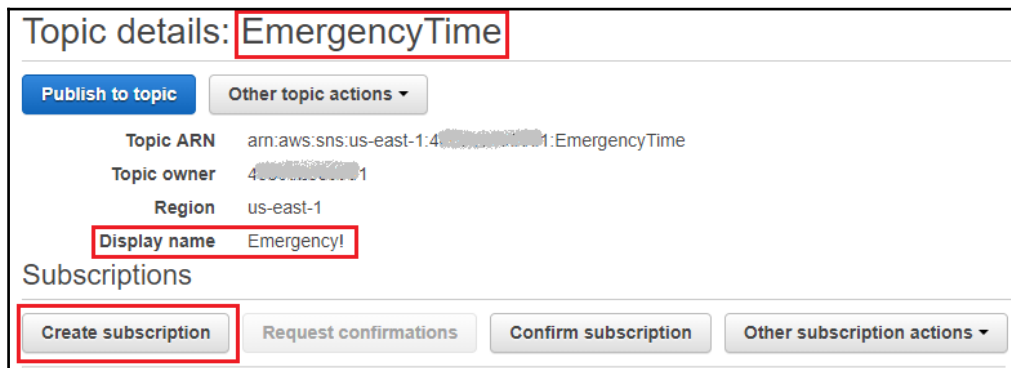
SNS is a really simple and yet extremely useful service that you can use for a variety of purposes, the most common being pushing notifications or system alerts to cloud administrators whenever a particular event occurs. We have been using SNS throughout this book for this same purpose; however, there are many more features and use cases that SNS can be leveraged for. For example, you can use SNS to send out promotional emails or SMS to a large group of targeted audiences, or even use it as a mobile push notification service where the messages are pushed directly to your Android or IOS applications.

With this in mind, let's quickly go ahead and create a simple SNS topic of our own:

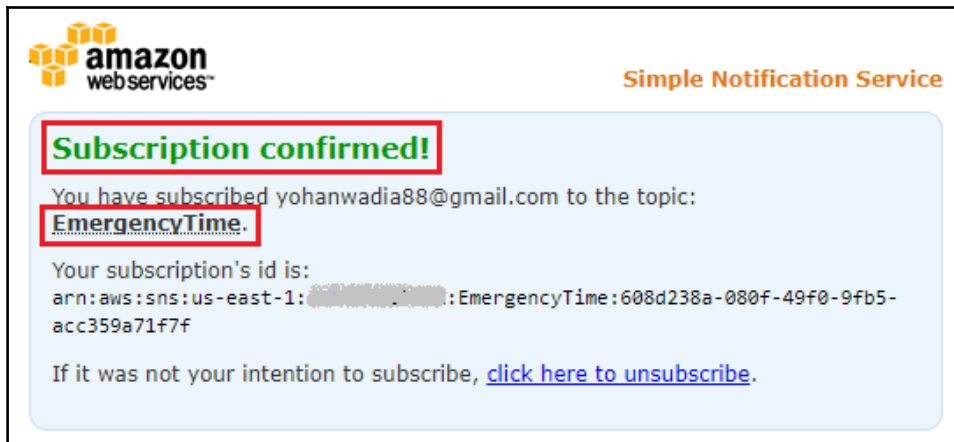
1. To do so, first log in to your AWS Management Console and, from the **Filter** option, filter out SNS service. Alternatively, you can also access the SNS dashboard by selecting <https://console.aws.amazon.com/sns>.
2. If this is your first time with SNS, simply select the **Get Started** option to begin. Here, at the SNS dashboard, you can start off by selecting the **Create topic** option, as shown in the following screenshot:



3. Once selected, you will be prompted to provide a suitable **Topic name** and its corresponding **Display name**. Topics form the core functionality for SNS. You can use topics to send messages to a particular type of subscribing consumer. Remember, a single topic can be subscribed by more than one consumer. Once you have typed in the required fields, select the **Create topic** option to complete the process. That's it! Simple, isn't it?
4. Having created your topic, you can now go ahead and associate it with one or more subscribers. To do so, first we need to create one or more subscriptions. Select the **Create subscription** option provided under the newly created topic, as shown in the following screenshot:



5. Here, in the **Create subscription** dialog box, select a suitable **Protocol** that will subscribe to the newly created topic. In this case, I've selected **Email** as the **Protocol**. Next, provide a valid email address in the subsequent **Endpoint** field. The **Endpoint** field will vary based on the selected protocol. Once completed, click on the **Create subscription** button to complete the process.
6. With the subscription created, you will now have to validate the subscription. This can be performed by launching your email application and selecting the **Confirm subscription** link in the mail that you would have received.
7. Once the subscription is confirmed, you will be redirected to a confirmation page where you can view the subscribed topic's name as well as the subscription ID, as shown in the following screenshot:

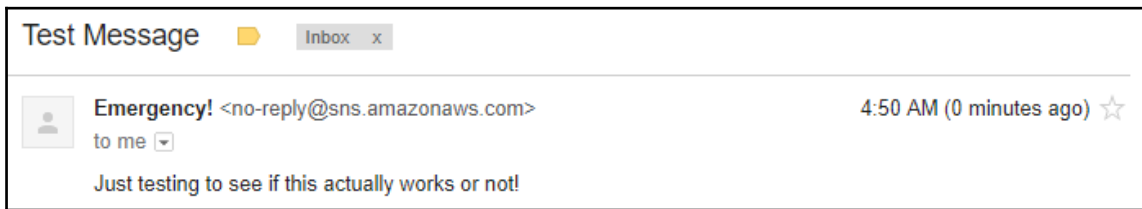


8. You can use the same process to create and assign multiple subscribers to the same topic. For example, select the **Create subscription** option, as performed earlier, and from the **Protocol** drop-down list, select **SMS** as the new protocol. Next, provide a valid phone number in the subsequent **Endpoint** field. The number can be prefixed by your country code, as shown in the following screenshot. Once completed, click on the **Create subscription** button to complete the process:

A screenshot of the 'Create subscription' form in the Amazon SNS console. The form has three main input fields: 'Topic ARN' with the value 'arn:aws:sns:us-east-1:408888888888:EmergencyTime', 'Protocol' with a dropdown menu showing 'SMS', and 'Endpoint' with the value '31-687551234'. A red rectangle highlights the 'Protocol' and 'Endpoint' fields. At the bottom right, there are two buttons: 'Cancel' and 'Create subscription'.

9. With the subscriptions created successfully, you can now test the two by publishing a message to your topic. To do so, select the **Publish to topic** option from your topics page. Once a message is published here, SNS will attempt to deliver that message to each of its subscribing endpoints; in this case, to the email address as well as the phone number.

10. Type in a suitable **Subject** name followed by the actual message that you wish to send. Note that if your character count exceeds 160 for an SMS, SNS will automatically send another SMS with the remainder of the character count. You can optionally switch the **Message format** between **Raw** and **JSON** to match your requirements. Once completed, select **Publish Message**.
11. Check your email application once more for the published message. You should receive an mail, as shown in the following screenshot:



Similarly, you can create and associate one or more such subscriptions to each of the topics that you create. In the next section, we will look at how you can leverage SNS to send SMS messages or text messages to one or multiple phone numbers.

Sending text messages using SNS

Amazon SNS also provides users with a really easy-to-use interface which allows you to send text messages or SMS messages to one or multiple phone numbers. It also provides you with the ability to classify and send messages based on their criticality, as well as specify the maximum amount that you wish to spend on sending SMS messages each month. So, without wasting any time, let's get straight to it:

1. To send SMS messages using SNS, first log in to the SNS dashboard by selecting <https://console.aws.amazon.com/sns/>.
2. Once logged in, select the **Text messaging (SMS)** option from the navigation pane. This will bring up the **Text messaging (SMS)** dashboard, where you can set your SMS preferences as well as send messages to one or more phone numbers. First up, let's set some preferences by selecting the **Manage text messaging preferences** option from the dashboard.

3. Fill in the following preference fields:

- **Default message type:** SNS provides two message types: **Promotional** and **Transactional**. The **Promotional** option can be selected if the messages that you wish to send require less criticality, for example, simple marketing messages, and so on. On the other hand, **Transactional** messages are ideally suited for critical messages, such as one-time passwords, transaction details, and so on. SNS optimizes the message delivery for **Transactional** messages to achieve the best reliability.



At the time of writing this book, sending SMS messages is supported in the countries listed at https://docs.aws.amazon.com/sns/latest/dg/sms_supported-countries.html.

For this particular scenario, I've selected the **Promotional** option, as shown in the following screenshot:

The screenshot shows the 'Text messaging preferences' dialog box. The 'Default message type' dropdown is set to 'Promotional' and is highlighted with a red rectangle. Other fields include 'Account spend limit' (1 USD), 'IAM role for CloudWatch Logs access' (arn:aws:iam::[redacted]:role/SNSSuccessIAMRole), 'Default percentage of success to sample' (100), 'Default sender ID' (YoYo), and 'Reports storage' (Name of an S3 bucket). There are 'Cancel' and 'Update preferences' buttons at the bottom right.

Text messaging preferences

Default message type: Promotional ⓘ

Account spend limit: 1 USD ⓘ

IAM role for CloudWatch Logs access: arn:aws:iam::[redacted]:role/SNSSuccessIAMRole ⓘ Change | Clear

Default percentage of success to sample: 100

Default sender ID: YoYo ⓘ

Reports storage: Name of an S3 bucket ⓘ

Cancel Update preferences

- **Account spend limit:** The maximum amount you wish to spend, in USD, for sending messages in a month. By default, the limit is set to USD 1.00. For this scenario, we are not going to change this value.



Both **Promotional** and **Transactional** message types have different costs based on the specified country or region. You can look up the prices at <https://aws.amazon.com/sns/sms-pricing/>.

- **IAM role for CloudWatch Logs access:** This option is used to create an IAM role that basically allows Amazon SNS to write its logs to CloudWatch. Since this is the first time we are configuring this feature, select the **Create IAM role** option. This will redirect you to a new page where you should select the **Allow** option to grant SNS the necessary rights. Here is a snippet of the rights that are provided for your IAM role:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:PutMetricFilter",
        "logs:PutRetentionPolicy"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

- **Default percentage of success to sample:** This option is used to specify the percentage of successful SMS messages delivered, based on which SNS will write logs into CloudWatch. To write only logs for failed message deliveries, set this value to 0. By default, SNS will write logs for all successful deliveries (100%).
- **Default sender ID:** This option is used to specify the name of the message's sender. You can provide any meaningful name here.

- **Reports storage:** Use this option to configure an S3 bucket that will store daily SMS usage reports from Amazon SNS. If you are providing an existing bucket as your **Reports storage** then ensure that it has the necessary access rights to interact with the SNS service.
4. Once the required fields are filled in, select the **Update preferences** option to complete the process.

To send the SMS messages, simply select the **Send a text message (SMS)** option from the **Text messaging (SMS)** dashboard. This will bring up the **Send text message (SMS)** dialog box, as shown in the following screenshot. Provide a valid phone **Number** and a **Message**. Remember to prefix your country code in the phone number as well:

The screenshot shows a dialog box for sending an SMS. The 'Message type' dropdown is set to 'Promotional'. The 'Number' field contains '+31-678551234'. The 'Message' field contains 'This is just a test message from AWS!'. The 'Sender ID' field is empty. A character count '37 / 160' is displayed next to the message field. At the bottom right, there are two buttons: 'Cancel' and 'Send text message'.

You can optionally even overwrite the **Sender ID** field here, however, for this case, we have left it to the default value that was configured in the preferences stage. After the required fields are filled in, simply select **Send text message** to complete the message-sending process. You can also verify the delivery status of each message sent, either **Transactional** or **Promotional**, by using the **Account stats** section provided in the **Text messaging (SMS)** page.

Using Amazon SNS as triggers

One of the key benefits of having a service such as SNS is that it can also be used as a trigger mechanism for a variety of use cases. Messages sent by SNS can be used to trigger simple Lambda functions that in turn perform some action over another AWS service, or simply process the message from SNS and forward its contents to another application. In this section, we will be exploring a really simple use case where an SNS topic is used as a trigger mechanism for a Lambda function to push CloudWatch alerts over to Slack! The alerts will be sent out to a custom-made Slack channel that your IT team can use to track alerts and other important notifications with regards to your AWS environment.

At a broader level, here are the list of things that we plan to do for this activity:

- Create an SNS topic that will act as the Lambda trigger
- Create a CloudWatch alarm for one of our EC2 machines, say, if CPU utilization goes higher than 80% then trigger the alarm
- The CloudWatch alarm will post the notification to an SNS topic
- The SNS topic will act as a trigger to our Lambda function
- As soon as the Lambda function gets a trigger, it will post the notification to our Slack channel

Sounds simple? Let's get down to implementing it then:

1. First, we will need to create a simple SNS topic which will act as a trigger for the Lambda function. Go ahead and create a simple SNS topic as we did in our earlier steps. Once completed, make a note of the SNS topic's ARN from the topics dashboard. In this case, our SNS is configured to send notifications to an email subscriber in the form of an IT admin email alias.
2. Next up, we create our CloudWatch alarm. To do so, select the **CloudWatch** service from the AWS Management Console and click on **Alarms** in the navigation panel. Select **Create alarm** to get started.
3. In this scenario, we will be monitoring the EC2 instances in our environment, so I've gone ahead and selected the **EC2 Metrics** option. Alternatively, you can select any other **Metrics**, as per your requirements. In our case, we have gone ahead and configured a simple **CPUUtilization** alarm, as shown in the following screenshot:

1. Select Metric
2. Define Alarm

Alarm Threshold

Provide the details and threshold for your alarm. Use the graph on the right to help set the appropriate threshold.

Name:

Description:

Whenever: CPUUtilization

is:

for: consecutive period(s)

Additional settings

Provide additional configuration for your alarm.

Treat missing data as: ⓘ

- Make sure that you set up a notification for the alerts and point it to the newly created SNS topic, as shown in the following screenshot:

Notification
Delete

Whenever this alarm:

Send notification to: ⓘ [New list](#) [Enter list](#)

This notification list is managed in the SNS console.

+ Notification
+ AutoScaling Action
+ EC2 Action

With the SNS topic and CloudWatch alarm in place, we now need to configure a Slack channel where the alert notifications will be posted. For that, we will need an incoming webhook to be set and a hook URL that will be used to post the notifications:

1. Go to your Slack team's settings page and select the **Apps & integrations** option, as shown in the following screenshot:



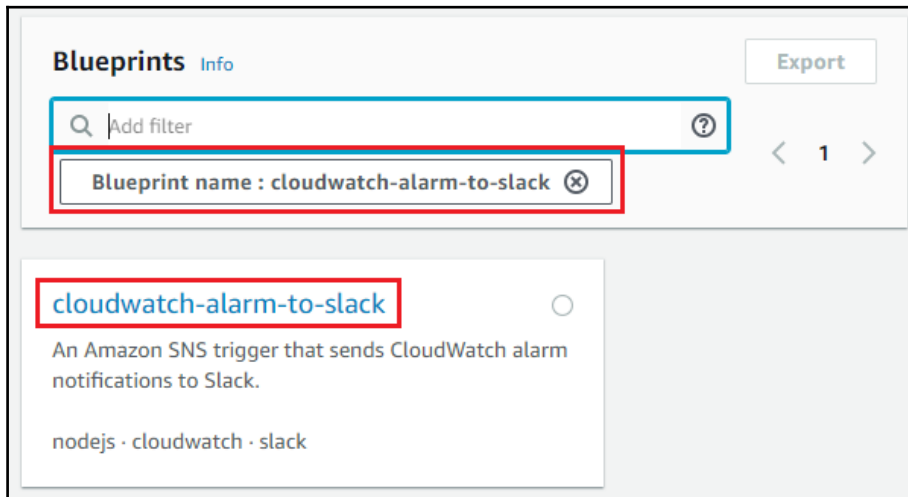
You can sign up for a free Slack account at <https://slack.com/get-started>.

2. Once you click on **Apps & integrations**, it will take you to a new page which lists a variety of pre-configured apps. Search for **Incoming** and select the **Incoming Webhooks** from the options that appear.
3. Next, click on **Add Configuration**. It will ask you to select the **Channel** to post, along with a few other necessary parameters. Make sure that you copy and save the **Webhook URL** before you proceed any further with the next steps.

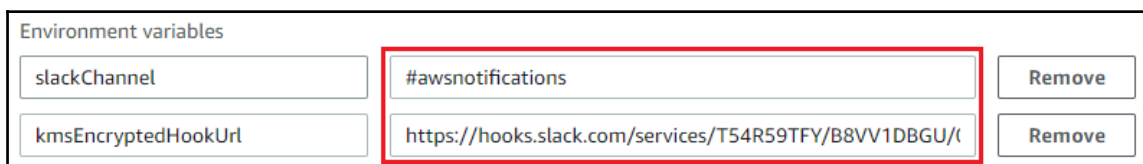
Now that we have our Slack hook URL ready, we can finally get started with deploying our Lambda function. For this exercise, we will be using an existing AWS Lambda function blueprint designed for Slack integration, using the Node.js 4.3 version:

1. From the AWS Management dashboard, filter the service **Lambda** using the **Filter** option, or alternatively, select <https://console.aws.amazon.com/lambda/home>.
2. From the **AWS Lambda** landing page, select the **Create a function** option to get started.

- For working with Lambda functions, you can choose to create your own function from scratch, or alternatively, filter and use a function from a list of predefined and configured blueprints. In this case, select the **Blueprints** option and use the adjoining blueprints filter to search for the following function: **Blueprint name: cloudwatch-alarm-to-slack** (as shown in the following screenshot):



- Select the blueprint and fill out the necessary information for your function, such as its name, role name, and so on. Once done, from the **SNS** section, select the newly created SNS topic from the drop-down list.
- Remember to select the **Enable trigger** checkbox before proceeding with the next steps.
- Finally, in the **Environment variables** section, provide the appropriate values for the `slackChannel` and `kmsEncryptedHookUrl` parameters, as shown in the following screenshot. Remember, the `kmsEncryptedHookUrl` is nothing but the **Slack hook URL** that we created a while back:



- With the values filled in, simply select the **Create function** option and let the magic begin!

Based on the selected CloudWatch metric for your alarm, go ahead and create some synthetic load for your EC2 instance. Once the load crosses the set threshold in the alarm, it triggers a corresponding message to the SNS topic, which in turn triggers the Lambda function to post the alert over on the Slack channel. In this way, you can also use the same SNS topic for subscribing to various other services, such as Amazon SQS, for other processing requirements.

Monitoring Amazon SNS using Amazon CloudWatch metrics

Amazon SNS automatically collects and sends various metrics about message deliveries to Amazon CloudWatch. You can view these metrics and assign them with alarms to alert you, in case a message delivery rate drops beyond a certain threshold. You can additionally view the message delivery logs, as well using the CloudWatch Logs page:

1. To get started, first ensure that you have assigned an IAM role that allows SNS to write SMS delivery logs over to CloudWatch. To do so, from the navigation pane, select the **Text messaging (SMS)** option.
2. Next, from the **Manage text messaging preferences** option, ensure that you have a valid IAM role provided under the **IAM role for CloudWatch Logs access** field.
3. Once the IAM role is created, log in to your CloudWatch dashboard by selecting <https://console.aws.amazon.com/cloudwatch/home>.
4. Here, select the **Logs** option from the navigation pane to bring up the CloudWatch **Log Groups** page. You should see a default Log Group created here, by the name of `DirectPublishToPhoneNumber`.
5. Select the Log Group to view the SMS delivery log messages. The logs will either show a `SUCCESS` or `FAILURE` in the `status` field, as shown in the following screenshot:

```
{
  "notification": {
    "messageId": "1f82b95a-a4ea-5e7c-a1c0-eb80bbe7a296",
    "timestamp": "2018-01-18 04:04:29.587"
  },
  "delivery": {
    "phoneCarrier": "KPN Mobile The Netherlands B.V.",
    "mnc": 8,
    "destination": "+31678551234", 2
    "priceInUSD": 0.15219,
    "smsType": "Promotional",
    "mcc": 204,
    "providerResponse": "Message has been accepted by phone carrier",
    "dwellTimeMs": 457,
    "dwellTimeMsUntilDeviceAck": 1332
  },
  "status": "SUCCESS"
}
```

6. You can additionally create and associate CloudWatch alarms with your monitored SNS metrics. To do so, from the CloudWatch dashboard, select the **Metrics** option.
7. From the **All metrics** tab, filter and select the **SNS** option.
8. Based on the requirements, you can now select between viewing the metrics based on the **PhoneNumber**, or **Country**, **SMSType**, and so on. In this case, we have selected the **PhoneNumber** option to view the **NumberOfNotificationsFailed** and **NumberOfNotificationsDelivered** metrics.
9. Next, select the **Graphed metrics** tab to view the two metrics and their associated actions. Using the **Actions** column, select the **Create alarm** option for the metric that you wish to monitor.
10. Fill in the respective details and configure the alarm's threshold values based on your requirements. Once completed, click on **Create Alarm** to complete the process.

In this way, you can leverage Amazon CloudWatch to create and view logs and alerts generated by the SNS service. In the next section, we will be exploring and learning a bit about the second part of the AWS messaging services: SQS.

Introducing Amazon Simple Queue Service

Amazon SQS is a managed, highly scalable, and durable service that provides developers with a mechanism to store messages that can be later consumed by one or more applications. In this section, we will be exploring a few of the concepts and terminologies offered by SQS along with an understanding of which SQS, queue to use for what scenarios, so let's get started!

To start off with, SQS is provided in two different modes:

- **Standard queue:** Standard queues are the default selection when it comes to working with SQS. Here, the queues created offer a nearly-unlimited **transaction per second (TPS)** rate coupled with an *at-least-once* delivery model. What this model means is that a message can be delivered at least once, but occasionally there is a good probability that more than one copy of that same message can be delivered as well. This is due to the fact that SQS is designed and built on a highly distributed system that is known to create copies of the same message in order to maintain a high-availability scenario. As a result, you may end up with the same message more than once.

Standard queues also work on a *best-effort ordering* model, in which case, messages might be delivered in a different order to the one in which they were sent. It is up to your application to sort the messages into the right order in which the messages should be received. So, when is the standard queue an ideal choice for decoupling your applications? Well, if your application has a high throughput requirement, for example, processing of batch messages, decoupling incoming user requests from an intense background processing work, and so on, then standard queues are the right way to go.

Standard queues are available across all AWS regions.

- **FIFO queues:** When working with standard queues, there is a problem of maintaining the order of the messages and also ensuring that each message is processed only once. To solve this issue, AWS introduced the FIFO queue that provides developers with a guaranteed order of delivery of messages, as well as the assurance that each message is delivered only once, where no duplicates or copies are ever sent out.

FIFO queues, on the other hand, do not offer an unlimited throughput capacity, unlike their predecessor. At the time of writing this book, FIFO queues support up to 300 messages sent per second, with an additional 3,000 messages per second capacity if a batch of 10 messages per operation is performed.

Such queues are really useful when the order of the messages is of critical importance, for example, ensuring that a user follows the correct order of events while registering or purchasing a product, and so on.

FIFO queues are currently only available in the US East (N. Virginia), US East (Ohio), US West (Oregon), and EU (Ireland) regions.

With this basic understanding, let's look at some simple steps to get you started with your very own queue in a matter of minutes!

Creating your first queue

Getting started with your own SQS queue is a fairly straightforward process. In this section, we will be looking at how you can create your very own standard queue using the AWS Management Console:

1. To begin with, log in to your AWS Management Console and filter out the **SQS** service using the **Filter** option provided. Alternatively, you can also access the SQS dashboard by selecting <https://console.aws.amazon.com/sqs/home>.
2. Since this is our first time configuring the SQS queue, select the **Get started now** option to continue.
3. Here, in the **Create New Queue** page, start off by providing a suitable name for your queue by filling in the **Queue Name** field.



If you are building a FIFO queue, you will need to suffix `.fifo` after your queues name, for example: `myQueue.fifo`.

4. With the queue name filled out, the next step is to select the type of queue you wish to set up. In this case, let's first start off by selecting the **Standard Queue** option.

5. Next, select the **Configure Queue** option to go through some of the queue's configuration parameters. Alternatively, you can also select the **Quick>Create Queue** option to select all the default parameters for your queue.
6. In the **Queue Attributes** section, feel free to modify the following set of parameters for your queue, based on your requirements:
 - **Default Visibility Timeout:** Amazon SQS does not automatically delete messages from the queue, even if they are processed by the consumers. Hence, it is the consumer's duty to delete the respective message from the queue after it has been received and processed.

However, due to the distributed nature of SQS, there is no guarantee that other consumers may not try to read from a copy of the same message. To prevent such scenarios from occurring, SQS sets a small *Visibility Timeout* period on a message once it is received by a consumer. This prevents other consumers from reading that message until the timeout expires.

By default, the Visibility Timeout can be set to a minimum of 30 seconds to a maximum of 12 hours. If, by chance, the consumer is not able to process the message in the allocated timeout window, then the message will be delivered to another consumer and the process will continue until the message is not deleted from the queue by a consumer.

- **Message Retention Period:** The amount of time Amazon SQS retains a message in case it is not deleted. The accepted values here are a minimum of 1 minute and a maximum of 14 days.
- **Maximum Message Size:** The maximum message size in bytes accepted by Amazon SQS. The maximum limit is 256 KB.
- **Delivery Delay:** Amazon SQS allows you to temporarily delay the delivery of new messages in a queue for a specified amount of seconds. This is achieved by placing the new messages in a **Delay queue** which is completely managed by AWS itself. Although it seems similar to the concept of Visibility Timeouts, a delay queue hides a message when it is first added to the queue, unlike the latter where the message is hidden when it is picked up by a consumer. The accepted values here are between 0 seconds and 15 minutes:

Queue Attributes

Default Visibility Timeout ⓘ

30

seconds ▾

Value must be between 0 seconds and 12 hours.

Message Retention Period ⓘ

4

days ▾

Value must be between 1 minute and 14 days.

Maximum Message Size ⓘ

256

KB

Value must be between 1 and 256 KB.

Delivery Delay ⓘ

0

seconds ▾

Value must be between 0 seconds and 15 minutes.

Receive Message Wait Time ⓘ

0

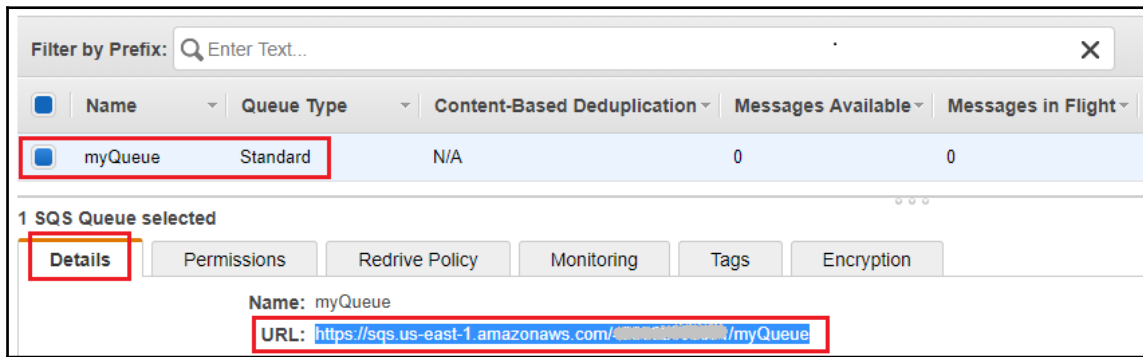
seconds

Value must be between 0 and 20 seconds.

- **Receive Message Wait Time:** Amazon SQS periodically queries a small subset of the servers to determine if any new messages are available for consumption. This method is called **short polling** and is generally enabled by default when the **Receive Message Wait Time** is set to 0. This method, however, results in a lot of empty responses as well, as sometimes messages just may not be present in the queue for consumption. In that case, SQS also provides a concept of **long polling**, whereby Amazon SQS waits until a message is available in the queue before sending a response. This drastically reduces the number of empty responses and is helpful in reducing the overall running costs of your system. To enable long polling, simply change the value of **Receive Message Wait Time** to a value between 0 and 20 seconds.

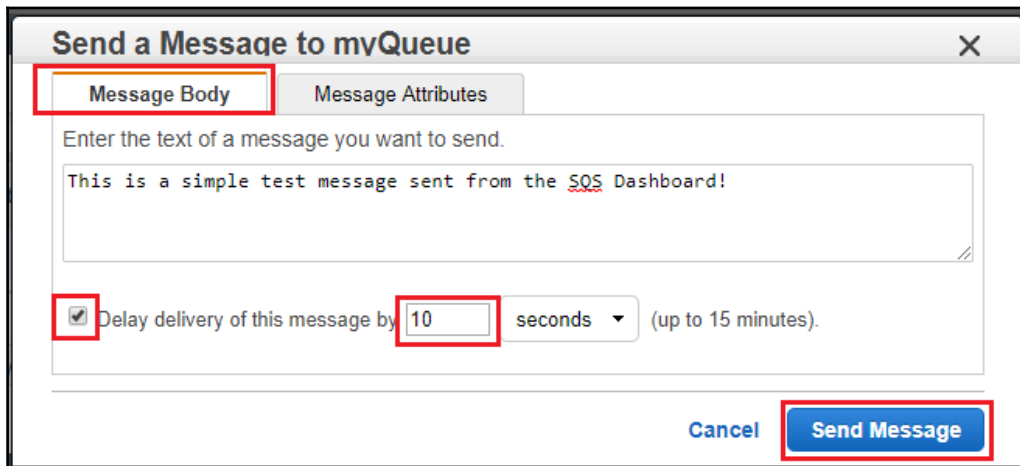
With these basic settings configured, you can now go ahead and create your very own queue. Note, however, that there are a few additional settings that you can configure, such as a **dead letter queue** and a **server-side encryption**. However, we will park these out for the time being. Select **Create Queue** once done.

With the new queue created, you can now start using it by simply copying the queue's URL (`https://sqs.us-east-1.amazonaws.com/<ACCOUNT_ID>/<QUEUE_NAME>`) and providing the same to your applications or consumers to consume from:



You can also test the functionality of your queue by sending a test message to it using the **SQS** dashboard itself. Select the newly-created queue from the **SQS** dashboard, and from the **Queue Actions** drop-down menu select the **Send a Message** option.

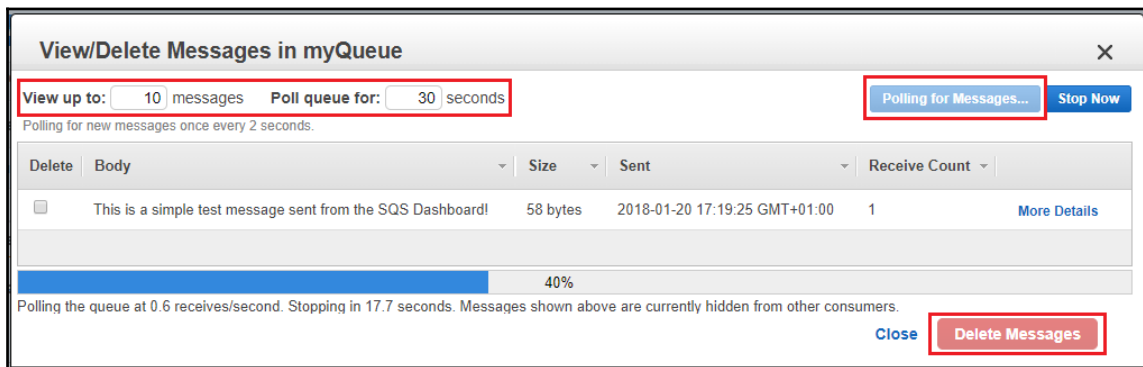
This will bring up the **Send a Message** dialog box, as shown in the following screenshot. Next, type in a test message in the **Message Body** section and click on **Send Message** to complete the process:



You can optionally also change the delivery delay of this individual message by enabling the **Delay delivery of this message by** option and providing a value between 0 and 15 minutes.

With the message sent, you will be notified of the message's *identifier* along with an *MD5 checksum* of the body. Click on **Close** to close the **Send a Message** dialog box. With this, the status of the **Messages Available** column should change to **1** as the new message is now waiting to be read or consumed. To read the message from the **SQS** dashboard, once again select the **Queue Actions** drop-down menu and select the **View/Delete Messages** option.

This brings up the **View/Delete Messages** dialog box, as shown in the following screenshot. Here, the dialog will poll the queue once every 2 seconds until you have specified the polling to run using the **Poll queue for** option. You can also change the maximum number of messages viewed by modifying the **View up to** field. Once done, select the **Start Polling for Messages** option to get things underway:



With the polling started, you should see your test message in the display area, shortly. You can also verify the validity of the message by selecting the **More Details** option adjoining the message and verifying the MD5 checksum from the earlier recorded one.

Once completed, select the message and click on the **Delete Messages** option to remove the message from the queue. Remember, this is a permanent action and it cannot be undone. With the message deleted, your queue should once again show zero messages in flight or available.

Creating a FIFO queue using the AWS CLI

Working with the AWS Management Console is easy enough, but the AWS CLI makes things even simpler! In this section, we will look at a few simple AWS CLI commands that you can use to create and work on your first FIFO queue:

1. To get started, we require a server or instance with the latest version of the AWS CLI installed and configured. If you don't already have this working, you might want to have a quick look at the detailed steps provided at <https://docs.aws.amazon.com/cli/latest/userguide/installing.html>.
2. With the AWS CLI installed and prepped, you can now use the following command to create your first FIFO queue. First, create a simple JSON file that will store the necessary list of attributes that we wish to pass to our new FIFO queue:

```
# vi fifo-queue.json
#### PASTE THE FOLLOWING CONTENTS ####
{"VisibilityTimeout" : "30",
 "MaximumMessageSize" : "262144",
 "MessageRetentionPeriod" : "345600",
 "DelaySeconds" : "10",
 "ReceiveMessageWaitTimeSeconds" : "0",
 "FifoQueue" : "true",
 "ContentBasedDeduplication" : "true"
}
```

Here, most of the values are probably known to you already, such as the `VisibilityTimeout`, the `MaximumMessageSize`, `DelaySeconds`, and so on. The two new attributes listed here specifically for the FIFO queue are:

- `FifoQueue`: Used to designate a queue as a FIFO queue. Note that you cannot change an existing standard queue to a FIFO queue. You will have to create a new FIFO queue altogether. Additionally, when you set this attribute for your queue, you must also provide the `MessageGroupId` for your messages explicitly.
- `ContentBasedDeduplication`: It enables each message to be processed exactly one time from the queue. Once `ContentBasedDeduplication` is enabled, messages with identical content sent within the deduplication interval are treated as duplicates and only one copy of the message is actually delivered.

3. Once the JSON file is created, run the following command to create your FIFO queue:

```
# aws sqs create-queue --queue-name myQueue.fifo
--attributes file://fifo-queue.json
```

You should receive the new FIFO queue's endpoint URL in the output, as shown in the following screenshot:

```
yoyo@YoYoNux:~$
yoyo@YoYoNux:~$ aws sqs create-queue --queue-name myQueue.fifo \
> --attributes file://fifo-queue.json
{
  "QueueUrl": "https://queue.amazonaws.com/12345678910/myQueue.fifo"
}
```

4. With the queue created, you can additionally use the CLI to pass messages to the queue as well. This is also accomplished by using the following command:

```
# aws sqs send-message
--queue-url https://queue.amazonaws.com/012345678910/myQueue.fifo
--message-body "Well this is far easier than I expected."
--message-group-id "R@nD0M"
```

The `send-message` command accepts the queue URL as one of the input parameters, along with the actual message that has to be sent. The message can be raw, JSON, or XML formatted. In addition to this, the `send-message` command also uses the `--message-group-id` parameter that essentially tags the message to belong to a specific message group. Messages that belong to the same message group are processed in a FIFO manner:

```
yoyo@YoYoNux:~$
yoyo@YoYoNux:~$ aws sqs send-message \
> --queue-url https://queue.amazonaws.com/4567891012345678910/myQueue.fifo \
> --message-body "Well this is far easier than I expected." \
> --message-group-id "R@nD0M"
{
  "MD5OfMessageBody": "d733b7da2656fffc18d99bea3613e24d7",
  "SequenceNumber": "18834975361266927872",
  "MessageId": "a075bd88-4942-416d-b632-0258a985a8c8"
}
```



The `--message-id-group` parameter is mandatory when working with FIFO queues.

5. With the message now sent to the queue, you can use the AWS CLI to receive the message as well. Use the following command to fetch the messages from your FIFO queue:

```
# aws sqs receive-message
--queue-url https://queue.amazonaws.com/012345678910/myQueue.fifo
```

You can also additionally use the `--max-number-of-messages` attribute to list up to 10 messages that are currently available in the queue. Here is a snippet of the output that you may get with the previous command:

```
{
  "Messages": [
    {
      "Body": "Well this is far easier than I expected.",
      "ReceiptHandle": "AQnmzJjGNrI9cl7ZyZ2NyVDDdy==",
      "MD5OfBody": "d733b7da2656ffc18d99bea3613e24d7",
      "MessageId": "a075bd88-4942-416d-b632-0258ac8"
    }
  ]
}
```

You can similarly use the AWS CLI to list the available queues in your environment, modify their parameters, push and poll for new messages, delete messages, and much more! Remember, the messages will persist in the queue unless you manually delete them or the validity of the queue's `MessageRetentionPeriod` has expired.

Integrating Amazon SNS and Amazon SQS

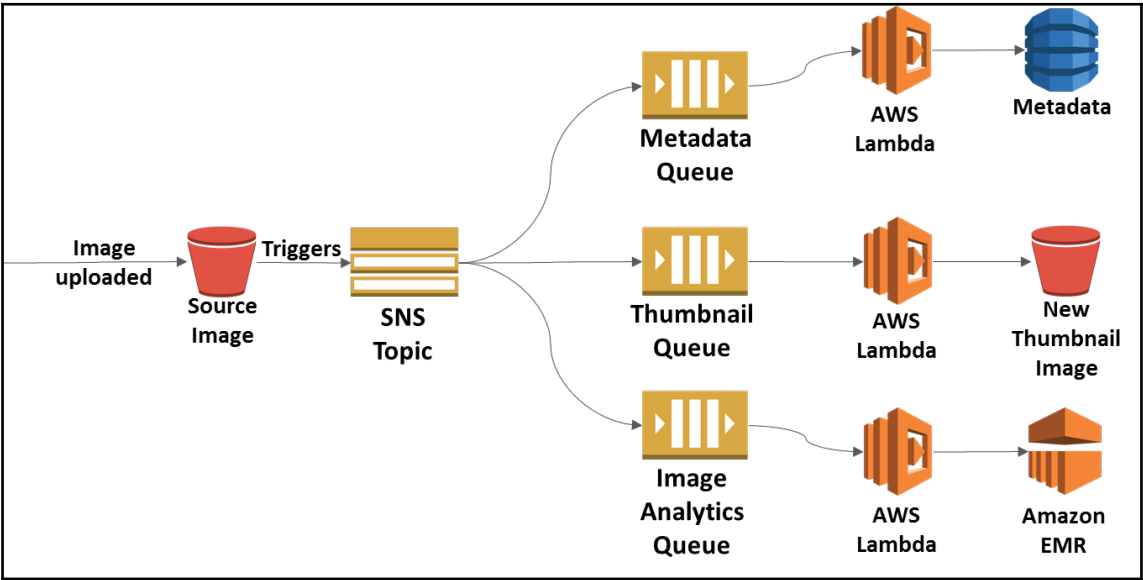
One of the key features of Amazon SQS is that it can easily be integrated with other AWS services, such as Amazon SNS. Why would I need something like that? To begin with, let us quickly recap the things we have learned so far about both SNS and SQS:

Amazon SNS	Amazon SQS
Leverages the push mechanism	Leverages the polling mechanism
Amazon SNS messages can push messages to mobile devices or other subscribers directly	Amazon SQS needs a worker to poll the messages

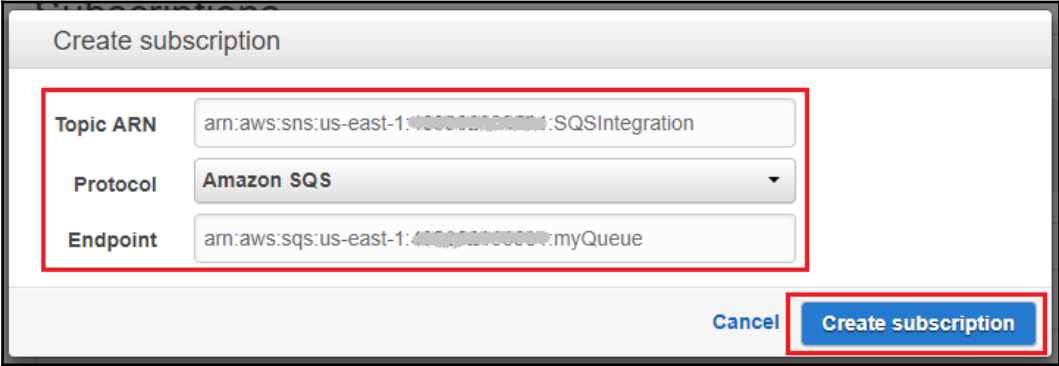
Persistence of messages is not supported	Amazon SQS supports message persistence which can come in really handy if you can't reach your consumers due to a network failure
--	---

From the table, it is easy to see that both the services offer their own pros and cons when it comes to working with them. However, when we join the two services, you can actually leverage them to design and build massively scalable yet decoupled applications. One common architectural pattern that you can leverage by combining both SNS and SQS is called the **fan out pattern**.

In this pattern, a single message published to a particular SNS topic can be distributed to a number of SQS queues in parallel. Thus, you can build highly-decoupled applications that take advantage of parallel and asynchronous processing. Consider a simple example to demonstrate this pattern. A user uploads an image to his S3 bucket, which triggers an SNS notification to be sent to a particular SNS topic. This topic can be subscribed by a number of SQS queues, each running a completely independent process from the other. For example, one queue can be used to process the image's metadata while the other can be used to resize the image to a thumbnail, and so on. In this pattern, the queues can work independently of each other without even having to worry about whether or not the other completed its processing or not. Here is a representational figure of this pattern:



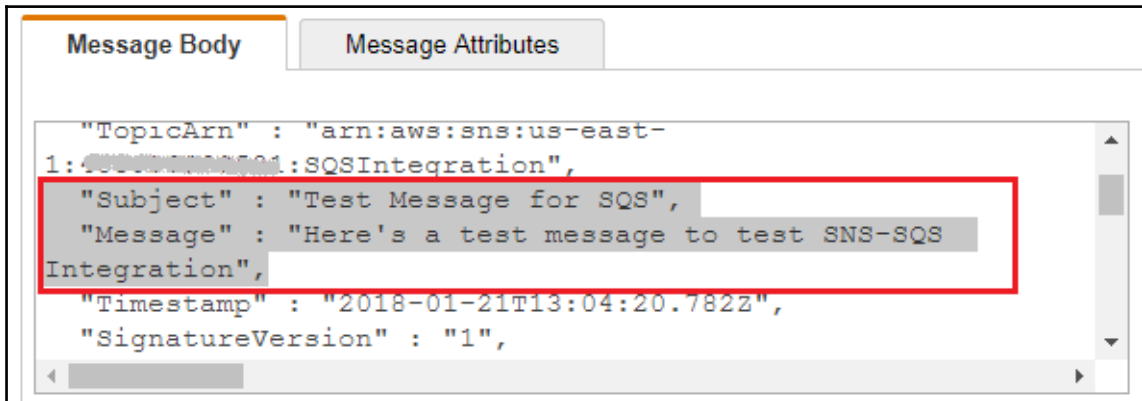
1. To integrate both the SNS and SQS services, you will first be required to create a simple SNS topic of your own. Go ahead and create a new SNS topic using the AWS Management Console, as performed earlier in this chapter.
2. Once the topic is ready, the next step involves the creation of an associated subscription. To do so, from the SNS dashboard, select the **Subscriptions** option from the navigation pane and click on **Create subscription** to get started.
3. In the **Create subscription** dialog box, copy and paste the newly created topic's ARN in the **Topic ARN** field, as shown in the following screenshot:



4. Once the **Topic ARN** is pasted, select the **Amazon SQS** option from the **Protocol** drop-down list, followed by pasting a queue's ARN in the **Endpoint** field. In this case, I'm using the standard queue's endpoint that we created a while back in this chapter.
5. With the required fields filled out, select **Create subscription** to complete the process.
6. Next, from the SQS dashboard, select the queue that you have identified for this integration and, from the **Permissions** tab, select **Add a Permission** to allow the SNS service to send messages to the queue. To do so, provide the following set of permissions:
 - **Effect:** Allow
 - **Principal:** Everybody
 - **Actions:** SendMessage

7. Once done, click on **Add Permission** to grant the SNS service the required set of permissions.
8. We are now ready to test the integration! To do so, simply fire a sample message using the **Publish to Topic** option from the SNS dashboard. Once the message is successfully sent, cross over to the SQS dashboard and poll the queue using the **View/Delete Messages** option from under the **Queue Actions** drop-down list.

Here is a snippet of the **Message Body** obtained after long polling the queue:



Similarly, you can use such a fan out pattern to design and build your very own highly scalable and decoupled cloud-ready applications.

Planning your next steps

Well, that was really quite a lot to learn and try out, but we are not done yet! There are still a few things that you ought to try on your own with SNS, as well as with SQS. First up, Amazon SNS mobile push notifications.

We have already touched upon the fact that Amazon SNS can be used to send notifications to a variety of subscribers, including HTTP, HTTPS endpoints, Amazon SQS, and AWS Lambda, but one other key feature recently added is SNS' ability to push notifications directly to your applications on mobile devices. This is called **SNS mobile push notifications** and, as of now, SNS supports the following push notification services:

- **Amazon Device Messaging (ADM)**
- **Apple Push Notification Service (APNS)** for both iOS and macOS
- **Baidu Cloud Push (Baidu)**
- **Google Cloud Messaging (GCM)** for Android
- **Microsoft Push Notification Service (MPNS)** for Windows Phone
- **Windows Push Notification Services** or **Windows Notification Service (WNS)**

It's pretty easy and straightforward to get started with mobile push notifications. All you need is a set of credentials for connecting to one of the supported push notification services, a device token or registration ID for the mobile application and device itself, and an Amazon SNS configured to send push notification messages to the mobile endpoints.



You can read more about SNS mobile push notification services at <https://docs.aws.amazon.com/sns/latest/dg/SNSMobilePush.html>.

The other important feature worth trying out is the configuration of server-side encryption for your Amazon SQS queue. You can leverage SSE to encrypt and protect data stored in your queue, however, this feature is only available in the US East (N. Virginia), US East (Ohio), and US West (Oregon) regions at present.

Encrypting the queue can be done at the time of the queue's creation, as well as after the queue has been created. Old messages present in the queue, however, are not encrypted if the SSE is switched on in an existing queue.

You can configure SSE for an existing queue simply by selecting it from the **SQS** dashboard and selecting the **Configure Queue** option present in the **Queue Actions** drop-down menu. Here, check the **Use SSE** checkbox to enable the server-side encryption on your queue. At this time, you will be prompted to select a **customer master key (CMK)** ID which you can leave to the default value if you do not have an CMK of your own. Once done, set a duration for the **Data key reuse period** of between 1 minute and 24 hours. Click on **Save changes** to apply the recent modifications to the queue.

You can read more about SSE and how to enable it on a new queue, at <https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-create-queue-sse.html>.

Last, but not the least, I also recommend that you try out the *dead letter queue* feature provided by Amazon SQS. Dead letter queues are nothing more than queues that you create for storing messages that could not be processed by your application's main processing queue. This comes in really handy when you need to debug issues in your application or the messaging system. However, it is very important to note that the dead letter queue of a standard queue is always a standard queue, and the same applies for a FIFO-based queue as well.

You can configure any queue within your account to be a dead letter queue for another queue by simply configuring the **Redrive Policy** of your application's main queue. To know more about dead letter queues and how they work, check out <https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-dead-letter-queues.html>.

Summary

Well, this has been a really insightful chapter indeed. Before we move on to the next chapter, here's a quick look at the things we have learned so far!

First up, we started with a quick look at the various messaging services that are provided by AWS. Next, we deep dived into the amazing world of Amazon SNS, created our very first topic, and subscribed to both email and phone subscriptions. We also looked at how to configure and leverage SNS' text messaging service.

Once we had the basics of SNS covered, we moved on to the next messaging service, SQS, and learned a bit about its concepts and terminologies as well. We created our first queues using both the AWS Management Console and the AWS CLI, and finally we looked at a really useful integration of the two services that you can use to design and build scalable and decoupled cloud applications. We finally topped it all off with a handy next steps guide that you ought to try out in your free time!

In the next chapter, we will be learning and exploring two really awesome analytics services, in the form of Amazon Elastic MapReduce and Amazon Redshift, so stay tuned!

9

Powering Analytics Using Amazon EMR and Amazon Redshift

In the previous chapter, we learned about two really useful services that developers can leverage to build highly scalable and decoupled applications in the cloud: Amazon SNS and Amazon SQS.

In this chapter, we will be turning things up a notch and exploring two amazingly powerful AWS services that are ideal for processing and running large-scale analytics and data warehousing in the cloud: Amazon EMR and Amazon Redshift.

Keeping this in mind, let's have a quick look at the various topics that we will be covering in this chapter:

- Understanding the AWS analytics suite of services with an in-depth look at Amazon EMR, along with its use cases and benefits
- Introducing a few key EMR concepts and terminologies, along with a quick getting started tour
- Running a sample workload on EMR, using steps
- Introducing Amazon Redshift
- Getting started with an Amazon Redshift cluster
- Working with Redshift databases and tables
- Loading data from Amazon EMR into Amazon Redshift

So without any further ado, let's get started right away!

Understanding the AWS analytics suite of services

With the growth of big data and its adoption across organizations on the rise, many cloud providers today provide a plethora of services that are specifically designed to run massive computations and analytics on large volumes of data. AWS is one such cloud provider that also has invested a lot into the big data and analytics paradigm with a host of services offering ready-to-use frameworks, business insights and data warehousing solutions, as well. Here is a brief explanation of the AWS analytics suite of services:

- **Amazon EMR:** Amazon Elastic MapReduce or EMR is a quick and easy to use service that provides users with a scalable, managed Hadoop ecosystem and framework. You can leverage EMR to process vast amounts of data without having to worry about configuring the underlying Hadoop platform. We will be learning and exploring more on EMR in the subsequent sections of this chapter.
- **Amazon Athena:** Amazon Athena takes big data processing up a notch by providing a standard SQL interface for querying data that is stored directly on Amazon S3. With Athena, you do not have any underlying hardware to manage or maintain; it is all managed by AWS itself. This *serverless* approach makes Athena ideal for processing data that does not require any complex ETL processing. All you need to do is create a schema, point Athena to your data on Amazon S3, and start querying it using simple SQL syntax.
- **Amazon Elasticsearch Service:** Amazon Elasticsearch Service provides a managed deployment of the popular open source search and analytics engine: Elasticsearch. This service comes in really handy when you wish to process streams of data originating from various sources such as logs generated from instances, and so on.
- **Amazon Kinesis:** Unlike the other services discussed so far, Amazon Kinesis is more of a streaming service provided by AWS. You can use Amazon Kinesis to push vast amounts of data originating from multiple sources, into one or more streams that can be consumed by other AWS services for performing analytics and other data processing processes.
- **Amazon QuickSight:** Amazon QuickSight is an extremely cost-effective business insights solution that can be used to perform fast ad hoc analysis on data.
- **Amazon Redshift:** Amazon Redshift is a petabyte-scale data warehousing solution provided by AWS that you can leverage for analyzing your data, using an existing set of tools. We will be learning more about Redshift a bit later during this chapter. The services are depicted here:

COLLECT	MOVE	STORE	ANALYZE
 AWS Snowball  Amazon Kinesis	 AWS Data Pipeline  Amazon Kinesis	 Amazon S3  Amazon EFS  Amazon DynamoDB	 Amazon EMR  Amazon ES  Amazon Athena  AWS Glue  Amazon Redshift  Amazon QuickSight

- **AWS Data Pipeline:** Moving large amounts of data between AWS services can be difficult to perform, especially when the data sources vary. AWS Data Pipeline makes it easier to transfer data between different AWS storage and compute services, as well as helping in the initial transformation and processing of data. You can even use Data Pipeline to transfer data reliably from an on-premise location into AWS storage services, as well.
- **AWS Glue:** AWS Glue is a managed **ETL (Extract, Transform and Load)** service recently launched by AWS. Using AWS Glue greatly simplifies the process of preparing, extracting, and loading data from large datasets into an AWS storage service.

With this brief overview of the AWS analytics suite of services, let's now move forward and get started with understanding a bit more about Amazon EMR!

Introducing Amazon EMR

As mentioned earlier, Amazon EMR is a managed service that provides big data analytics frameworks, such as Apache Hadoop and Apache Spark straight out of the box and ready for use. Using Amazon EMR, you can easily perform a variety of use cases such as batch processing, big data analytics, low-latency querying, data streaming, or even use EMR as a large datastore itself!

With Amazon EMR, there is very little underlying infrastructure to manage on your part. You simply have to decide the number of instances you initially want to run your EMR cluster on and start consuming the framework for analytics and processing. Amazon EMR provides you with features that enable you to scale your infrastructure based on your requirements, without affecting the existing setups. Here is a brief look at some of the benefits that you can obtain by leveraging Amazon EMR for your own workloads:

- **Pricing:** Amazon EMR relies on EC2 instances to spin up your Apache Hadoop or Apache Spark clusters. Although you can vary costs by selecting the instance types for your cluster from large to extra large and so on, the best part of EMR is that you can also opt between using a combination of on-demand EC2 instances, reserved and spot instances based on your setup, thus providing you with flexibility at significantly lower costs.
- **Scalability:** Amazon EMR provides you with a simple way of scaling running workloads, depending on their processing requirements. You can resize your cluster or its individual components as you see fit and additionally, configure one or more instance groups for a guaranteed instance availability and processing.
- **Reliability:** Although you, as an end user, have to specify the initial instances and their sizes, AWS ultimately ensures the reliability of the cluster by swapping out instances that either have failed or are going to in the due course of time.
- **Integration:** Amazon EMR integrates with the likes of other AWS services to provide your cluster with additional storage, network, and security requirements. You can use services such as Amazon S3 to store both the input as well as the output data, AWS CloudTrail for auditing the requests made to EMR, VPC to ensure the security of your launched EMR instances and much more!

With these details in mind, let's move an inch closer to launching our very own EMR cluster by first visiting some of its key concepts and terminologies.

Concepts and terminologies

Before we get started with Amazon EMR, it is important to understand some of its key concepts and terminologies, starting out with clusters and nodes:

- **Clusters:** Clusters are the core functioning component in Amazon EMR. A cluster is a group of EC2 instances that together can be used to process your workloads. Each instance within a cluster is termed as a node and each node has a different role to perform within the cluster.
- **Nodes:** Amazon EMR distinguishes between clusters instances by providing them with one of these three roles:
 - **Master node:** An instance that is responsible for the overall manageability, working and monitoring of your cluster. The *master node* takes care of all the data and task distributions that occur within the cluster.
 - **Core node:** The core nodes are very similar to the master node; however, they are primarily used to run tasks and store data on your **Hadoop Distributed File System (HDFS)**. The core node can also contain some additional software components of Hadoop applications within itself.
 - **Task node:** Task nodes are only designed to run tasks. They do not contain any additional software components of Hadoop applications within themselves and are optional when it comes to the cluster's deployment.
- **Steps:** Steps are simple tasks or jobs that are submitted to a cluster for processing. Each step contains some instructions on how the particular job is to be performed. Steps can be ordered such that a particular step can be used to fetch the input data from Amazon S3, while a second step can be used to run a Pig or Hive query against it, and finally a third step to store output data to say Amazon DynamoDB. If one step fails, the subsequent steps are automatically cancelled from execution, however, you can choose to overwrite this behavior by selecting your steps to ignore failures and process further.

Apart from these concepts, you will additionally be required to brush up on your Apache Hadoop framework and terminologies, as well. Here's a quick look at some of the Apache frameworks and applications that you will come across while working with Amazon EMR:

- **Storage:** A big part of EMR is how the data is actually stored and retrieved. The following are some of the storage options that are provided to you while using Amazon EMR:
 - **Hadoop Distributed File System (HDFS):** As the name suggests, HDFS is a distributed and scalable filesystem that allows data to be stored across the underlying node instances. By default, the data is duplicated and stored across the instances present in the cluster. This provides high availability and data resiliency in case of an instance failure. You can read more about HDFS at: <https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html>.
 - **EMR File System (EMRFS):** EMRFS is an extension of the HDFS filesystem, using which you can access and store data directly on Amazon S3, just as a normal filesystem.
 - **Local filesystem:** Apart from HDFS, each instance within the cluster is also provided with a small block of pre-attached ephemeral disks which is also called the local filesystem. You can use this local filesystem to store additional software or applications required by your Hadoop frameworks.
- **Frameworks:** As mentioned before, Amazon EMR provides two data processing frameworks that you can leverage based on your processing needs: Apache Hadoop MapReduce and Apache Spark:
 - **Apache Hadoop MapReduce:** MapReduce is by far the most commonly used and widely known programming model when it comes to building distributed applications. The open source model relies on a `Mapper` function that maps the data to sets of key-value pairs and a `Reducer` function that combines these key-value pairs, applies some additional processing, and finally generates the desired output. To know more about MapReduce and how you can leverage it, check out this URL: https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html.

- **Apache Spark:** Apache Spark is a fast, in-memory data processing model using which a developer can process streaming, machine learning or SQL workloads that require fast iterative access to datasets. It is a cluster framework similar to Apache Hadoop; however, Spark leverages graphs and in-memory databases for accessing your data. You can read more about Spark at <https://spark.apache.org/>.
- **Applications and programs:** With the standard data processing framework, Amazon EMR also provides you with additional applications and programs that you can leverage to build native distributed applications. Here's a quick look into a couple of them:
 - **YARN: Yet Another Resource Negotiator**, is a part of the Hadoop framework and provides management for your cluster's data resources
 - **Hive:** Hive is a distributed data warehousing application that leverages standard SQL to query extremely large datasets stored on the HDFS filesystem.

There are yet many other applications and programs made available for use by Amazon EMR, such as Apache Pig, Apache HBase, Apache Zookeeper, and so on. In the next section, we will be looking at how to leverage these concepts and terminologies to create our very own Amazon EMR Cluster, so let's get busy!

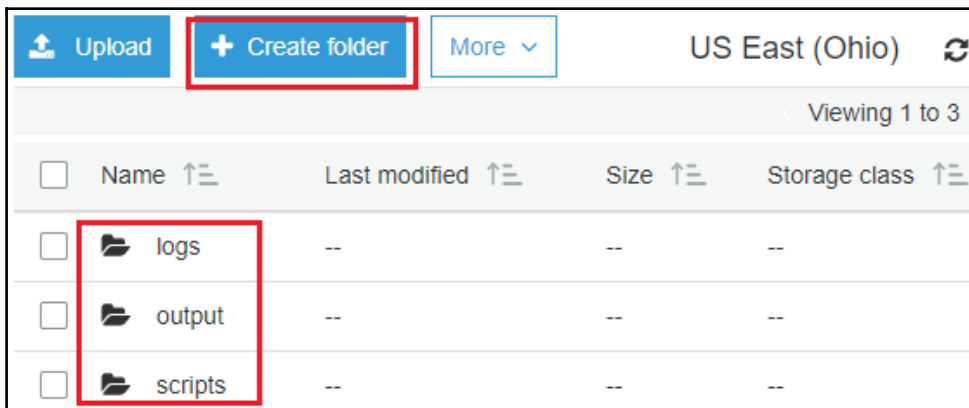
Getting started with Amazon EMR

With the basics covered, in this section we will be working with the Amazon EMR dashboard to create our very first cluster. However, before we get going, here's a small list of prerequisite steps that we need to complete first.

To begin with, we will need to create an Amazon S3 bucket that will be used to store the output, logs generated by EMR, as well as some additional script and software files:

1. From the AWS Management Console, filter and select the **Amazon S3** service by using the **Filter** option. Alternatively, launch the Amazon S3 dashboard by navigating to this URL: <https://s3.console.aws.amazon.com/s3/>.

2. Next, select the **Create bucket** option. In the **Create bucket** wizard, provide a suitable **Bucket name** followed by the selection of an appropriate **Region** to create the bucket in. For this use case, the EMR cluster, as well as the S3 buckets, are created in the **US East (Ohio)** region, however you can select an alternative based on your requirements. Click on **Next** to continue with the process.
3. On the **Set properties** page, you can optionally choose to provide some *tags* for your bucket for cost allocations and tracking purposes. Click **Next** to continue.
4. In the **Set permissions** page, ensure that the no public read access is granted to the bucket. Click on **Next** to review the settings and finally, select **Create bucket** to complete the process.
5. Once the bucket is created, use the **Create folder** option to create dedicated folders for storing the logs, output, as well as some additional scripts that we might use in the near future. Here is a representational screenshot of the bucket after you have completed all of the previous steps:



6. With the bucket created and ready for use, the next prerequisite item left to create is a key pair using which you can SSH into your EC2 instances. Ensure that the key pair is created in the same region (**US East (Ohio)** in this case) as your EMR cluster.

Now that the prerequisites are out of the way, we can finally get started with our EMR cluster setup!

1. From the AWS Management Console, filter and select the **Amazon EMR** service by using the **Filter** option. Alternatively, launch the Amazon EMR dashboard by selecting this URL: <https://us-east-2.console.aws.amazon.com/elasticmapreduce/home>.

2. Since this is the first time we've created an EMR cluster, select the **Create cluster** option to get started.
3. You can configure your EMR cluster using two ways: a fast and easy **Quick Options** which is shown to you by default, and an **Advanced options** page where you can select and configure the individual items for your cluster. In this case, we will go ahead and select **Go to advanced options**.
4. The Advanced options page provides us with a four-step wizard that essentially guides us to configuring a fully functional EMR cluster. To begin with, the first step is where you can select and customize the *software* that you wish to install on your EMR cluster.
5. From the **Release** drop-down list, select the appropriate EMR release that you would like to work with. The latest version released as of writing this book is `emr-5.11.1`. Each release contains several distributed applications available for installation on your cluster. For example, selecting `emr-5.11.1` which is a 2018 release, contains Hadoop v2.7.3, Flink v1.3.2, Ganglia v3.7.2, HBase v1.3.1, and many other such applications and software.



For a complete list of available EMR releases and their associated software versions, go to <https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-release-components.html>.

6. In this case, I have gone ahead and selected the basic applications that we will be requiring for this scenario, including Hadoop, Hive and Hue. Feel free to select other applications as per your requirements.
7. The next couple of sections are optional, however, it is important to know their purpose:
 - **AWS Glue Data Catalog settings:** With EMR version 5.8.0 and above, you optionally have the choice to configure Spark SQL to use the AWS Glue Data Catalog (an external Hive table) as its metastore.

- **Edit software settings:** You can use this option to override the default configuration settings for certain applications. This is achieved by providing a configuration object in the form of a JSON file. You can either **Enter configuration** or **Load JSON from S3** as well:

Create Cluster - Advanced Options [Go to quick options](#)

Software Configuration

Release emr-5.11.1

☒ Hadoop 2.7.3	☐ Zeppelin 0.7.3	☐ Livy 0.4.0
☐ Tez 0.8.4	☐ Flink 1.3.2	☐ Ganglia 3.7.2
☐ HBase 1.3.1	☐ Pig 0.17.0	☒ Hive 2.3.2
☐ Presto 0.187	☐ ZooKeeper 3.4.10	☐ MXNet 0.12.0
☐ Sqoop 1.4.6	☐ Mahout 0.13.0	☒ Hue 4.0.1

AWS Glue Data Catalog settings (optional)

☐ Use for Hive table metadata

Edit software settings (optional)

☒ Enter configuration ☐ Load JSON from S3

```
classification=config-file-name,properties=[myKey1=myValue1,myKey2=myValue2]
```

- **Add steps:** The final optional parameter left on the **Software Configuration** page is the *add steps*. As discussed briefly earlier in this chapter, steps are essentially a unit of work that we submit to the cluster. This can be something as trivial as loading input data from S3, or processing and running a MapReduce job on the data. We will be exploring steps a little more in detail a bit later in this chapter, so leave this field to its default value and select **Next** to continue with the process.
8. The second step in the **Advanced options** wizard is configuring the cluster's hardware, or the instance configurations, as well as the cluster's networking.

EMR provides two options: instance fleets and instance groups; both explained briefly here:

- **Instance fleets:** Instance fleets allows you to specify a target capacity for the instances present in a cluster. With this option, you get the widest variety of instance provisioning options where you can leverage mixed instance types for your nodes, and even go for different purchasing options for the same. With each instance fleet that created, you get to establish a target capacity for on-demand, as well as for spot instances.



You can have only one instance fleet per node type (master, core, task).

- **Instance groups:** Instance groups on the other hand do not offer many custom configurable options per node type. In instance groups, each node consists of the same instance type and the same purchasing option, as well. Once these settings are configured during the cluster's creation, they cannot be altered; however, you can always add more instances as you see fit.

9. For this particular use case, we are going to go ahead and select **Uniform instance groups**, as depicted in the following screenshot:

Hardware Configuration ⓘ

If you need more than 20 EC2 instances, [see this topic](#).

Instance group configuration

- ☒ **Uniform instance groups**
Specify a single instance type and purchasing option for each node type.
- ☐ **Instance fleets**
Specify target capacity and how Amazon EMR fulfills it for each node type. Mix instance types and purchasing options. [Learn more](#)

Network vpc-2de01844 (172.31.0.0/16) (default) [Create a VPC](#) ⓘ

EC2 Subnet subnet-adf932c4 | Default in us-east-2a

Root device EBS volume size 10 GiB ⓘ

10. Next, from the **Network** drop-down list, select the appropriate VPC in which you wish to launch your EMR cluster. You can alternatively choose to create a new VPC specifically for EMR, using the adjoining **Create a VPC** option.
11. Similarly, select the appropriate subnet from the **EC2 Subnet** drop-down list.
12. Finally, assign a value for the **Root device EBS volume size** that will be provisioned for each instance in the cluster. You can provide values between 10 GB and 100 GB.
13. Using the edit options provided, you can additionally configure the **Instance type**, the **Instance count** as well as the **Purchasing option** for each node type, as depicted in the following screenshot. Note that these options are provided because we selected instance groups as our preferred mode of instance configurations. The options will vary if the **Instance Fleet** option is selected:

Node type	Instance type	Instance count	Purchasing option	Auto Scaling
Master Master - 1	m4.large 4 vCPU, 8 GiB memory, EBS only storage EBS Storage: 32 GiB	1 Instances	On-demand Spot Maximum Spot price: \$	Not available for Master
Core Core - 2	m4.large 4 vCPU, 8 GiB memory, EBS only storage EBS Storage: 32 GiB	1 Instances	On-demand Spot Maximum Spot price: \$ 0.024	Not enabled
Task Task - 3	m4.large 4 vCPU, 8 GiB memory, EBS only storage EBS Storage: 32 GiB	1 Instances	On-demand Spot Maximum Spot price: \$ 0.024	Not enabled

+ Add task instance group

Cancel Previous Next

14. You can additionally choose to enable autoscaling for the **Core** and **Task** nodes by selecting the **Not enabled** option under the **Auto scaling** column. Subsequently, you can add additional task instance groups by selecting the **Add task instance group** option, as well. Once done, select the **Next** option to proceed with the set up.
15. The third step in the **Advanced options** provides general configurations that you can set, based on your requirements. To start off, provide a suitable **Cluster name** followed by selecting the **Logging** option for your EMR cluster. Use the folder option to browse to our newly created S3 bucket, as shown in the following screenshot:

Create Cluster - Advanced Options [Go to quick options](#)

Step 1: Software and Steps

Step 2: Hardware

Step 3: General Cluster Settings

Step 4: Security

General Options

Cluster name

☒ Logging ⓘ

S3 folder

☒ Debugging ⓘ

☒ Termination protection ⓘ

16. You can additionally enable the **Termination protection** option to prevent against accidental deletions of your cluster.
17. Moving on, the final configuration item left on the cluster's **General Options** page is the **Bootstrap Actions**. Bootstrap actions as the name implies are certain scripts or code that you wish to execute on your cluster's instances at the time of booting up. This feature thus comes in very handy when you have to add new instances to an existing running cluster.

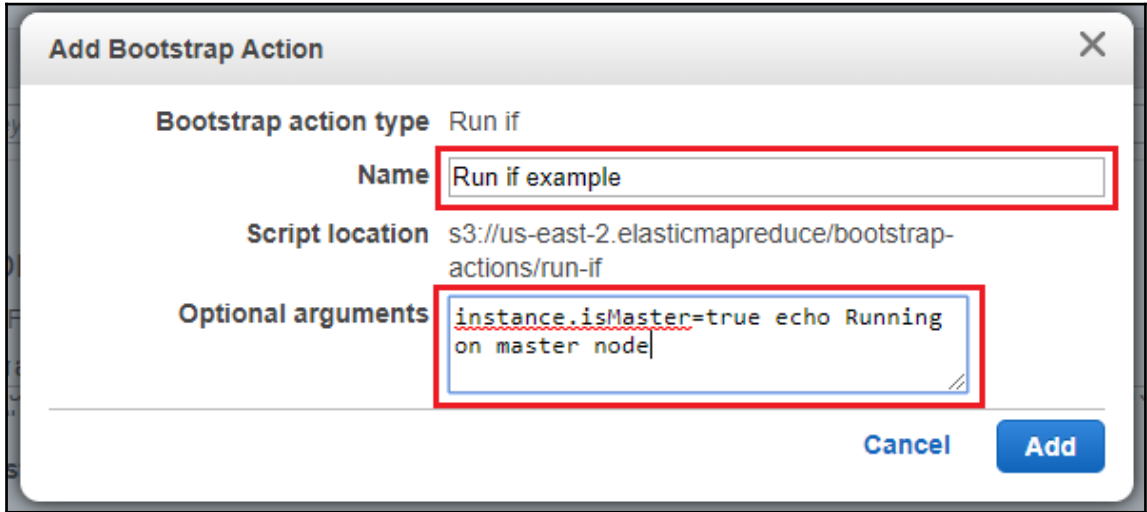


Bootstrap actions are executed using the Hadoop user by default. You can switch to root privileges by using the `sudo` command.

There are two types of Bootstrap actions that you can execute on your instances:

- **Run if:** The **Run if** action executes an action when an *instance-specific* value is found in either the `instance.json` or the `job-flow.json` file. This is a predefined bootstrap action and comes in very handy when you only want to execute the action on a particular type of instance, for example, execute the bootstrap action only if the instance type is `master`.
 - **Custom action:** Custom actions leverage your own scripts to perform a customized bootstrap action.
18. To create a bootstrap action, select the **Configure and add** option from the **Add Bootstrap Action**. Make sure the **Run if** action is selected before proceeding.

19. This will bring up the **Add Bootstrap Action** dialog as depicted in the following screenshot. Type in a suitable **Name** for your **Run if** action. Since the **Run if** action is a predefined bootstrap action, the script's location is not an editable field. You can, however, add **Optional arguments** for the script, as shown here. In this case, the **Run if** action will only echo the message if the instance is a **master**:



Add Bootstrap Action

Bootstrap action type **Run if**

Name **Run if example**

Script location **s3://us-east-2.elasticmapreduce/bootstrap-actions/run-if**

Optional arguments **instance.isMaster=true echo Running on master node**

Cancel **Add**

20. Click on **Add** once done. Similarly, you can add your custom bootstrap actions as well, by placing the executable scripts in the Amazon S3 bucket that we created during the prerequisite phase of this chapter and providing that path here.
21. Moving on to the final step in this cluster creation process, on the **Security Options** page, you can review the various permissions, roles, authentication, and encryption settings that the cluster will use once it's deployed. Start off by selecting the **EC2 key pair** that we created at the start of this chapter. You can additionally opt to change the **Permissions** or use the default ones provided.
22. Once done, click on **Create cluster** to complete the process.

The cluster's creation takes a couple of minutes, depending on the number of instances selected for the cluster, as well as the software identified to be installed. Once done, you can use the EMR dashboard to view the cluster's health status and other vital information.

Connecting to your EMR cluster

Once you have provisioned the EMR cluster, you should see its state change from **Starting** to **Bootstrapping** to finally into a **Running** state. If you do not have any jobs currently executing, then your cluster may go into a **Waiting** state as well. Here, you can now start using the EMR cluster for running your various jobs and analysis. But before that, here's a quick introduction of a few ways in which you can connect to your running EMR cluster.

First up, connecting to the master node using a simple SSH. Connecting to the master node via SSH can be used for monitoring the cluster, viewing Hadoop's log files or for even running an interactive shell for Hive or Pig programming:

1. To do so, log in to your Amazon EMR dashboard and select your newly created cluster's name from the **Cluster list** page. This will display the clusters **Details** page where you can manage, as well as monitor your cluster.
2. Next, copy the **Master public DNS** address. Once copied, open up a PuTTY Terminal and paste the copied public DNS in the **Host Name (or IP Address)** field.
3. Convert the key pair that you associated with this EMR cluster into a private key and attach that private key in PuTTY by selecting the **Auth** option present under the **SSH** section.
4. Once done, click on **Open** to establish the connection. At the certificate dialog, accept the certificate and type in `hadoop` as the username when prompted. You should get SSH access into your cluster's master node now!

The same task can be performed using the AWS CLI as well:

1. From the Terminal, first type in the following command to retrieve the running cluster's ID. The cluster's ID will be in this format `j-xxxxxxxx`:
2. To list the instances running in your cluster, use the cluster ID obtained from the previous command's output in the following command:

```
# aws emr list-clusters
```

```
# aws emr list-instances --cluster-id <CLUSTER_ID>
```



Copy the `PublicDnsName` value from the output of this command. You can then use the following set of commands to get access to your master node.

3. Ensure that the cluster's private key has the necessary permissions:

```
# chmod 400 <PRIVATEKEY.pem>
```

4. Once done, SSH to the master node using the following command:

```
# ssh hadoop@<PUBLIC_DNS_NAME> -i <PRIVATEKEY.pem>
```

You can additionally connect to the various application web interfaces, such as *Hue* or the *Hadoop HDFS NameNode*, using a few simple steps:

1. To get started, you will once again require the public DNS name of your master node. You can obtain that from the EMR dashboard or by using the CLI steps we just walked through.
2. Next, using PuTTY, paste the public DNS name in the **Host Name (or IP Address)** field as done earlier. Browse and load the private key using the **Auth** option as well.
3. Under the **SSH** option from PuTTY's navigation pane, select **Tunnels**.
4. Fill in the required details as mentioned in the following list:
 - Set source port field to 8157
 - Enable the **Dynamic** and **Auto** options
5. Once completed, select **Add** and finally **Open** the connection.

This form of tunnelling or port forwarding is essential as the web interfaces can only be viewed from the master node's local web server. Once completed, launch your favorite browser and view the respective web interfaces, as given here:

- For accessing Hue, type in the following in your web browser:

```
http://<PUBLIC_DNS_NAME>:8888/
```

- For accessing the Hadoop HDFS NameNode, type in the following:

```
http:// <PUBLIC_DNS_NAME>:50070/
```

Hadoop	Overview	Datanodes	Datanode Volume Failures	Snapshot	Startup Progress	Utilities ▾
Overview 'ip-172-31-6-217.us-east-2.compute.internal:8020' (active)						
Started:		Wed Jan 24 09:48:40 UTC 2018				
Version:		2.7.3-amzn-6, rfc548a0642e795113789414490c9e59e6a8b91e4				
Compiled:		2017-12-13T22:46Z by ec2-user from (HEAD detached at fc548a0642)				
Cluster ID:		CID-8ddbd912-4b42-431e-bf66-0af89b3b2fe5				
Block Pool ID:		BP-1404246105-172.31.6.217-1516787314975				

You can even use the CLI to create a tunnel. To do so, substitute the public DNS name and the private key values in the following command:

```
# ssh -i <PRIVATEKEY.pem> -N -D 8157 hadoop@<PUBLIC_DNS_NAME>
```



The `-D` flag indicates that the port forwarding is dynamic.

Running a job on the cluster

With the connectivity established, you can now execute jobs as one or more steps on your cluster. In this section, we will be demonstrating the working of a step using a simple example which involves the processing of a few Amazon CloudFront logs. The details of the sample data and script can be found at: <https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-gs-prepare-data-and-script.html>. You can use similar techniques and bases to create and execute your own jobs as well:

1. To get started with a job, from the EMR dashboard select your cluster's name from the **Cluster list** page. This will bring up the newly created clusters details page. Here, select the **Steps** tab.

2. Since this is going to be our first step, go ahead and click on the **Add step** option. This brings up the **Add step** dialog as shown in the following screenshot. Fill in the required information as described and, once all the fields are filled in, click on **Add** to complete the step's creation:

Add step

Step type Hive program

Name MySampleHiveProgram

Script S3 location* s3://us-east-2.elasticmapreduce.samples/cloudfront/co S3 location of your Hive script.
s3://<bucket-name>/<path-to-file>

Input S3 location s3://us-east-2.elasticmapreduce.samples S3 location of your Hive input files.
s3://<bucket-name>/<folder>/

Output S3 location s3://mapreduce-loganalysis-us-east-2/output/ S3 location of your Hive output files.
s3://<bucket-name>/<folder>/

Arguments -hiveconf
hive.support.sql11.reserved.keywords
=false Specify optional arguments for your script.

Action on failure Continue What to do if the step fails.

Cancel Add

- **Step type:** You can choose between various options such as **Streaming program** which essentially will prompt you to provide Mapper and Reducer function details, or alternatively, you can also select **Hive program**, **Pig program**, **Spark program** or a **Custom application**. In this case, we select the **Hive program** option.
- **Name:** A suitable name for your step.
- **Script S3 location:** Provide the Hive script's location here. Since we are using a predefined script, simply replace the <REGION> field with your EMR's operating region:
s3://<REGION>.elasticmapreduce.samples/cloudfront/co
de/Hive_CloudFront.q.

- **Input S3 location:** Provide the input data file's location here. Replace the <REGION> placeholder with your EMR's operating region as done before:
`s3://<REGION>.elasticmapreduce.samples.`
- **Output S3 location:** Specify where the processed output files have to be stored. In this case, I'm using the custom S3 bucket that we created as a prerequisite step during the EMR cluster creation. You can provide any other alternative bucket as well.
- **Arguments:** You can use this field to provide any optional arguments required by the script to run. In this case, copy, and paste the following `-hiveconf`
`hive.support.sql11.reserved.keywords=false.`
- **Action on failure:** You can optionally choose what EMR should do in case the step's execution undergoes a failure. In this case, we have selected the default **Continue** value.

3. Once the required fields are filled out, click on **Add** to complete the process.

The step now starts executing the supplied script on the EMR cluster. You can view the progress by viewing the changes in the step's status from **Pending** to **Running** to **Completed**, as shown in the following screenshot:

	ID	Name	Status	Start time (UTC+1) ▼	Elapsed time
☐	s-10VKSFGZDRFVW	MySampleHiveProgram	Completed	2018-01-24 11:18 (UTC+1)	1 minute
JAR location : command-runner.jar Main class : None <code>hive-script --run-hive-script --args -f s3://us-east-2.elasticmapreduce.samples/cloudfront/code/Hive_Cloud</code> Arguments : OUTPUT=s3://mapreduce-loganalytics-us-east-2/output/ -hiveconf hive.support.sql11.reserved.keywords Action on failure: Continue					

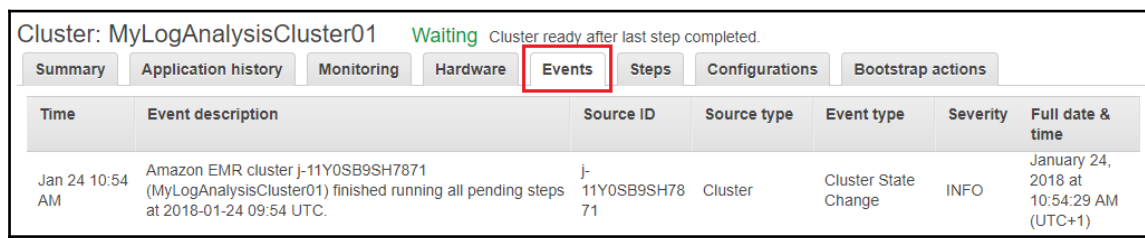
Once the job completes its execution, head back to your Amazon S3's output bucket and view the output of the processing. In this case, the output contains the number of access requests made to CloudFront, sorted by the operating system.

Monitoring EMR clusters

The EMR dashboard provides a rich feature set using which you can manage and monitor your EMR clusters all from one place. You can additionally view logs and leverage Amazon CloudWatch as well to track the performance of your cluster.

In this section, we will be looking at a few simple ways using which you can monitor your EMR clusters. To start off, let's look at how to monitor the status of your cluster using the EMR dashboard:

1. From the EMR dashboard, select your cluster name from the cluster list page. This will bring up the newly created cluster's details page. Here, select the **Events** tab, as shown in the following screenshot:



Cluster: MyLogAnalysisCluster01 Waiting Cluster ready after last step completed.							
Summary	Application history	Monitoring	Hardware	Events	Steps	Configurations	Bootstrap actions
Time	Event description	Source ID	Source type	Event type	Severity	Full date & time	
Jan 24 10:54 AM	Amazon EMR cluster j-11Y0SB9SH7871 (MyLogAnalysisCluster01) finished running all pending steps at 2018-01-24 09:54 UTC.	j-11Y0SB9SH7871	Cluster	Cluster State Change	INFO	January 24, 2018 at 10:54:29 AM (UTC+1)	

The **Events** tab allows you to view the event logged by your cluster. You can use this to view events generated by the cluster, by running applications, by step execution and much more.

2. The dashboard also provides an in-depth look into the performance of the cluster over a period. To view the performance indicators, select the **Monitoring** tab from the cluster's **Details** page.

Here, you can view essential details and status about your cluster, the running nodes, as well as the underlying I/O and data storage.

3. Alternatively, you can also use Amazon CloudWatch to view and monitor the cluster's various metrics. To do so, launch the Amazon CloudWatch dashboard by selecting this URL: <https://console.aws.amazon.com/cloudwatch/home>.

- Next, from the navigation pane, select the **Metrics** option to view all the metrics associated with EMR. Use the `JobFlowID` dimension to filter the EMR cluster in case you have multiple clusters running in the same environment.

Here is a list of some important EMR metrics worth monitoring:

Metric name	Metric description
AppsFailed	The number of applications submitted to the EMR cluster that have failed to complete. This application status is monitored internally and reported by YARN.
MRUnhealthyNodes	The number of nodes available to MapReduce jobs marked in an UNHEALTHY state.
MRLostNodes	The number of nodes allocated to MapReduce that have been marked in a LOST state.
CorruptBlocks	The number of blocks that HDFS reports as corrupted.



You can view the complete list of monitored metrics at: https://docs.aws.amazon.com/emr/latest/ManagementGuide/UsingEMR_ViewingMetrics.html.

- Once a Metric is identified, select the **Metric** and click on the **Graphed metrics** tab. Here, select the **Create alarm** option provided under the **Actions** column to create and set an alarm threshold, as well as its corresponding action.

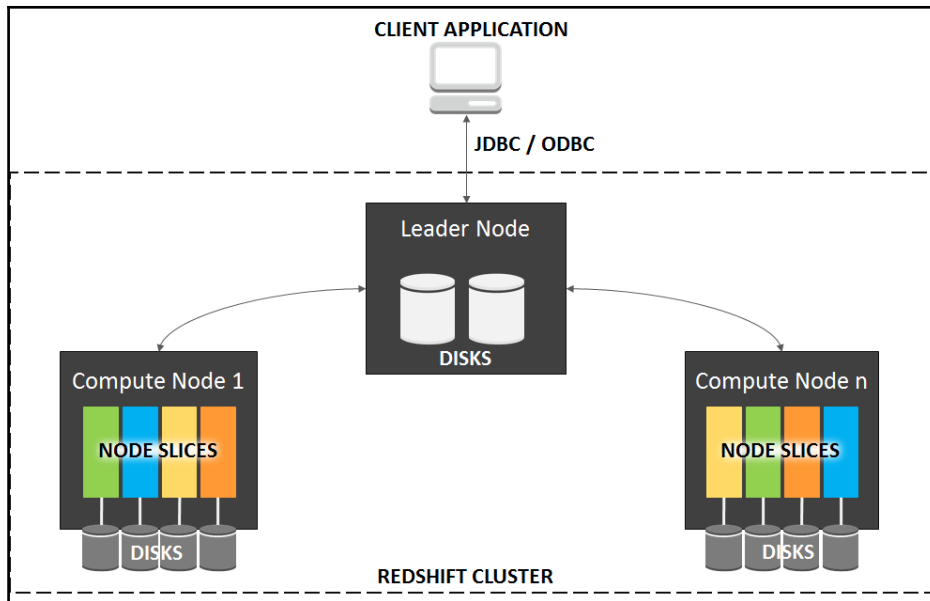
In this way, you can also leverage Amazon CloudWatch events to periodically monitor the events generated by the cluster. Remember, however, that EMR tracks and records events only for a period of seven days. With this, we come to the end of this particular section and EMR, as well. In the next section, we will be learning and exploring a bit about yet another awesome analytics service called Amazon Redshift!

Introducing Amazon Redshift

Amazon Redshift is one of the **database as a service (DBaaS)** offerings from AWS that provides a massively scalable data warehouse as a managed service, at significantly lower costs. The data warehouse is based on the open source PostgreSQL database technology however; not all features offered in PostgreSQL are present in Amazon Redshift. Here's a look at some of the essential concepts and terminologies that you ought to keep in mind when working with Amazon Redshift:

- **Clusters:** Just like Amazon EMR, Amazon Redshift too relies on the concept of clusters. Clusters here are logical containers containing one or more instances or compute nodes, and one leader node that is responsible for the cluster's overall management. Here's a brief look at what each node provides:
 - **Leader node:** The leader node is a single node present in a cluster that is responsible for orchestrating and executing various database operations, as well as facilitating communication between the database and associate client programs.
 - **Compute node:** Compute nodes are responsible for executing the code provided by the leader node. Once executed, the compute nodes share the results back to the leader node for aggregation. Amazon Redshift supports two types of compute nodes: dense storage nodes and dense compute nodes. The dense storage nodes provide standard hard disk drives for creating large data warehouses; whereas, the dense compute nodes provide higher performance SSDs. You can start off by using a single node that provides 160 GB of storage and scale up to petabytes by leveraging one or more 16 TB capacity instances as well.
- **Node slices:** Each compute node is partitioned into one or more smaller chunks or slices by the leader node, based on the cluster's initial size. Each slice contains a portion of the compute nodes memory, CPU and disk resource, and uses these resources to process certain workloads that are assigned to it. The assignment of workloads is again performed by the leader node.
- **Databases:** As mentioned earlier, Amazon Redshift provides a scalable database that you can leverage for a data warehouse, as well as analytical purposes. With each cluster that you spin in Redshift, you can create one or more associated databases with it. The database is based on the open source relational database PostgreSQL (v8.0.2) and thus, can be used in conjunction with other RDBMS tools and functionalities. Applications and clients can communicate with the database using standard PostgreSQL JDBC and ODBC drivers.

Here is a representational image of a working data warehouse cluster powered by Amazon Redshift:



With this basic information in mind, let's look at some simple and easy to follow steps using which you can set up and get started with your Amazon Redshift cluster.

Getting started with Amazon Redshift

In this section, we will be looking at a few simple steps which you can take to have a fully functioning Amazon Redshift cluster up and running in a matter of minutes:

1. First up, we have a few prerequisite steps that need to be completed before we begin with the actual set up of the Redshift cluster. From the AWS Management Console, use the **Filter** option to filter out **IAM**. Alternatively, you can also launch the **IAM** dashboard by selecting this URL: <https://console.aws.amazon.com/iam/>.

2. Once logged in, we need to create and assign a role that will grant our Redshift cluster read-only access to Amazon S3 buckets. This role will come in handy later on in this chapter when we load some sample data on an Amazon S3 bucket and use Amazon Redshift's `COPY` command to copy the data locally into the Redshift cluster for processing. To create the custom role, select the **Role** option from the **IAM** dashboards' navigation pane.
3. On the **Roles** page, select the **Create role** option. This will bring up a simple wizard using which we will create and associate the required permissions to our role.
4. Select the **Redshift** option from under the **AWS Service** group section and opt for the **Redshift - Customizable** option provided under the **Select your use case** field. Click **Next** to proceed with the set up.
5. On the **Attach permissions policies** page, filter and select the **AmazonS3ReadOnlyAccess** permission. Once done, select **Next: Review**.
6. In the final **Review** page, type in a suitable name for the role and select the **Create Role** option to complete the process. Make a note of the role's ARN as we will be requiring this in the later steps. Here is snippet of the role policy for your reference:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:Get*",
        "s3:List*"
      ],
      "Resource": "*"
    }
  ]
}
```

With the role created, we can now move on to creating the Redshift cluster.

7. To do so, log in to the AWS Management Console and use the **Filter** option to filter out **Amazon Redshift**. Alternatively, you can also launch the Redshift dashboard by selecting this URL: <https://console.aws.amazon.com/redshift/>.
8. Select **Launch Cluster** to get started with the process.

9. Next, on the **CLUSTER DETAILS** page, fill in the required information pertaining to your cluster as mentioned in the following list:
- **Cluster identifier:** A suitable name for your new Redshift cluster. Note that this name only supports *lowercase* strings.
 - **Database name:** A suitable name for your Redshift database. You can always create more databases within a single Redshift cluster at a later stage. By default, a database named `dev` is created if no value is provided:

Redshift dashboard

Clusters

Snapshots

Security

Parameter groups

Workload management

Reserved nodes

Events

Connect client

CLUSTER DETAILS **NODE CONFIGURATION**

Provide the details of your cluster. Fields marked with * are required.

Cluster identifier* sampledredshiftcluster

Database name loganalysis

Database port* 5439

Master user name* cloudadmin

Master user password*

Confirm password*

- **Database port:** The port number on which the database will accept connections. By default, the value is set to 5439, however you can change this value based on your security requirements.
 - **Master user name:** Provide a suitable username for accessing the database.
 - **Master user password:** Type in a strong password with at least one uppercase character, one lowercase character and one numeric value. Confirm the password by retyping it in the **Confirm password** field.
10. Once completed, hit **Continue** to move on to the next step of the wizard.
11. On the **NODE CONFIGURATION** page, select the appropriate **Node type** for your cluster, as well as the **Cluster type** based on your functional requirements. Since this particular cluster setup is for demonstration purposes, I've opted to select the **dc2.large** as the **Node type** and a **Single Node** deployment with 1 compute node. Click **Continue** to move on the next page once done.



It is important to note here that the cluster that you are about to launch will be live and not running in a sandbox-like environment. As a result, you will incur the standard Amazon Redshift usage fees for the cluster until you delete it. You can read more about Redshift's pricing at: <https://aws.amazon.com/redshift/pricing/>.

12. In the **ADDITIONAL CONFIGURATION** page, you can configure add-on settings, such as encryption enablement, selecting the default VPC for your cluster, whether or not the cluster should have direct internet access, as well as any preferences for a particular Availability Zone out of which the cluster should operate. Most of these settings do not require any changes at the moment and can be left to their default values.
13. The only changes required on this page is associating the previously created IAM role with the cluster. To do so, from the **Available Roles** drop-down list, select the custom Redshift role that we created in our prerequisite section. Once completed, click on **Continue**.
14. Review the settings and changes on the **Review** page and select the **Launch Cluster** option when completed.

The cluster takes a few minutes to spin up depending on whether or not you have opted for a single instance deployment or multiple instances. Once completed, you should see your cluster listed on the **Clusters** page, as shown in the following screenshot. Ensure that the status of your cluster is shown as **healthy** under the **DB Health** column. You can additionally make a note of the cluster's endpoint as well, for accessing it programmatically:

Clusters						
Launch Cluster Manage Tags Manage IAM roles			Refresh Help			
☐	Cluster	Cluster Status	DB Health	In Maintenance	Recent Events	Config timeline
☐	☐ sampleredshiftcluster	available	healthy	no	3	View timeline
Endpoint sampleredshiftcluster.comfgcquubwy.us-east-2.redshift.amazonaws.com:5439 (authorized) ⓘ						

With the cluster all set up, the next thing to do is connect to the same. In the next section, we will be looking at a few simple steps you can take to connect to your newly deployed Redshift cluster.

Connecting to your Redshift cluster

You can use a number of tools to connect to your Redshift cluster once its up and running. Most of these tools are PostgreSQL compliant and easily available off the shelf. In this case, we are going to install and use an open source SQL client tool called **SQL Workbench/J**.

To begin with, you will need to have Java runtime installed on your local workstation. The Java runtime version will have to *match* the requirements of SQL Workbench/J, otherwise it simply won't work. You can check the version of the installed Java runtime on your local desktop by either locating the Java configuration on the **Control Panel** or by typing in the following command in a Terminal if you are working with a Linux distribution:

```
# java --version
```

In this case, we are using a simple Windows desktop for installing SQL Workbench/J. Download the correct version of the software from here: <http://www.sql-workbench.net/downloads.html>.

With the software downloaded, the installation is pretty straightforward. Accept the end user license agreement, select a path for the software's installation and that's it! You should have the SQL Workbench/J up and running now:

1. To connect SQL Workbench/J with your Redshift cluster, you will need your newly created database's JDBC URL. You can copy it by selecting the **Connect client** option from Redshift's navigation pane and selecting your newly deployed cluster from the **Get cluster connection URL** section, as shown in the following screenshot:

2. Download Amazon Redshift drivers:

JDBC Driver: JDBC 4.2 (.jar) v1.2.8.1005
Download

ODBC Driver: Microsoft Windows OS 64-bit (.msi) v1.3.7
Download

3. Get cluster connection URL:

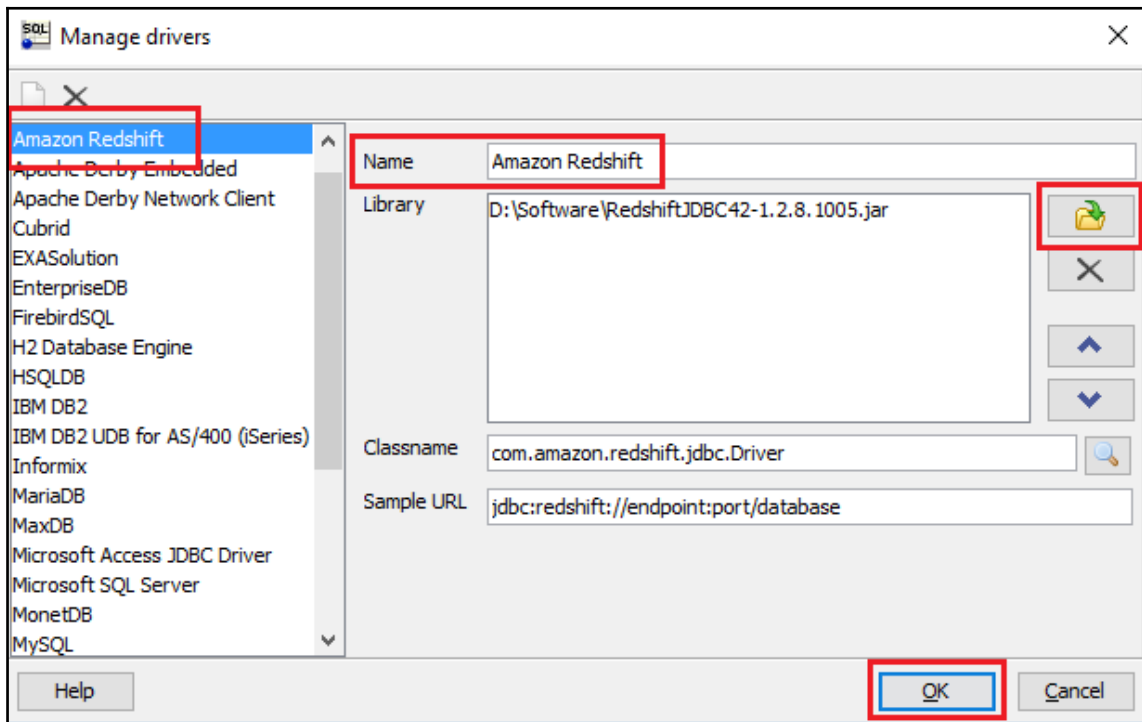
Cluster: sampledredshiftcluster

JDBC URL: jdbc:redshift://sampledredshiftcluster.comfgcquubwy.us-east-2.redshift.amazonaws.com:5439/loganalysis

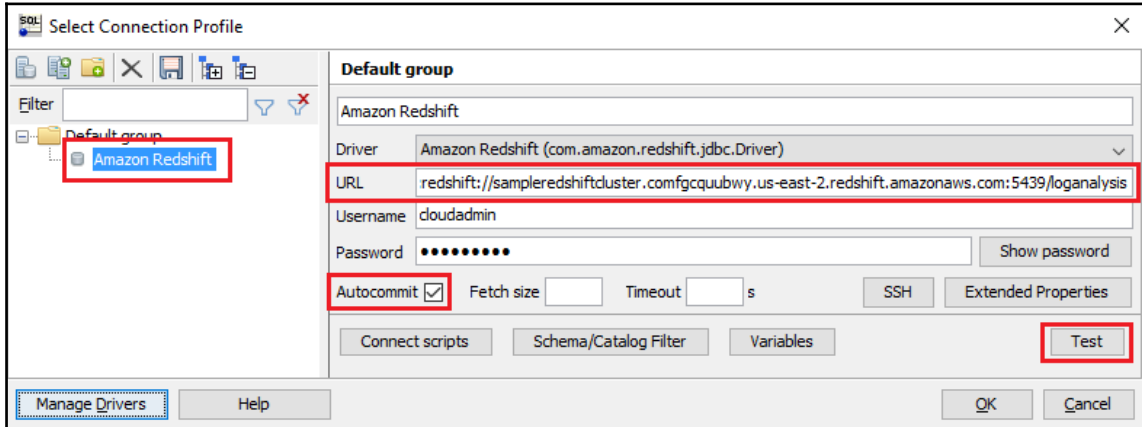
ODBC URL: Driver={Amazon Redshift (x64)}; Server=sampledredshiftcluster.comfgcquubwy.us-east-2.redshift.amazonaws.com; Database=loganalysis; UID=cloudadmin; PWD=insert_your_master_user_password_here; Port=5439

2. You will also need to download the correct version of the associated Amazon Redshift **JDBC Driver JAR** using the same page as well.

3. Once completed, from the SQL Workbench/J client, select **File**, followed by the **Connect window** option.
4. Here, click on **Create a new connection profile** to get started. This will pop up a **New profile** box where you will need to enter a name for this new profile.
5. Once the profile is created; select the **Manage drivers** option. This will display the **Manage drivers** dialog box, as shown in the following screenshot. Select the **Amazon Redshift** option and provide a suitable **Name** for your connection driver, as well. Click on the browse icon and select the downloaded Amazon Redshift driver JAR that we downloaded from Redshift a while back. Click on **OK** to complete the driver settings:



6. With the driver in place, the final thing left to do is connect to the database and test it. For that, select the newly created **Connection profile** from SQL Workbench/J and paste the copied database JDBC URL in the **URL** field as shown. Provide the database's **Username** and **Password** as configured during the cluster's setup. Additionally, ensure that the **Autocommit** option is checked as shown here:



7. You can also test the connection by selecting the **Test** option on the SQL Workbench/J screen. Once completed, click **OK** to establish and open the SQL prompt.

With this step completed, you should have a running Redshift cluster connected to the SQL Workbench/J client as well. The next and final step left for us is to run a few sample queries and test the cluster's functionality, so let's get started with that right away!

Working with Redshift databases and tables

Before we start querying the Redshift database, we will first need to upload some same data to it. For this particular scenario, we are going to use a small subset of HTTP request logs that originated from a web server at the NASA Kennedy Space Center in Florida. This data is available for public use and can be downloaded from here: <http://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html>.

The log file essentially contains the following set of columns:

- **Host:** The host that is making the web request to the web server. This field contains fully qualified hostnames or IP addresses as well.
- **Timestamp:** The timestamp of the particular web request. The format is DAY MON DD HH:MM:SS YYYY. This timestamp uses a 24-hour clock.
- **Request:** The method used to request the server (GET/HEAD/POST).
- **URL:** The URL of the resource that was requested by the client.
- **Response:** This contains the HTTP response code (200, 302, 304, and 404).
- **Bytes:** The size of the reply in bytes.

Here's a snippet of the data for your reference:

```
pppa006.compuserve.com,807256800,GET,/images/launch-logo.gif,200,1713  
vcc7.langara.bc.ca,807256804,GET,/shuttle/missions/missions.html,200,8677  
pppa006.compuserve.com,807256806,GET,/history/apollo/images/apollo-  
logo1.gif,200,1173
```



You can download the sample CSV file (2.14 MB containing 30,970 entries) used for this scenario using the following link:

<https://github.com/yoyoclouds/Administering-AWS-Volume2>.

With the file downloaded, all you need to do is upload it to one of your Amazon S3 buckets. Remember, that this bucket should be accessible by Amazon Redshift otherwise you may get a `S3ServiceException: Access Denied` exception during execution.

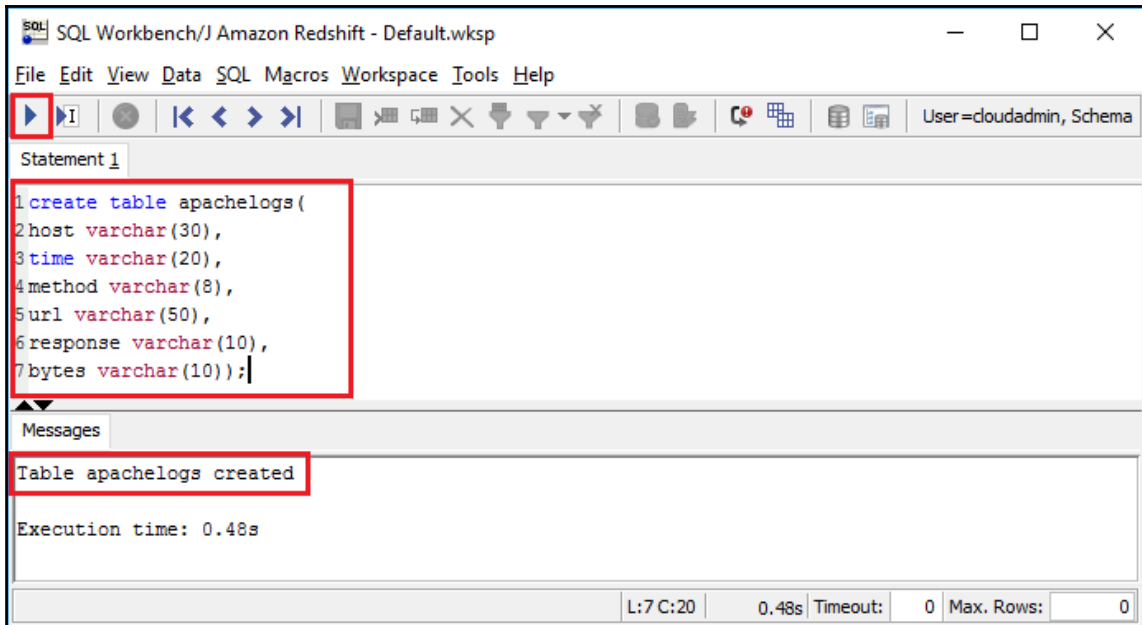
Next, from the SQL Workbench/J client, type in the following code to create a new table within our Redshift database:

```
create table apachelogs(  
  host varchar(100),  
  time varchar(20),  
  method varchar(8),  
  url varchar(200),  
  response varchar(10),  
  bytes varchar(10));
```



You can find the complete copy of the previous code at: <https://github.com/yoyoclouds/Administering-AWS-Volume2>.

Select the **Execute Query** button. You should receive an output stating that the table is created, as shown in the following screenshot:

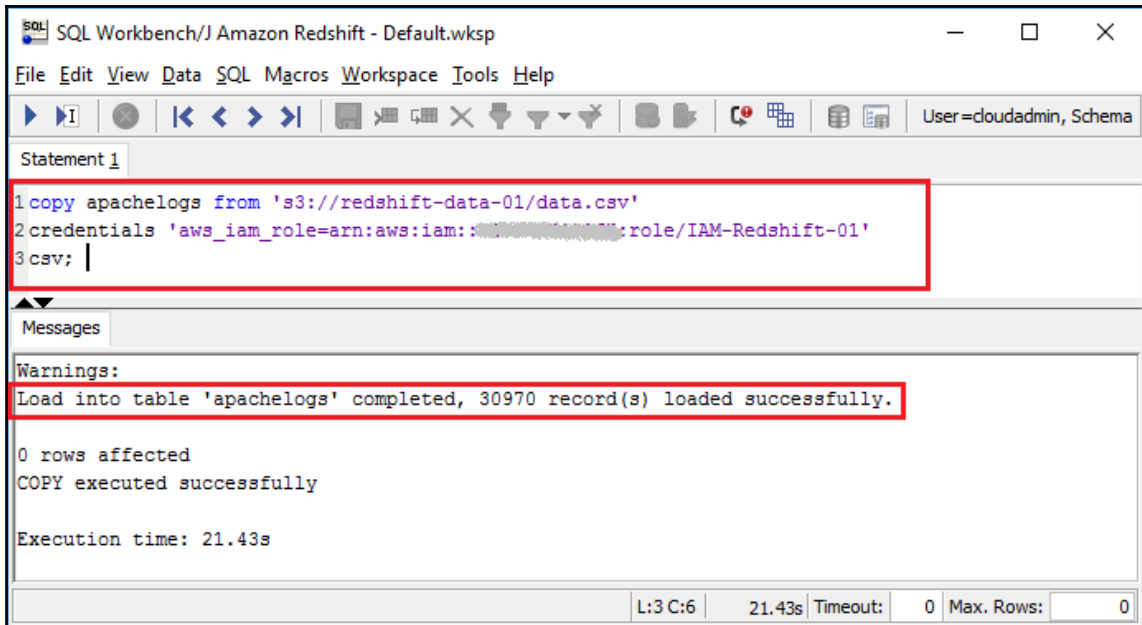


Next, use the `COPY` command to load the contents of the data file stored in Amazon S3 into the newly created Redshift table. The `COPY` command is a very versatile command and can be used to load data residing in Amazon S3, Amazon EMR, or even from an Amazon DynamoDB table into Amazon Redshift. To know more about the `COPY` command, navigate to this URL: https://docs.aws.amazon.com/redshift/latest/dg/r_COPY.html.

Substitute the values of `<REDSHIFT_TABLE_NAME>` with the name of the newly created table, the `<BUCKET_NAME>` with the name of the S3 bucket that contains the data file, and `<REDSHIFT_IAM_ROLE_ARN>` with the ARN of the IAM read-only access role that we created as a part of Amazon Redshift's prerequisite process:

```
copy <REDSHIFT_TABLE_NAME> from 's3://<BUCKET_NAME>/data.csv'  
credentials 'aws_iam_role=<REDSHIFT_IAM_ROLE_ARN>'  
csv;
```

Once the code is pasted into the SQL Workbench/J, click on the **Execute Query** button. Here is a snapshot of the command execution from SQL Workbench/J:



With the data loaded, you can now use simple queries to query the dataset, as described in this section. The following command will list all 30,970 records from the table:

```
select * from apachelogs;
```

The following command will list only those records whose response value was 404:

```
select * from apachelogs where response=404;
```

The following command will list all the hosts that have requested for the particular resource:

```
select host from apachelogs where url='/images/NASA-logosmall.gif';
```

You can also use the Redshift dashboard to view the performance and runtime of each individual query by first selecting your Redshift cluster name from the **Cluster** page. Next, select the **Queries** tab to bring up the list of the most recently executed queries, as shown in the following screenshot:

	Query	Run time	Start time	Status	User	SQL
☒	178	2.2s	January 29, 2018 at 7:34:06 PM UTC+1	COMPLETED	cloudadmin	select host from apachelogs where url='/images/NASA-logosmall
☐	165	10.71s	January 29, 2018 at 7:31:17 PM UTC+1	COMPLETED	cloudadmin	select * from apachelogs where response=200
☐	163	11.92s	January 29, 2018 at 7:30:33 PM UTC+1	COMPLETED	cloudadmin	select * from apachelogs

You can drill down into each query by further selecting the *query identification number* as well.

Planning your next steps

Before we conclude by summarizing the chapter, there are a few things I highly recommend that you try out with Amazon EMR, as well as with Amazon Redshift. First up, EMRFS.

We briefly touched upon the topic of EMRFS while deciding which filesystem to opt for when it comes to deploying the EMR Cluster. **EMR File System (EMRFS)** is an implementation of the traditional HDFS that allows for reading and writing files from Amazon EMR directly to Amazon S3. This essentially allows you to leverage the consistency provided by S3, as well as some of its other feature sets, such as data encryption. To read more about EMRFS and how you can use it for your EMR clusters, visit: <https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-fs.html>.

Secondly, Amazon EMR also provides an enterprise-grade Hadoop distribution in the form of MapR. The MapR distribution of Hadoop provides you with a plethora of features that enhances your overall experience when it comes to building distributed applications, as well as managing the overall Hadoop cluster. For example, selecting MapR as the Hadoop distribution provides support for industry-standard interfaces, such as NFS and ODBC, using which you can connect your EMR cluster with any major BI tool, including Tableau and Toad. MapR internally also provides built-in high availability, data protection, higher performances, and a whole list of additional features. You can read more about the MapR distribution for Hadoop at EMR at: <https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-mapr.html>.

Last but not the least, I would also recommend that you try out some of Amazon Redshift's advanced features in the form of reserved nodes and parameter groups. Parameter groups are essentially a group of parameters that are applied to the database when it is created. You can find the parameter group for your existing database by selecting the **Parameter Group** option from the Redshift's navigation pane. You can use and tweak these parameter groups based on your requirements to fine tune and customize the database. To know how to leverage parameter groups for your database tuning, visit: <https://docs.aws.amazon.com/redshift/latest/mgmt/working-with-parameter-groups.html>.

Summary

Well that brings us to the end of yet another amazing chapter. Let's quickly summarize what we have learnt so far!

To start off, we began by learning a bit about the various services offered by AWS for big data analytics followed by a quick getting started with Amazon EMR guide. We learnt about a few of Amazon EMR's concepts as well as launched our very first EMR cluster, as well. We also ran our first simple job on the EMR cluster and learnt how to monitor its performance using the likes of Amazon CloudWatch.

Towards the end of the chapter, we got to know Amazon Redshift along with its core concepts and workings. We also created our first Redshift cluster, connected to it using an open source client and ran a couple of SQL queries, as well.

In the next chapter, we will be learning and exploring yet another AWS service designed for data orchestration so stick around, we still have much to learn!

10

Orchestrating Data using AWS Data Pipeline

In the previous chapter, we explored the AWS analytics suite of services by deep diving into Amazon EMR and Amazon Redshift services.

In this chapter, we will be continuing the trend and learning about an extremely versatile and powerful data orchestration and transformation service called AWS Data Pipeline.

Let's have a quick look at the various topics that we will be covering in this chapter:

- Introducing AWS Data Pipeline along with a quick look at some of its concepts and terminologies
- Getting started with Data Pipeline using a simple Hello World example
- Working with the Data Pipeline definition file
- Executing scripts and commands on remote EC2 instances using a data pipeline
- Backing up data from one S3 bucket to another using a simple, parameterized data pipeline
- Building pipelines using the AWS CLI

So without any further ado, let's get started right away!

Introducing AWS Data Pipeline

AWS Data Pipeline is an extremely versatile web service that allows you to move data back and forth between various AWS services, as well as on-premise data sources. The service is designed specifically to provide you with an in-built fault tolerance and highly available platform, using which you can define and build your very own custom data migration workflows. AWS Data Pipeline also provides add-on features such as scheduling, dependency tracking, and error handling, so that you do not have to waste extra time and effort in writing them on your own. This easy-to-use and flexible service, accompanied by its low operating costs, make the AWS Data Pipeline service ideal for use cases such as:

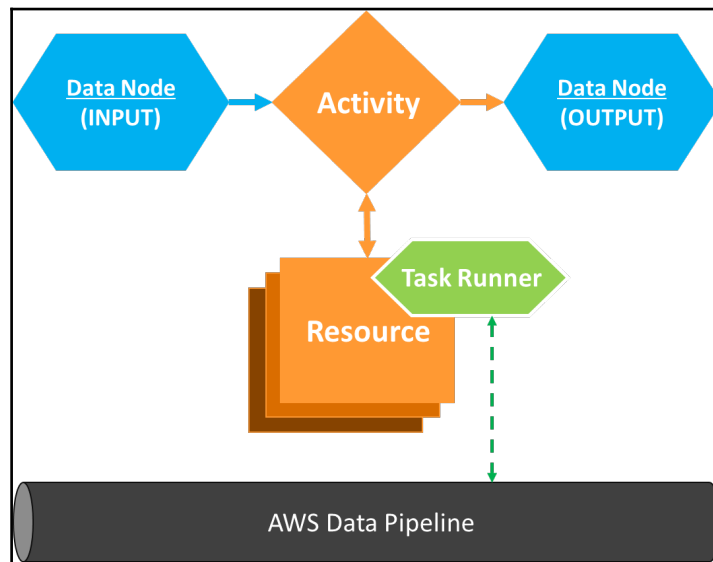
- Migrating data on a periodic basis from an Amazon EMR cluster over to Amazon Redshift for data warehousing
- Incrementally loading data from files stored in Amazon S3 directly into an Amazon RDS database
- Copying data from an Amazon MySQL database into an Amazon Redshift cluster
- Backing up data from an Amazon DynamoDB table to Amazon S3
- Backing up files stored in an Amazon S3 bucket on a periodic basis, and much more

In this section, we will be understanding and learning a bit more about AWS Data Pipeline by first getting to know some of its internal components, concepts and terminologies.

The core foundation of AWS Data Pipeline is, as the name suggests, a pipeline. You can create pipelines to schedule and run your data migration or transformation tasks. Each pipeline relies on a pipeline definition that essentially contains the business logic required to drive the data migration activities. We will be learning more about the data pipeline definition in the upcoming sections. For now, let's dive a bit into a few essential pipeline concepts and components:

- **Pipeline components:** A single pipeline can comprise of multiple sections, each having its own specific place in the overall functioning of the pipeline. For example, a pipeline can contain sections for specifying the input data source from where the data has to be collected, the activity that needs to be performed on this data along with a few necessary conditions, the time at which the activity has to be triggered, and so on. Each of these sections, individually, are called the pipeline's components and are used together to build a pipeline definition.

- **Task runners:** Task runners are special applications or agents that carry out the task assigned in a pipeline. The task runners poll AWS Data Pipeline for any active tasks available. If found, the task is assigned to a task runner and executed. Once the execution completes, the task runner will report the status (either success or failure) back to AWS Data Pipeline. By default, AWS provides a default task runner for resources that are launched and managed by AWS Data Pipeline. You can also install the task runner on instances or on-premise servers that you manage:



- **Data nodes:** Data nodes are used to define the location and the type of input as well as output data for the pipeline. As of now, the following data nodes are provided by AWS Data Pipeline:
 - `S3DataNode`: Used to define an Amazon S3 location as an input or output for storing data
 - `SqlDataNode`: Defines a SQL table or a database query for use in the pipeline
 - `RedshiftDataNode`: Used to define an Amazon Redshift table as input or output for the pipeline
 - `DynamoDBDataNode`: Used to specify a DynamoDB table as input or output for the pipeline

- **Activities:** With the data's location and type selected using the data nodes, the next component left to define is the type of activity to be performed on that data. AWS Data Pipeline provides the following set of pre-packaged activities that you can use and extend as per your requirements:
 - **CopyActivity:** Used to copy data from one data node to another
 - **ShellCommandActivity:** Used to run a shell command as an activity
 - **SqlActivity:** Executes a SQL query on a data node such as `SqlDataNode` or `RedshiftDataNode`
 - **RedshiftCopyActivity:** A specific activity that leverages the `COPY` command to copy data between Redshift tables
 - **EmrActivity:** Used to run an EMR cluster
 - **PigActivity:** Used to run a custom Pig script on an EMR cluster
 - **HiveActivity:** Runs a Hive query on an EMR cluster
 - **HiveCopyActivity:** Used to run a Hive `COPY` query for copying the data from the EMR cluster to an Amazon S3 bucket or an Amazon DynamoDB table
- **Resources:** With the data nodes and activities selected, the next step in configuring a pipeline is selecting the right resource for executing the activity. AWS Data Pipeline supports two types of resources:
 - **Ec2Resource:** An EC2 instance is leveraged to execute the activity selected in the pipeline. This resource type is common for activities, such as `CopyActivity`, `ShellCommandActivity`, and so on.
 - **EmrCluster:** An Amazon EMR cluster is used to execute the activity selected in the pipeline. This resource is best suited for activities such as `EmrActivity`, `PigActivity`, `HiveActivity`, and so on.
- **Actions:** Actions are certain steps that a pipeline takes whenever a *success*, *failure* or *late activity* event occurs. You can use actions as a way to monitor and notify the execution status of your pipeline; for example, send an SNS notification in case a `CopyActivity` fails, and so on.

With these concepts and terms done and dusted, let's move on to some hands-on action where we will be creating our very first simple and minimalistic pipeline.

Getting started with AWS Data Pipeline

Creating your own pipeline is a fairly simple process, once you get to know the intricacies of working with the pipeline dashboard. In this section, we will be exploring the AWS Data Pipeline dashboard, its various functions, and editor to create a simple Hello World example pipeline. To start off, here are a few necessary prerequisite steps that you need to complete first, starting with a simple Amazon S3 bucket for storing all our data pipeline logs.



AWS Data Pipeline is only available in the EU (Ireland), Asia Pacific (Sydney), Asia Pacific (Tokyo), US East (N. Virginia), and the US West (Oregon) regions. For the purpose of the scenarios in this chapter, we will be using the US East (N. Virginia) region only.

From the AWS Management Console, launch the Amazon S3 console by either filtering the service name from the **Filter** option or navigating to this URL: <https://s3.console.aws.amazon.com/s3/home?region=us-east-1>.

Next, select the **Create bucket** option and provide a suitable value in the **Bucket name** field. Leave the rest of the fields to their default values and select **Create** to complete the process.

With the log bucket created, the next prerequisite step involves the creation of a couple of IAM Roles that are required by AWS Data Pipeline for accessing resources, as well as what particular action it can perform over them. Since we are going to use the AWS Data Pipeline console for our first pipeline build, Data Pipeline provides two default IAM Roles that you can leverage out of the box:

- **DataPipelineDefaultRole**: An IAM Role that grants AWS Data Pipeline access to all your AWS resources, including EC2, IAM, Redshift, S3, SNS, SQS and EMR. You can customize it to restrict the AWS services that Data Pipeline can access. Here is a snippet of the policy that is created:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:*",
        "datapipeline:DescribeObjects",
        "datapipeline:EvaluateExpression",
        "dynamodb:BatchGetItem",
        "dynamodb:DescribeTable",
        "dynamodb:GetItem",
```

```

        ...
        "ec2:RunInstances",
        "ec2:StartInstances",
        "ec2:StopInstances",
        ...
        "elasticmapreduce:*",
        "iam:GetInstanceProfile",
        "iam:GetRole",
        "iam:GetRolePolicy",
        ...
        "rds:DescribeDBInstances",
        "rds:DescribeDBSecurityGroups",
        "redshift:DescribeClusters",
        "redshift:DescribeClusterSecurityGroups",
        "s3:CreateBucket",
        "s3:DeleteObject",
        "s3:Get*",
        "s3:List*",
        "s3:Put*",
        ...
        "sns:ListTopics",
        "sns:Publish",
        "sns:Subscribe",
        ...
        "sqs:GetQueue*",
        "sqs:PurgeQueue",
        "sqs:ReceiveMessage"
    ],
    "Resource": [
        "*"
    ]
},
{
    "Effect": "Allow",
    "Action": "iam:CreateServiceLinkedRole",
    "Resource": "*",
    "Condition": {
        "StringLike": {
            "iam:AWSServiceName": [
                "elasticmapreduce.amazonaws.com",
                "spot.amazonaws.com"
            ]
        }
    }
}
]
}

```

- `DataPipelineDefaultResourceRole`: This Role allows applications, scripts, or code executed on the Data Pipeline resources' (EC2/EMR instances) access to your AWS resources:

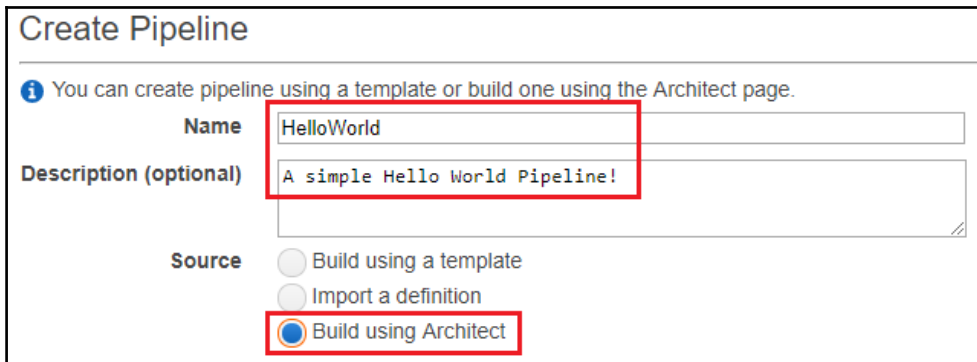
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:*",
        "datapipeline:*",
        "dynamodb:*",
        "ec2:Describe*",
        "elasticmapreduce:AddJobFlowSteps",
        "elasticmapreduce:Describe*",
        "elasticmapreduce:ListInstance*",
        "elasticmapreduce:ModifyInstanceGroups",
        "rds:Describe*",
        "redshift:DescribeClusters",
        "redshift:DescribeClusterSecurityGroups",
        "s3:*",
        "sdb:*",
        "sns:*",
        "sqs:*"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

With the prerequisites out of the way, let's now move on to creating our very first pipeline:

1. From the AWS Management Console, filter out **Data Pipeline** using the **Filter** option or alternatively, selecting this URL provided here <https://console.aws.amazon.com/datapipeline/home?region=us-east-1>. Select the **Get started now** option.
2. This will bring up the **Create Pipeline** wizard as displayed. Start by providing a suitable name for the pipeline using the **Name** field followed by an optional **Description**.

3. Next, select the **Build using Architect** option from the **Source** field.

AWS Data Pipeline provides different ways for creating pipelines. You can leverage either one of the several pre-built templates using the **Build using a template** option, or opt for a more customized approach by selecting the **Import a definition** option, where you can create and upload your own data pipeline definitions. Finally, you can use the data pipeline architect mode to drag-drop and customize your pipeline using a simple intuitive dashboard, which is what we are going to do in this use case:



Create Pipeline

i You can create pipeline using a template or build one using the Architect page.

Name HelloWorld

Description (optional) A simple Hello World Pipeline!

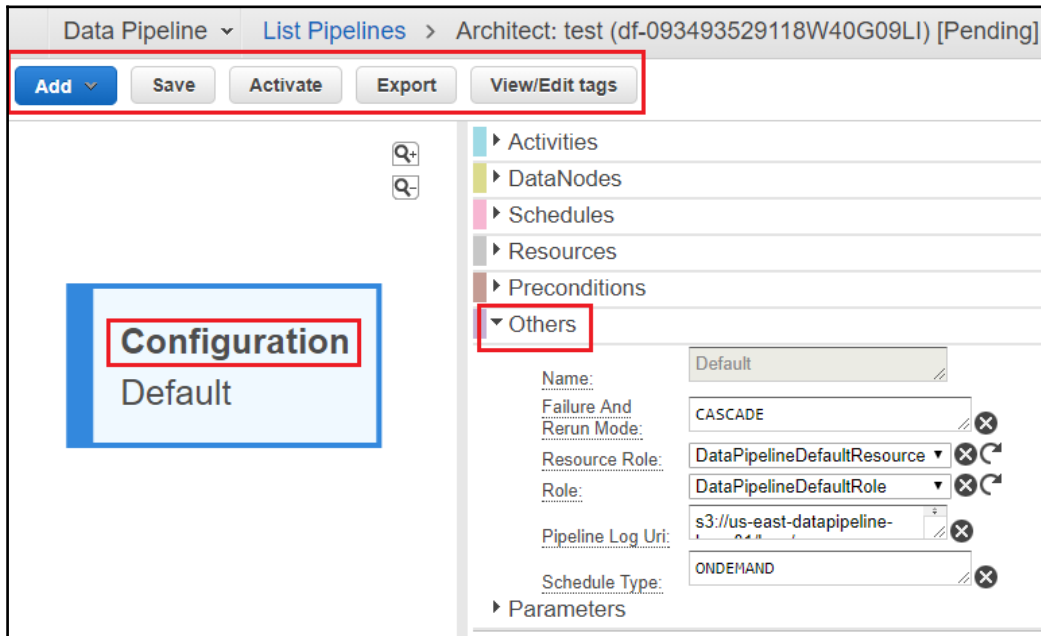
Source

- ☐ Build using a template
- ☐ Import a definition
- ☒ Build using Architect

4. Moving on, you can also schedule the run of your pipeline by selecting the correct option, provided under the **Schedule** section. For now, select the **On pipeline activation** option, as we want our pipeline to start its execution only when it is first activated.
5. Next, browse and select the correct *S3 bucket* for logging the data pipelines' logs using the **S3 location for logs** option. This should be the same bucket that was created during the prerequisite section of this scenario.
6. Optionally, you can also provide your custom IAM Roles for Data Pipeline by selecting the **Custom** option provided under the **Security/Access** section. In this case, we have gone ahead and selected the **Default** IAM Roles themselves.
7. Once all the required fields are populated, select the **Edit in Architect** option to continue.

With this step completed, you should see the *architect* view of your current pipeline as depicted. By default, you will only have a single box called **Configuration** displayed.

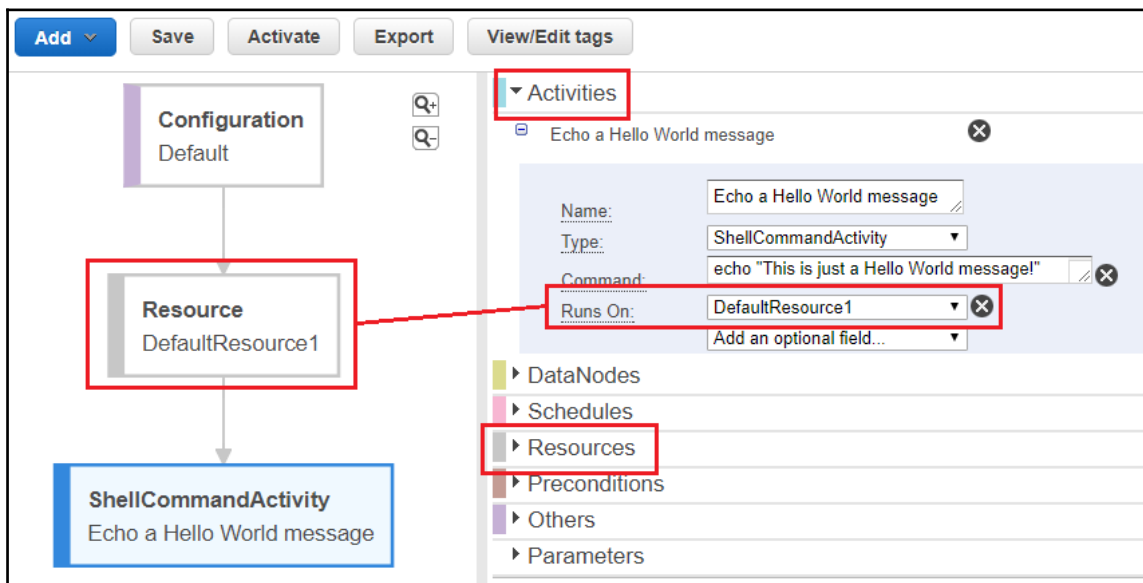
8. Select the **Configuration** box to view the various configuration options required by your pipeline to run. This information should be visible on the right-hand side navigation pane under the **Others** section, as shown in the following screenshot:



You can use this **Configuration** to edit your pipeline's **Resource Role**, **Pipeline Log Uri**, **Schedule Type**, and many other such settings as well.

9. To add **Resources** and **Activities** to your pipeline, select the **Add** drop-down list as shown. Here, select `ShellCommandActivity` to get started. We will use this activity to echo a simple Hello World message for starters.
10. Once the `ShellCommandActivity` option is selected, you should be able to see its corresponding configuration items in the adjoining navigation pane under the **Activities** tab.
11. Type in a suitable **Name** for your activity. Next, from the **Type** section, select the **Add an optional field** drop-down list and select the **Command** option as shown. In the new **Command** field, type `echo "This is just a Hello World message!"`.

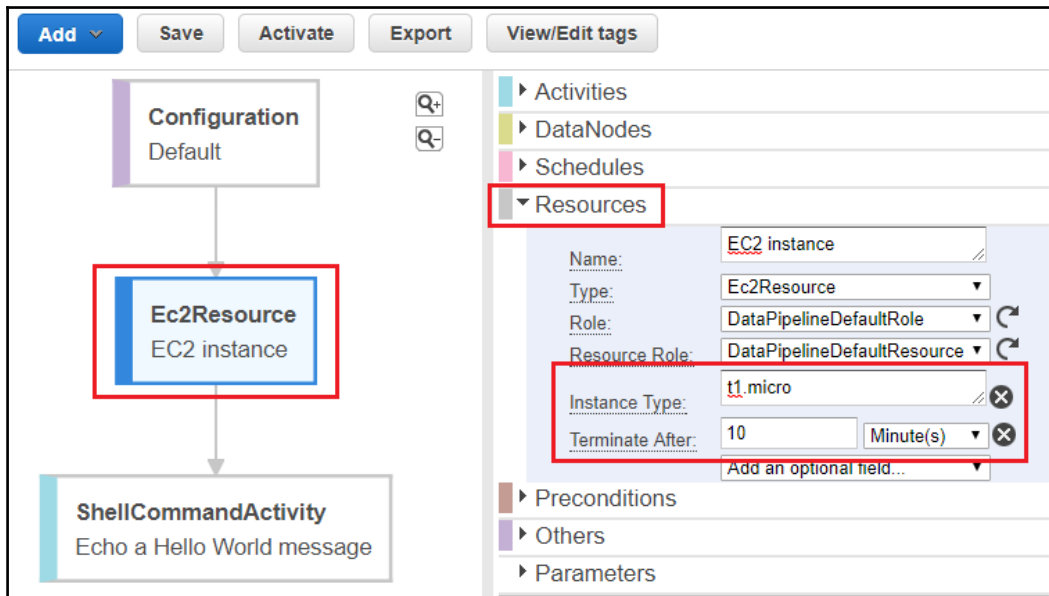
12. With the activity in place, the final step left is to provide and associate a resource to the pipeline. The resource will execute the `ShellCommandActivity` on either an EC2 instance or an EMR instance.
13. To create and associate a resource, from the **Activities** section, select the **Add an optional field** option once again and from the drop-down list, select the **Runs On** option. Using the **Runs On** option, you can create and select **Resources** for executing the task for your pipeline.
14. Select the **Create new: Resource** option to get started. This will create a new resource named `DefaultResource1`, as depicted in the following screenshot:



15. Select the newly created resource or alternatively, select the **Resources** option from the navigation pane to view and add resource specific configurations.
16. Fill in the following information as depicted in the previous screenshot in the **Resources** section of your pipeline:
 - **Name:** Provide a suitable name for your new resource.
 - **Type:** Select the **Ec2Resource** option from the drop-down list.
 - **Role/Resource Role:** You can choose to provide different IAM Roles, however I have opted to go for the default pipeline roles itself.

- **Instance Type:** Type in `t1.micro` in the adjoining field. If you do not provide or select the instance type field, the resource will launch a **m1.medium** instance by default.
- **Terminate After:** Select the appropriate time after which the instance should be terminated. In this case, I have selected to terminate after 10 minutes.

Here's a screenshot of what the final pipeline would look like once the **Resources** section is filled out:

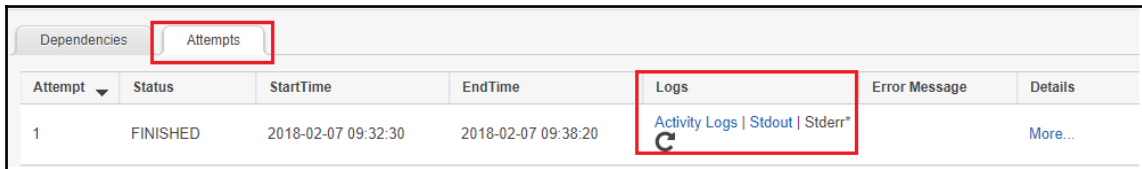


17. Once the pipeline is ready, click on **Save** to save the changes made. Selecting the **Save** option automatically compiles your pipeline and checks for any errors as well. If any errors are found, they will be displayed in the **Errors/Warnings** section. If no errors are reported, click on **Activate** to finally activate your pipeline.

The pipeline takes a few minutes to transition from **WAITING_FOR_RUNNER** state to a **FINISHED** state. This process involves first spinning up the EC2 instance or resource, which we defined in the pipeline. Once the resource is up and running, Data Pipeline will automatically install the *task runner* on this particular resource, as Data Pipeline itself manages it. With the task runner installed, it starts polling the data pipeline for pending activities and executes them.

Once the pipeline's status turns to **FINISHED**, expand the pipeline's component name and select the **Attempts** tab, as shown. If not specified, Data Pipeline will try and execute your pipeline for a default three attempts before it finally stops the execution.

For each attempt, you can view the corresponding **Activity Logs**, **Stdout** as well as the **Stderr** messages:



Dependencies		Attempts				
Attempt	Status	StartTime	EndTime	Logs	Error Message	Details
1	FINISHED	2018-02-07 09:32:30	2018-02-07 09:38:20	Activity Logs Stdout Stderr*		More...

Select the **Stdout** option to view your Hello World message! Et voila! Your first pipeline is up and running!

Feel free to try out a few other options for your pipeline by simply selecting the pipeline name and click on the **Edit Pipeline** option. You can also export your pipeline's definition by selecting the pipeline name and from the **Actions** tab, opting for the **Export** option.

Pipeline definitions are a far better and easier way of creating pipelines if you are a fan of working with JSON and CLI interfaces. They offer better flexibility and usability as compared to the standard pipeline dashboard which can take time to get used to for beginners. With this in mind, in the next section we will be exploring a few basics on how you can get started by creating your very own pipeline definition file.

Working with data pipeline definition Files

The AWS Data Pipeline console provides us with three different options to get started with creating a new pipeline. You could use the architect mode, which is exactly what we ended up working with in the earlier section, or alternatively, use any one of the pre-defined templates as a boilerplate and build your pipeline from them. Last but not the least, the console also provides you with an ability to upload your very own pipeline definition file, which is basically a collection of various pipeline objects and conditions written in a JSON format. In this section, we will be learning how to write our very own pipeline definitions and later, use the same for building a custom pipeline as well.

To start, you will need two components to build up a pipeline definition file: objects and fields:

- **Objects:** An object is an individual component required to build a pipeline. These can be data nodes, conditions, activities, resources, schedules, and so on.
- **Fields:** Each object is described by one or more fields. The fields are made up of key-value pairs that are enclosed in double quotes and separated by a colon.

Here is a skeleton structure of a pipeline definition file:

```
{
  "objects" : [
    {
      "key1" : "value1",
      "key2" : "value2"
    },
    {
      "key3" : "value3"
    }
  ]
}
```

Here is a look at the pipeline definition file obtained by exporting the **Hello World** pipeline example that we performed a while back:

```
{
  "objects": [
    {
      "failureAndRerunMode": "CASCADE",
      "resourceRole": "DataPipelineDefaultResourceRole",
      "role": "DataPipelineDefaultRole",
      "pipelineLogUri": "s3://us-east-datapipeline-logs-01/logs/",
      "scheduleType": "ONDEMAND",
      "name": "Default",
      "id": "Default"
    },
    {
      "name": "myActivity",
      "id": "ShellCommandActivityId_2viZe",
      "runsOn": {
        "ref": "ResourceId_EhxAF"
      },
      "type": "ShellCommandActivity",
      "command": "echo \"This is just a Hello World message!\""
    },
    {

```

```

    "resourceRole": "DataPipelineDefaultResourceRole",
    "role": "DataPipelineDefaultRole",
    "name": "myEC2Resource",
    "id": "ResourceId_EhxAF",
    "type": "Ec2Resource",
    "terminateAfter": "10 Minutes"
  }
],
"parameters": []
}

```



You can find the complete copy of code at <https://github.com/yoyoclouds/Administering-AWS-Volume2>.

Each object generally contains an `id`, `name`, and `type` fields that are used to describe it and its functionality. For example, the `Resource` object in the Hello World scenario contains the following values:

```

{
  "name": "myEC2Resource",
  "id": "ResourceId_EhxAF",
  "type": "Ec2Resource",
  ...
}

```

You can also find the same fields in both the `ShellCommandActivity`, as well as the default configurations objects.

A pipeline object can refer to other objects within the same pipeline using the `"ref" : "ID_of_referred_resource"` field. Here is an example of the `ShellCommandActivity` referencing to the EC2 resource, using the resource ID:

```

{
  "name": "myActivity",
  "id": "ShellCommandActivityId_2viZe",
  "runsOn": {
    "ref": "ResourceId_EhxAF"
  },
  "type": "ShellCommandActivity",
  "command": "echo \"This is just a Hello World message!\"",
},
{
  "resourceRole": "DataPipelineDefaultResourceRole",
  "role": "DataPipelineDefaultRole",
}

```

```
"name": "myEC2Resource",
"id": "ResourceId_EhxAF",
"type": "Ec2Resource",
"terminateAfter": "10 Minutes"
}
```

You can additionally create custom or user-defined fields and refer them to other pipeline components, using the same syntax as described in the previous code:

```
{
  "id": "ResourceId_EhxAF",
  "type": "Ec2Resource",
  "myCustomField": "This is a custom field.",
  "myCustomReference": {"ref": "ShellCommandActivityId_2vi"}
},
```



You can find the detailed references for data nodes, resources, activities, and other objects at <https://docs.aws.amazon.com/datapipeline/latest/DeveloperGuide/dp-pipeline-objects.html>.

Last but not the least; you can also leverage a parameterized template to customize the pipeline definition. Using this method, you can basically have one common pipeline definition and pass different values to it at the time of pipeline creation.

To parametrize a pipeline definition you need to specify a variable using the following syntax:

```
"#{VARIABLE_NAME}"
```

With the variable created, you can define its value in a separate `parameters` object which can be stored in the same pipeline definition file, or in a separate JSON file altogether as well. Consider the following example where we pass the same Hello World message in the `ShellCommandActivity` however, this time using a variable definition:

```
{
  "name": "myActivity",
  "id": "ShellCommandActivityId_2viZe",
  "runsOn": {
    "ref": "ResourceId_EhxAF"
  },
  "type": "ShellCommandActivity",
  "command": "#{myVariable}"
}
```

Once the variable is defined, we pass its corresponding values and expression in a separate `parameters` object, as shown in the following code:

```
{
  "parameters": [
    {
      "id": "myVariable",
      "description": "Shell command to run",
      "type": "String",
      "default": "echo \"Default message!\""
    }
  ]
}
```

In this case, the variable `myVariable` is a simple string type and we have also provided it with a default value, in case a value is not provided to this variable at the time of the pipeline's creation.



To know more about how to leverage and use variable and parameters in your pipeline definitions, visit <https://docs.aws.amazon.com/datapipeline/latest/DeveloperGuide/dp-custom-templates.html>.

With this, we come towards the end of this section. In the next section, we will look at how you can leverage the AWS Data Pipeline to execute scripts and commands on remote EC2 instances using a parameterized pipeline definition.

Executing remote commands using AWS Data Pipeline

One of the best parts of working with Data Pipeline is that versatility of tasks that you can achieve by just using this one tool. In this section, we will be looking at a relatively simple pipeline definition using which you can execute remote scripts and commands on EC2 instances.

How does this setup work? Well, to start with, we will be requiring one S3 bucket (can be present in any AWS region) to be created that will store and act as a repository for all our shell scripts. Once the bucket is created, simply create and upload the following shell script to the bucket. Note however that in this case, the shell script is named `simplescript.sh` and the same name is used in the following pipeline definition, as well:

```
#!/bin/bash
echo "-----"
echo "Your username is: $(echo $USER)"
echo "-----"
echo "The current date and time : $(date)"
echo "-----"
echo "Users currently logged on this system: "
echo "$(who)"
echo "-----"
echo "AWS CLI installed at: "
echo "$(aws --version)"
echo "-----"
```

The script is pretty self-explanatory. It will print out a series of messages based on the EC2 instance it is launched from. You can substitute this script with any other shell script that can either be used to take backups of particular files, or archive existing files into a `tar.gz` and push it over to an awaiting S3 bucket for archiving, and so on.

With the script file uploaded to the correct S3 bucket, the final step is to copy and paste the following pipeline definition in a file and upload it to Data Pipeline for execution:

```
{
  "objects": [
    {
      "failureAndRerunMode": "CASCADE",
      "resourceRole": "DataPipelineDefaultResourceRole",
      "role": "DataPipelineDefaultRole",
      "pipelineLogUri": "s3://<DATAPIPELINE_LOG_BUCKET>",
      "scheduleType": "ONDEMAND",
      "name": "Default",
      "id": "Default"
    },
    {
      "name": "CliActivity",
      "id": "CliActivity",
      "runsOn": {
        "ref": "Ec2Instance"
      },
      "type": "ShellCommandActivity",
      "command": "(sudo yum -y update aws-cli) && ({myCustomScriptCmd})"
```

```

    },
    {
      "instanceType": "t1.micro",
      "name": "Ec2Instance",
      "id": "Ec2Instance",
      "type": "Ec2Resource",
      "terminateAfter": "15 Minutes"
    }
  ],
  "parameters": [
    {
      "watermark": "aws [options] <command> <subcommand> [parameters]",
      "description": "AWS CLI command",
      "id": "myCustomScriptCmd",
      "type": "String"
    }
  ],
  "values": {
    "myCustomScriptCmd": "aws s3 cp
s3://<S3_BUCKET_SCRIPT_LOCATION>/simplescript.sh . && sh simplescript.sh"
  }
}

```

Remember to swap out the values for `<DATAPIPELINE_LOG_BUCKET>` and `<S3_BUCKET_SCRIPT_LOCATION>` with their corresponding actual values, and to save the file with a JSON extension.

This particular pipeline definition relies on the `ShellCommandActivity` to first install the AWS CLI on the remote EC2 instance and then execute the shell script by copying it locally from the S3 bucket.

To upload the pipeline definition, use the AWS Data Pipeline console to create a new pipeline. In the **Create Pipeline** wizard, provide a suitable **Name** and **Description** for the new pipeline. Once done, select the **Import a definition** option from the **Source** field, as shown in the following screenshot:

Create Pipeline

Name

Description (optional)

Source

☐ Build using a template

☒ Import a definition

☐ Build using Architect

Once the script loads, you should see the custom AWS CLI command in the **Parameters** section. With the pipeline definition successfully loaded, you can now choose to run the pipeline, either on a schedule or on activation. In my case, I have select to run the pipeline on activation itself, as this is for demo purposes.

Ensure that the *logging* is enabled for the new pipeline and the correct S3 bucket for storing the pipeline's logs is mentioned. With all necessary fields filled, click on **Activate** to start up the pipeline.

Once again, the pipeline will transition from **WAITING_FOR_RUNNER** state to the **FINISHED** state. This usually takes a good minute or two to complete.

From the Data Pipeline console, expand on the existing pipeline and select the **Attempts** tab as shown in the following screenshot. Here, click on **Stdout** to view the output of the script's execution:

Edit Pipeline

Show components in state with between

Filter: 1 instances (all loaded)

Component Name	Schedule Interval (UTC)	Type	Status
☒ CliActivity	2018-02-07 10:22:33 - 2018-02-07 10:22:33	ShellCommandActivity	RUNNING

Attempt	Status	StartTime	EndTime	Logs	Error Message
1	RUNNING	2018-02-07 10:22:36		Activity Logs <input checked="" type="button" value="Stdout"/> Stderr	

Once the output is viewed, you can optionally select the pipeline and click on **Mark Finished** option, as well. This will stop the pipeline from undertaking any further attempts on executions.

Simple, isn't it! You can use a similar method and approach to back up your files and execute some commands over managed instances. In the next section, we will be looking at one last pipeline definition example as well, that essentially helps us take periodic backups of content stored in one Amazon S3 bucket to another using both the Data Pipeline console, as well as the AWS CLI!

Backing up data using AWS Data Pipeline

One of the most widely used use cases for AWS Data Pipeline is its ability to synchronize and schedule backup jobs. You can use Data Pipeline to take backups of data stored within EC2 instances, EBS volumes, databases and even S3 buckets. In this section, we will walk through a simple, parameterized pipeline definition using which you can effectively schedule and perform backups of files stored within an Amazon S3 bucket.

First up, let's have a look at the pipeline definition file itself:



You can find the complete copy of code at <https://github.com/yoyoclouds/Administering-AWS-Volume2>.

To start with, we once again provide a list of *objects* that describe the pipeline components starting with a pipeline configuration object, as highlighted in the following code:

```
"objects": [
  {
    "failureAndRerunMode": "CASCADE",
    "resourceRole": "DataPipelineDefaultResourceRole",
    "role": "DataPipelineDefaultRole",
    "pipelineLogUri": "#{myDataPipelineLogs}",
    "scheduleType": "ONDEMAND",
    "name": "Default",
    "id": "Default"
  },
]
```

Next, we provide the definition for other pipeline objects, including the data nodes:

```
{
  "filePath": "#{myInputS3FilePath}",
  "name": "inputS3Bucket",
  "id": "InputS3FilePath",
  "type": "S3DataNode"
},
{
  "filePath": "#{myOutputS3FilePath}/{format(@scheduledStartTime,
'YYYY-MM-dd-HH-mm-ss')}.bak",
  "name": "outputS3Bucket",
  "id": "OutputS3FilePath",
  "type": "S3DataNode"
},
```

In this case, we are using the `#{VARIABLE_NAMES}` to declare a set of variables to make the pipeline definition more reusable. Once the data nodes are configured, we also have to define a set of actions that will trigger SNS alerts based on the pipeline's success or failure. Here is a snippet of the same:

```
{
  "role": "DataPipelineDefaultRole",
  "subject": "Failure",
  "name": "SNSAlertonFailure",
  "id": "OnFailSNSAlert",
  "message": "File was not copied over successfully. Pls check with Data
Pipeline Logs",
  "type": "SnsAlarm",
  "topicArn": "#{mySNSTopicARN}"
},
```

With the objects defined, the second section requires the `parameters` to be set up, where each of the variables declared in the objects section are detailed and defined:

```
"parameters": [
  {
    "watermark": "s3://mysourcebucket/filename",
    "description": "Source File Path:",
    "id": "myInputS3FilePath",
    "type": "AWS::S3::ObjectKey",
    "myComment": "The File path from the Input S3 Bucket"
  },
  {
    "watermark": "s3://mydestinationbucket/filename",
    "description": "Destination (Backup) File Path:",
    "id": "myOutputS3FilePath",
```

```

    "myComment": "The File path for the Output S3 Bucket",
    "type": "AWS::S3::ObjectKey"
  },
  {
    "watermark": "arn:aws:sns:us-east-1:28619EXAMPLE:ExampleTopic",
    "description": "SNS Topic ARN:",
    "id": "mySNSTopicARN",
    "type": "string",
    "myComment": "The SNS Topic's ARN for notifications"
  },
  . . .
]
}

```

With this in mind, let us first look at uploading this definition to AWS Data Pipeline using the web console:

1. Log in to the AWS Data Pipeline console by navigating to this URL: <https://console.aws.amazon.com/datapipeline/home?region=us-east-1>.



We have deployed all of our pipelines so far in the US East (N. Virginia) region itself. You can opt to change the region, as per your requirements.

2. Once done, select the **Create Pipeline** option to get started. In the **Create Pipeline** page, fill in a suitable **Name** and **Description** for the new pipeline:

Create Pipeline

You can create pipeline using a template or build one using the Architect page.

Name FileBackup

Description (optional) Simple JSON to copy file from one S3 Bucket to another

Source

☐ Build using a template

☒ Import a definition

☐ Build using Architect

Load from S3 Load local file

File uploaded: csvcopy.json

3. Next, select the **Import a definition** option and click on the **Load local file** as shown. Copy and upload the JSON file definition here.
4. With the file uploaded, fill out the **Parameters** section as explained here:
 - **S3 bucket path to data pipeline logs:** Browse and provide the bucket path for storing the pipeline's logs.
 - **Source file path:** Browse and select a file that you wish to backup from an Amazon S3 bucket.
 - **Destination (backup) file path:** Browse and select an Amazon S3 bucket path where you store the backed up file. You can optionally provide a backup folder name as well. Each file backed up to this location will follow a standard naming convention: `YYYY-MM-dd-HH-mm-ss.bak`.
 - **SNS Topic ARN:** Provide a valid SNS Topic ARN here. This ARN will be used to notify the user whether the pipeline's execution was a success or a failure.
 - **EC2 instance type:** You can optionally provide a different EC2 instance type as a resource here. By default, it will take the **t1.micro** instance type.
 - **EC2 instance termination:** Once again, you can provide a different instance termination value here. By default, it is set to 20 minutes. The termination time should be changed based on the approximate time taken to back up a file. The larger the file, the more time required to copy it and vice versa.

5. Once the parameter fields are populated, select the Edit in Architect option to view the overall components of the pipeline definition. You should see the following depiction:



6. Click on **Save** to validate the pipeline for any errors. Once done, select **Activate** to start the pipeline's execution process.
7. The pipeline takes a few minutes to transition from the **WAITING_FOR_RUNNER** state to the **FINISHED** state. Once done, check for the backed up file in your destination S3 folder.

You can further tweak this particular pipeline definition to include entire S3 folder paths rather than just an individual file as performed now. Additionally, you can also change the start of the pipeline's execution by changing the `scheduleType` from `ONDEMAND` to `Schedule`, as depicted in the following code snippet:

```
{
  "id" : "Default",
  "type" : "Schedule",
  "period" : "1 hours",
  "startDateTime" : "2018-03-01T00:00:00",
  "endDateTime" : "2018-04-01T00:00:00"
}
```

The following snippet will execute the pipeline every hour starting from March 1, 2018 at 00:00:00 until April 1, 2018 00:00:00.



To know more on how you can use the `Schedule` object, visit <https://docs.aws.amazon.com/datapipeline/latest/DeveloperGuide/dp-object-schedule.html>.

Now that the pipeline is up and running using the console, let us also have a look at a few simple AWS CLI commands using which you can achieve the same results:

1. To start with, create a blank pipeline using the following command:

```
# aws datapipeline create-pipeline
--name <NAME_OF_PIPELINE>
--unique-id <UNIQUE_TOKEN>
```

The `<UNIQUE_TOKEN>` can be any string of characters and is used to ensure idempotency during repeated calls to the `create-pipeline` command.

2. Once the pipeline is created, you will be presented with the pipeline's ID, as depicted in the following screenshot. Make a note of this ID as it will be required in the next steps:

```
yoyo@YoYoNux:~$ aws datapipeline create-pipeline --name CopyActivityPipeline \
> --unique-id S0m#Uniqu#T0kEn
{
  "pipelineId": "df-07900892NQ4TP65JLDRW"
}
yoyo@YoYoNux:~$
```

3. Next, we need to create three separate JSON files with the following content in them:

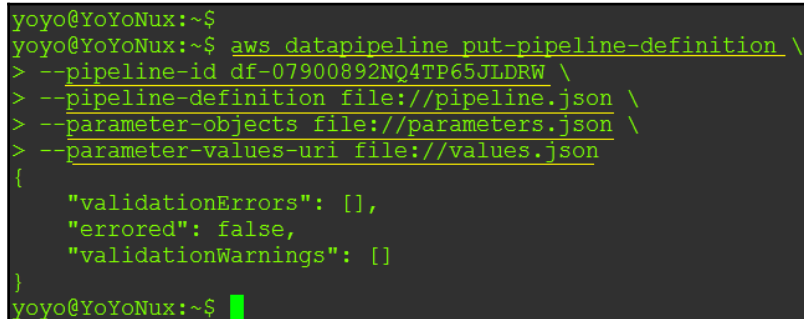
- `pipeline.json`: Copy and paste only the object definitions in this file.
- `parameters.json`: Copy and paste the parameter definitions here.
- `values.json`: Create a new file that contains the values for the parameters, as shown in the following code snippet. Remember to substitute the values in `<>` with those of your own:

```
{
  "values":
  {
    "myDataPipelineLogs": "s3://<BUCKET_NAME>",
    "myOutputS3FilePath": "s3://<BUCKET_NAME>/<FOLDER>",
    "myInputS3FilePath":
"s3://<BUCKET_NAME>/<FILE_NAME>",
    "mySNSTopicARN": "<SNS_ARN_FOR_NOTIFICATIONS>",
    "myEC2InstanceType": "t1.micro",
    "myEC2InstanceTermination": "20"
  }
}
```

4. Once done, save all three files and type in the following command to attach the pipeline definition to the newly created pipeline:

```
# aws datapipeline put-pipeline-definition
--pipeline-id <PIPELINE_ID>
--pipeline-definition file://pipeline.json
--parameter-objects file://parameters.json
--parameter-values-uri file://values.json
```

Here is a screenshot of the command's output for your reference:



```
yoyo@YoYoNux:~$
yoyo@YoYoNux:~$ aws datapipeline put-pipeline-definition \
> --pipeline-id df-07900892NQ4TP65JLDRW \
> --pipeline-definition file://pipeline.json \
> --parameter-objects file://parameters.json \
> --parameter-values-uri file://values.json
{
  "validationErrors": [],
  "errored": false,
  "validationWarnings": []
}
yoyo@YoYoNux:~$
```

5. With the pipeline definition uploaded, the final step left is to activate the pipeline using the following command:

```
# aws datapipeline activate-pipeline
--pipeline-id <PIPELINE_ID>
```

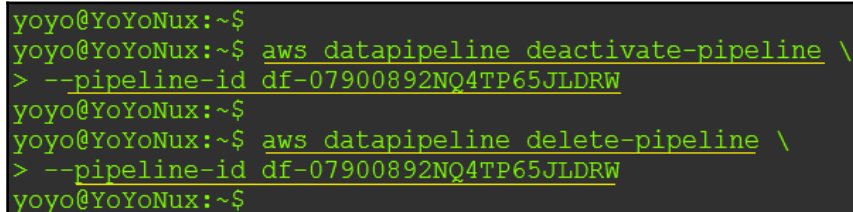
6. Once the pipeline is activated, you can view its status and last runtimes, using the following command:

```
# aws datapipeline list-runs
--pipeline-id <PIPELINE_ID>
```

7. Once the pipeline's execution completes, you can deactivate and delete the pipeline using the following set of commands:

```
# aws datapipeline deactivate-pipeline
--pipeline-id <PIPELINE_ID>
# aws datapipeline delete-pipeline
--pipeline-id <PIPELINE_ID>
```

Here is a screenshot of the command's output for your reference:



```
yoyo@YoYoNux:~$
yoyo@YoYoNux:~$ aws datapipeline deactivate-pipeline \
> --pipeline-id df-07900892NQ4TP65JLDRW
yoyo@YoYoNux:~$
yoyo@YoYoNux:~$ aws datapipeline delete-pipeline \
> --pipeline-id df-07900892NQ4TP65JLDRW
yoyo@YoYoNux:~$
```

With this, we come towards the end of yet another interesting chapter, but before we wind things up, here is a quick look at some important next steps that you should try out on your own.

Planning your next steps

Although we have covered quite a lot in this chapter, there is still a lot to be covered with Data Pipeline. One of the fastest and easiest ways to get started with Data Pipeline is by using one of the ready made pipeline definition templates.

As of date, Data Pipeline provides the following list of ready-to-use templates, using which you can get started with your own pipeline in a matter of minutes:

Getting Started
Getting Started using ShellCommandActivity
AWS Command Line Interface (CLI) Templates
Run AWS CLI command
DynamoDB Templates
Export DynamoDB table to S3
Import DynamoDB backup data from S3
Elastic MapReduce (EMR) Templates
Run job on an Elastic MapReduce cluster
RDS Templates
Full copy of RDS MySQL table to S3
Incremental copy of RDS MySQL table to S3
Load S3 data into RDS MySQL table
Redshift Templates
Full copy of RDS MySQL table to Redshift
Incremental copy of RDS MySQL table to Redshift
Load data from S3 into Redshift

You can additionally use these definitions as templates for further customizing and enhancing your own as well. Simply create a pipeline using one of the previously depicted templates, however do not activate them. Edit the pipeline in the architect mode and simple export the pipeline definition locally. Once the template's pipeline definition is saved locally, you can make further changes and enhancements to it or simply reuse components within it to make your own pipeline, as well. The possibilities are endless!

Another cool feature provided by pipelines is the use of *spot instances* as task nodes. By default, pipelines provide only on-demand instances as resources for your task nodes. You can optionally switch to spot instances by simply selecting the **Task instance Bid Price** option from the **Resources** pane of your pipeline. Provide a suitable amount in the adjoining field (between 0 and 20.00) and there you have it! The next time the pipeline activates and a task is run, it will be performed based on the availability of spot instances.

Summary

Well, that brings us to the end of yet another amazing chapter. Let's quickly summarize the things we have learnt so far!

First up, we started with a brief understanding of AWS Data Pipeline along with its concepts and terminologies. We later also learnt a bit about pipeline definitions and how easy it is to compose and use them. We even built our very first simple **Hello World** pipeline using a pipeline definition, followed by a series of examples that you can tweak and use, according to your own use cases. Towards the end, we also explored a few simple AWS CLI commands required to work with pipelines and topped it all off with a handy guide to some next steps as well.

In the next and final chapter, we will be learning and exploring AWS's versatile and powerful IoT services, so stick around!

11

Connecting the World with AWS IoT and AWS Greengrass

It's been quite a long journey so far and yet here we are; at the last chapter of this book! If you made it to here, then you definitely need to take a moment and give yourself a well-deserved pat on the back!

So far in this book, we have covered a plethora of services, such as Amazon EFS, AWS Beanstalk, AWS Code Suite, AWS Shield, and AWS Data Pipeline, just to name a few. In this final chapter, we will be exploring the IoT suite of services provided by AWS, with more emphasis on two core products, namely, AWS IoT and AWS Greengrass.

Let's have a quick look at the various topics that we will be covering in this chapter:

- A brief look at the building blocks required for IoT
- An introduction to the AWS IoT suite of services followed by a deep dive into AWS IoT, its concepts and terminologies
- Connecting to AWS IoT using a Raspberry Pi Zero device
- Exploring the AWS IoT Device SDK, using a few simple code examples
- Integrating AWS IoT with other AWS services, using IoT rules
- An introduction to AWS Greengrass, along with a simple getting started example
- Effectively monitoring IoT devices, as well as the IoT services

So without any further ado, let's get started!

IoT – what is it?

Well, to the uninitiated, **IoT** or **Internet of Things** is all about connecting everyday *objects* or *things* together, using a common medium of communication (in this case, the internet) for exchange of data. I know it doesn't sound much, but today, IoT is practically being implemented everywhere around us; from wearable devices, smartphones, home appliances, such as refrigerators, air conditioners, to vehicles and heavy machinery, and much more! Gartner predicts that by the year 2020, there will be an estimated 26 billion devices connected using IoT, and this number is set to grow even further, as IoT adoption becomes mainstream. But what exactly is IoT and how do you build it? Here's a quick look at some of the basic building blocks required in order to get started with IoT:

THING	SENSOR	PROCESSOR	GATEWAY	APPLICATION
				

- **Things:** To begin with, any form of IoT comprises end user devices that we use or leverage to perform some of our day-to-day tasks. These devices, or things, can be anything and everything, including simple electronic devices, such as smartphones, wearables, alarm clocks, light bulbs, to washing machines, garage doors, vehicles, ships, and the list just goes on!
- **Sensors:** Sensors are devices that can be incorporated within things that are used to capture or supply our data. Some of the most commonly used sensors are IR sensors, moisture sensors, gas and pressure sensors, and so on. Sensors are not designed to process data on their own. They simply collect and push the data out to one or more processors. For example, a light sensor monitoring whether a light bulb is switched on or off, and so on.

- **Processors:** Processors are the brains of the IoT system. Their main function is to process the data that is captured by the sensors. This processing can be based on certain triggers or can be performed close to real time, as well. A single processor can be used to connect and process data from multiple sensors, as well. The most commonly used type of processors include microcontrollers, embedded controllers, and so on.
- **Gateways:** Gateways are special devices that are responsible for collecting and routing data, processed by one or more processors, to IoT applications for further analysis. A gateway can collect, aggregate, and send data over the internet, either as streams or in batches, depending on its configuration and connectivity options.
- **Application:** Once data from various gateways is collected, it needs to be further analyzed to form meaningful insights, so that appropriate actions on the respective operation can be performed. This can be achieved by leveraging one or more applications, such as an industrial control hub, or even a home automation system. For example, an application can be used to remotely trigger a light bulb to switch on, once the ambient light in the room starts to fade, and so on.

With this essential information in mind, let's look at a few key AWS services you can use to get started with your very own IoT on the cloud.

Introducing the AWS IoT suite of services

AWS has vastly enhanced its IoT suite of services, ever since its first inception towards the end of 2015. Here is a brief explanation of what it offers:

- **AWS IoT Core:** The AWS IoT Core is a managed service that allows you to securely connect and interact with billions of IoT devices, without having to bother about setting up or managing any underlying infrastructure. You can use the IoT Core service to build IoT applications, using a combination of various AWS services as well, such as AWS Lambda, Amazon Elasticsearch, Amazon machine learning, and so on.
- **AWS IoT Device Management:** The AWS IoT Device Management service allows you to register, organize, and manage a large number of IoT devices, easily. You can use this service to on-board devices in bulk and then manage them all, using a single pane of glass view.

- **AWS Greengrass:** AWS Greengrass is a software service designed to execute Lambda functions locally, on your IoT devices. In addition to this, you can also use Greengrass to sync data between the device and the IoT Core, using data caching along with other functionalities, such as ML inference, messaging, and so on.
- **AWS IoT Analytics:** Connecting and managing billions of IoT devices is one task, and querying the large IoT data set is quite another. AWS IoT Analytics is a completely managed service that allows you to run analytics on extremely large volumes of IoT data, without having to configure or manage an underlying analytics platform. Using this service, you obtain better insights on your devices, as well as build resilient IoT applications.
- **AWS IoT Button:** The AWS IoT Button is a Wi-Fi enabled, programmable button which enables you to write and integrate an IoT application, without having to know about any device-specific code.
- **AWS IoT Device Defender:** With so many devices to manage and maintain, it is equally important to safeguard the devices against malicious attacks. AWS IoT Device Defender is a managed service that allows you to secure, manage, and audit remote devices against a set of security rules and policies. If any deviations are found, IoT Device Defender triggers appropriate notifications for them.
- **Amazon FreeRTOS:** Amazon FreeRTOS is a custom operating system built specifically for small, low-powered edge devices or microcontrollers. The operating system is based on the FreeRTOS kernel and helps to easily connect and manage devices with the AWS IoT service.

With this, we come to the end of this section. In the next section, we will learn a bit about the AWS IoT Core service in detail, along with a simple and easy-to-follow getting started guide.

Getting started with AWS IoT Core

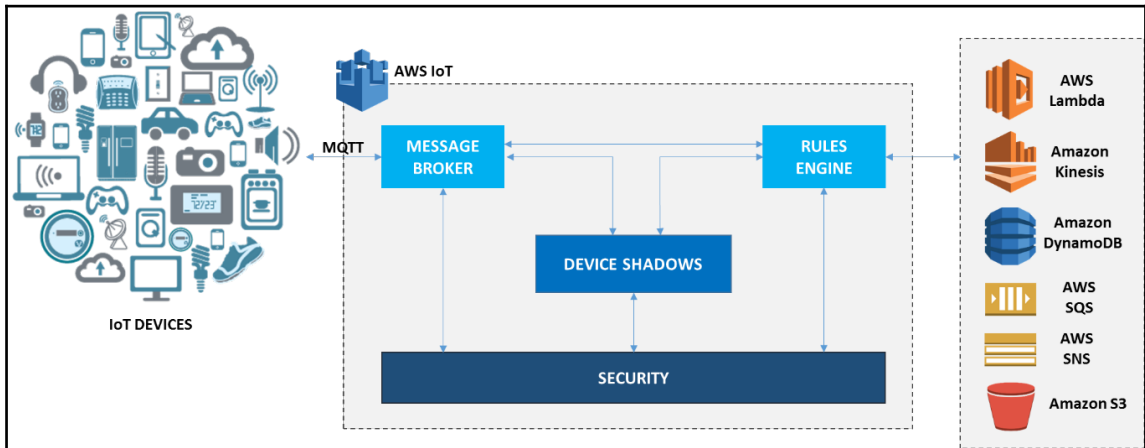
With a brief understanding of the AWS IoT suite of services covered, we can now dive deep into the world of the AWS IoT Core! However, before we get started with some actual hands-on projects, here is a quick look at some important AWS IoT Core concepts and terminologies:

The AWS IoT Core service provides bidirectional communication between devices and the AWS cloud, using a set of components described in the following list:

- **Device gateway:** This provides a secure mechanism for the IoT device to communicate with the AWS IoT service.
- **Device shadow:** A device shadow is a persistent representation of your IoT device on the cloud. A JSON-based document stores the current state of your device, which you can use to sync with the cloud.
- **Message broker:** The message broker provides a secure and reliable channel, using which the IoT device can communicate with the cloud. The broker is based on a publisher–subscriber model and can be used to leverage either the standard MQTT protocol, or the advanced MQTT over WebSockets for communication.
- **Registry:** Registry is a service that is used to securely register your IoT device with the cloud. You can use the registry to associate certificates and MQTT client IDs with your devices.
- **Groups:** Groups are logical containers used to group together similar devices in order to effectively manage them. You can use groups to propagate permissions and perform bulk actions on your connected devices.
- **Rules:** The rules engine service in AWS IoT Core provides a mechanism which enables you to process IoT data using simple SQL queries. You can additionally write rules that can integrate AWS IoT Core with other AWS services, such as AWS Lambda, Amazon S3, Amazon Kinesis, and so on.

Here is how it all fits together! You start off by preparing a device for connection with the AWS IoT Core. This involves creating a set of certificates that essentially authenticates the device when it connects to the AWS IoT Core. Once connected, the device starts publishing its current state in a JSON format using the standard MQTT protocol. These messages are sent to the message broker, which essentially routes them to their respective subscribing clients, based on the message's topic.

You can even create one or more rules to define a set of actions based on the data contained within the messages. When a particular data matches the configured expression, the rules engine invokes that particular action, which can be anything from sending the data to a file in Amazon S3 to processing the data using AWS Lambda or Amazon Kinesis. The following is a representation of these components put together:



Keeping this in mind, let's look at how you can connect your IoT device with the AWS IoT Core!

Connecting a device to AWS IoT Core

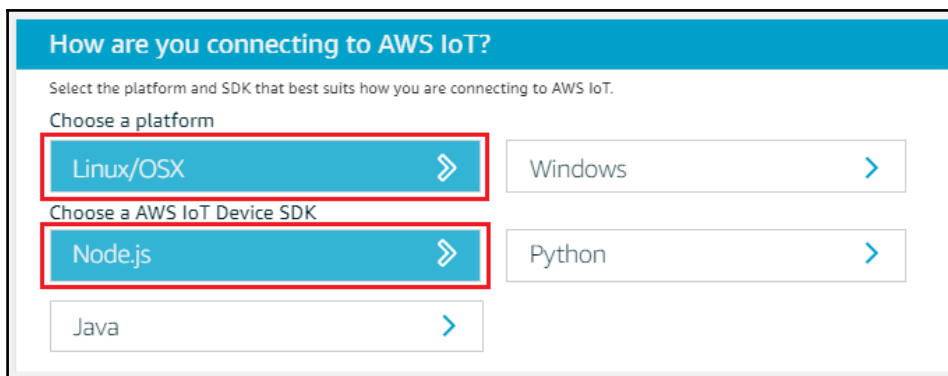
AWS IoT supports a wide variety of specialized IoT-embedded devices and microcontrollers that you can connect to. However, for simplicity, you can also simulate an IoT device using either a locally set up virtual machine or an EC2 instance, as well. For this section, we will be using a simple Ubuntu-based virtual machine, hosted using VirtualBox. The virtual machine has the basic operating system packages installed in it and runs off a 512 MB RAM and 1 CPU core allocation with a 10 GB disk. Ensure that your virtual machine has an open internet connectivity and a valid hostname set, before you proceed with any further steps.

The following list demonstrates the simulated IoT device's configuration for your reference:

- **CPU:** 1 CPU
- **RAM:** 512 MB
- **Operating System:** Ubuntu Server 16.04.2 LTS (Xenial) x86_64 architecture
- **Packages:** Core server packages along with vim, node, npm, git, wget

Once the device or virtual machine is prepped, we are good to connect with the AWS IoT Core:

1. From the AWS Management Console, filter and select the **AWS IoT** service using the **Filter** provided. Alternatively, select this URL, <https://console.aws.amazon.com/iot/home> to launch the AWS IoT console.
2. Select the **Get started** option to continue.
3. Once logged into the console, select the **Onboard** option from the navigation pane on the left-hand side of the console. Here, you can opt to get started with configuring your first device with the IoT service as well as other options, such as configuring the **AWS IoT Button** or getting started with the **AWS IoT Starter Kit**. For this section, select the **Get started** option under the **Configure a device** section.
4. The **Get started** option is a simple three-step process that involves first registering your device, followed by downloading a set of credentials and SDKs for the device to communicate with the IoT Core, and finally testing to check whether the device is successfully connected or not.
5. Select **Linux/OSX** from the **Choose a platform** option followed by **Node.js** from the **Choose a AWS IoT Device SDK**, as shown in the following screenshot. Note here, you can alternatively select the **Java** or **Python** SDKs as well; however, the rest of this particular use case will be based only upon Node.js:



6. Once the appropriate platform and IoT SDK are selected, click on **Next** to continue.
7. The next step involves the registration of a *thing* or in our case, the IoT device itself. Start off by providing a suitable **Name** for your thing and then select the **Show optional configuration** option.
8. In the **Apply a type to this thing** section, select the **Create type** option. A **Thing Type** simplifies managing IoT devices by providing a consistent registry data for things that share a particular type. Provide a suitable **Name** and an optional **Description** for your **Thing Type** and select **Create thing type** when done.
9. Here's what the final configuration should look like. In my case, I've created a **Thing Type** called **dummyIoTDevice** for logically classifying all virtual machine-based IoT devices together. Select the **Create Thing** option once completed:

Register a thing

This step creates an entry in the thing registry and a thing shadow for your device.

Name

myThingy

Hide optional configuration ▾

Apply a type to this thing

Using a thing type simplifies device management by providing consistent registry data for things that share a type. common set of attributes, which describe the identity and capabilities of your device, and a description.

Thing Type

dummyIoTDevice ▾

Create a type

10. With the thing successfully created, we now need to establish the connection between the thing and AWS IoT Core. To do so, select the newly created thing tile from the **Things** console to view the thing's various configurations. Among the important options is the **Security** option. Go ahead and select the **Security** option from the navigation pane.
11. Here, you can create and associate the necessary certificates, as well as policies that will be required for **Thing** to communicate with the IoT Core. Select the **Create certificate** option to begin with.

12. The necessary certificates are created automatically by AWS Core. Download these files and save them in a safe place. Certificates can be retrieved at any time, but the private and public keys *cannot be retrieved* after you close this page:
- **A certificate for this thing:** `xyz.cert.pem`
 - **A public key:** `xyz.public.key`
 - **A private key:** `xyz.private.key`

In addition, you will need to download the *root CA* for AWS IoT from Symantec. You can do that by selecting the following URL:

<https://www.symantec.com/content/en/us/enterprise/verisign/roots/VeriSign-Class%203-Public-Primary-Certification-Authority-G5.pem>

Remember to select the **Activate** option to successfully activate the keys. Once done, select the **Attach a policy** option, as shown in the following screenshot:

The screenshot shows a web interface for downloading certificates. At the top, it says "In order to connect a device, you need to download the following:". Below this is a table with three rows, each representing a different certificate file. Each row has a description, a filename, and a "Download" button. The buttons are highlighted with red boxes. Below the table, there is a section titled "You also need to download a root CA for AWS IoT from Symantec:" followed by a "Download" button, also highlighted with a red box. At the bottom left, there is a large blue "Activate" button, highlighted with a red box. At the bottom right, there is a "Done" button and a large blue "Attach a policy" button, which is also highlighted with a red box.

In order to connect a device, you need to download the following:		
A certificate for this thing	75e96f81a1.cert.pem	Download
A public key	75e96f81a1.public.key	Download
A private key	75e96f81a1.private.key	Download

You also need to download a root CA for AWS IoT from Symantec:

A root CA for AWS IoT [Download](#)

[Activate](#)

[Done](#) [Attach a policy](#)

13. Since this is our first time working with the IoT Core, we will be required to create a new policy from scratch. The policy will be used to authorize the certificates we created in the previous step. Select the **Create new policy** option to get started.
14. In the **Create a Policy** page, start by providing a suitable **Name** for your new policy. Once completed, you can use either the *basic* or *advanced* mode to create your IoT policy. For simplicity, select the **Advanced mode** option and paste the following policy snippet as shown:

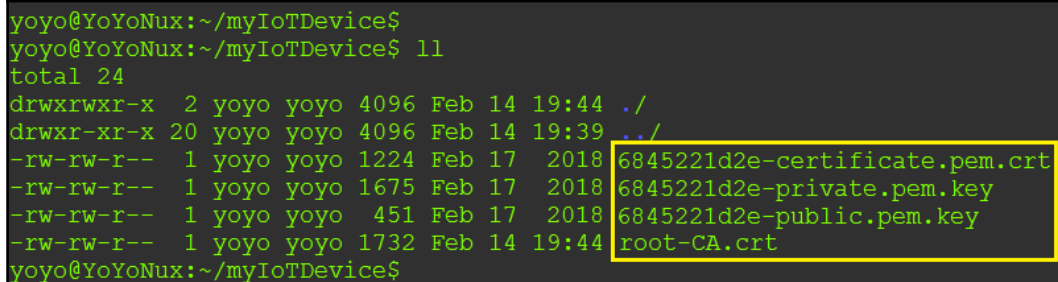
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:*",
      "Resource": "*"
    }
  ]
}
```

The following policy grants all devices permission to connect, publish, and subscribe to the AWS IoT message broker. You can alternatively tweak this policy as per your requirements, as well.

15. Once done, select the **Create** option to complete the policy creation process.

With this step completed, we are but a few steps away from establishing the connection between our IoT device and the AWS IoT Core.

16. With the necessary policy created and the certificates downloaded, we now need to copy these to our IoT device, in this case the Ubuntu virtual machine. You can use any SCP tool to perform this activity, such as WinSCP, as well. Here is a screenshot of the files on my Ubuntu virtual machine:



```
yoyo@YoYoNux:~/myIoTDevice$
yoyo@YoYoNux:~/myIoTDevice$ ll
total 24
drwxrwxr-x  2 yoyo yoyo 4096 Feb 14 19:44 ./
drwxr-xr-x 20 yoyo yoyo 4096 Feb 14 19:39 ../
-rw-rw-r--  1 yoyo yoyo 1224 Feb 17 2018 6845221d2e-certificate.pem.crt
-rw-rw-r--  1 yoyo yoyo 1675 Feb 17 2018 6845221d2e-private.pem.key
-rw-rw-r--  1 yoyo yoyo  451 Feb 17 2018 6845221d2e-public.pem.key
-rw-rw-r--  1 yoyo yoyo 1732 Feb 14 19:44 root-CA.crt
yoyo@YoYoNux:~/myIoTDevice$
```



For this scenario, I've called the downloaded Symantec Root CA file `root-CA.crt`.

Once the files are copied over to a destination folder in your IoT device, you are now ready to test the connectivity, but in order to do that, we will first need to install and configure the AWS IoT Device SDK on our IoT device.

Getting started with AWS IoT Device SDK

The AWS IoT Device SDK is a quick and easy way to connect your IoT devices with AWS IoT Core. To date, AWS provides IoT Device SDKs for Node.js, Java, Python, and Embedded C. For this particular section, we will be connecting our dummy IoT device with AWS IoT, using the Node.js SDK.

Before we get things started, ensure that you have the latest versions of node and NPM installed and running on your device. Since we are simulating an IoT device using an Ubuntu virtual machine, you can use the following commands to install and verify node and NPM versions:

```
# sudo apt-get update
# sudo apt-get install nodejs npm
# node -v && npm -v
```

Once the required packages are installed, we now need to install the AWS IoT Device SDK itself. Type in the following command as shown:

```
# npm install aws-iot-device-sdk
```

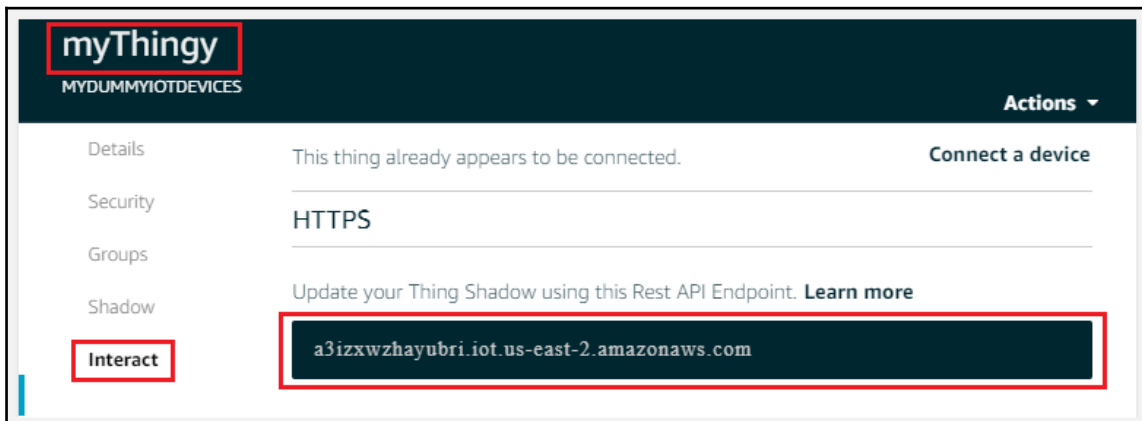
With the SDK installed, we can now begin testing the connectivity with a simple Node.js program. Copy and paste the following code snippet in a new `.js` file on your IoT device:

```
# vi test1.js
//Connection parameters
var awsIot = require('aws-iot-device-sdk');
var device = awsIot.device({
  keyPath: '<PATH_TO_PRIVATE.PEM.KEY>',
  certPath: '<PATH_TO_CERTIFICATE.PEM>CRT>',
  caPath: 'root-CA.crt',
  clientId: '<THING_NAME>',
  region: '<REGION>',
  host: '<IoT_DEVICE_REST_API_ENDPOINT>'
});
```

```
//Connection parameters end

//Device Object
device
.on('connect', function () {
  console.log('Yaaa! We are connected!');
});
```

What does the code do? Well for starters, the first section of the code is simply where you pass the required private key and certificates downloaded from the earlier steps, along with a few other configuration items, such as the `clientId`, the AWS region where the IoT Core is set up and finally, the `host`, which is basically a unique REST API endpoint for your device. You can find this endpoint by selecting your newly created IoT device from the AWS IoT dashboard and selecting the **Interact** tab, as shown in the following screenshot:



The second part of the code is where we use the configured parameters to actually connect to the AWS IoT Core. If the connection is successful, it will print a simple message as shown. To run the code, simply type in the following command:

```
# node test1.js
```

```
yoyo@YoYoNux:~/myIoTDevice$
yoyo@YoYoNux:~/myIoTDevice$ node test1.js
Yaaa! We are connected!
```

With the device now successfully connecting with the AWS IoT Core, let's look at a few other examples that you can use to interact with the message broker service. To start off, let's see how we can use the AWS IoT Device SDK to subscribe to a topic and print back any message that gets published to it.

Copy and paste the following code snippet below the connection parameters:

```
# vi test2.js
//Connection parameters
. . . . .
//Connection parameters end

//Device Object
device
.on('connect', function () {
  console.log('Yaaa! We are connected!');
  device.subscribe('Topic0');
  console.log('Subscribed to Topic');
});

device
.on('message', function (topic, payload) {
  console.log('Received following message: ', topic, payload.toString());
});
```

The following code subscribes the device to a topic named `Topic0`. Once the subscription is completed, the code will display any message that is published to it.

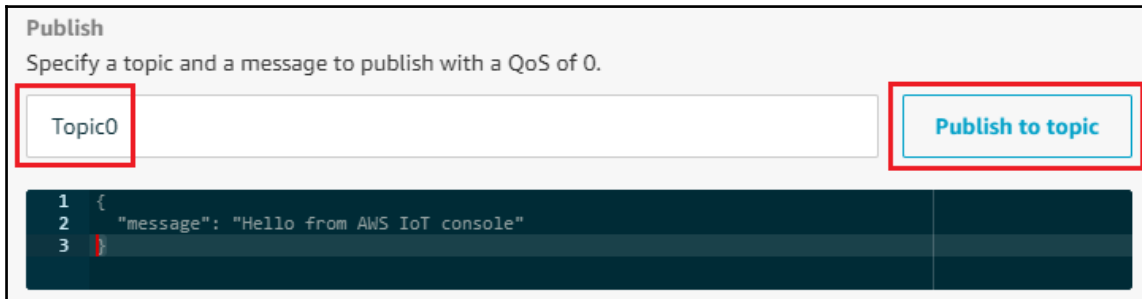
With the code in place, save the file and run the program using the following code:

```
# node test2.js
```

Next, let us look at how we can publish a message to the newly created topic:

1. To publish a message to the following topic, we will use the MQTT client provided by AWS IoT Core itself. To do so, from the AWS IoT Core dashboard, select the **Test** option from the navigation pane.
2. Using the MQTT client, you can subscribe, as well as publish, to a topic. Click on the **Publish to topic** option.

3. In the **Publish** section, provide the topic name (in this case, `Topic0`) where you want to publish the message and click on **Publish to topic**, as shown in the following screenshot:



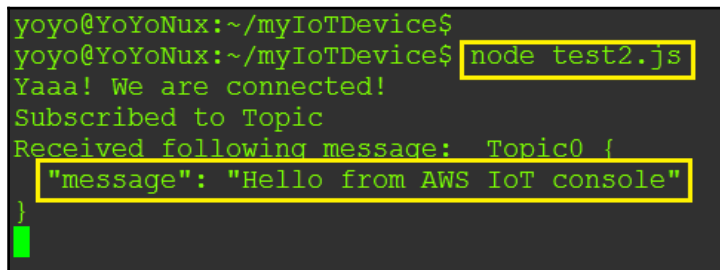
Publish
Specify a topic and a message to publish with a QoS of 0.

Topic0

Publish to topic

```
1 {  
2   "message": "Hello from AWS IoT console"  
3 }
```

Check the device Terminal for the corresponding message. You should see the following output:



```
yoyo@YoYoNux:~/myIoTDevice$  
yoyo@YoYoNux:~/myIoTDevice$ node test2.js  
Yaaa! We are connected!  
Subscribed to Topic  
Received following message: Topic0 {  
  "message": "Hello from AWS IoT console"  
}
```

With the device now able to subscribe to a topic, you can also get the device to publish a message to a topic as well.

Once again, create a new file and copy-paste the following code snippet below the connection parameters:

```
# vi test3.js  
//Connection parameters  
. . . . .  
//Connection parameters end  
  
//Device Object  
device  
.on('connect', function () {  
  console.log('Yaaa! We are connected!');  
  device.subscribe('Topic0');  
  console.log('Subscribed to Topic');
```

```
var msg = "Hello from IoT Device!";
    device.publish('Topic0', msg);
    console.log ("Publishing message: "+msg);
});

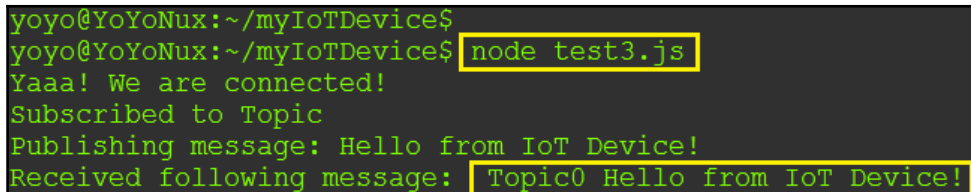
device
.on('message', function (topic, payload) {
console.log('Received following message: ', topic, payload.toString());
});
```

As you can see from the following code, we have simply added a `publish()` method that will publish a custom message to a pre-defined topic, in this case, `Topic0`.

Save the code and run the program using the following command:

```
# node test3.js
```

You should get the following output from the device Terminal:



```
yoyo@YoYoNux:~/myIoTDevice$
yoyo@YoYoNux:~/myIoTDevice$ node test3.js
Yaaa! We are connected!
Subscribed to Topic
Publishing message: Hello from IoT Device!
Received following message: Topic0 Hello from IoT Device!
```

Simple, isn't it! You can use the same concepts on a real IoT device as well, with only a few minor changes here and there. Here is an example code snippet that you can use to generate dummy data and publish the data to an MQTT topic.

Create a new file and paste the following code snippet below the connection parameters, as done throughout this section:

```
# vi test4.js
//Connection parameters
. . . . .
//Connection parameters end

var uuid = require('node-uuid');
var numbers = new Array(10);
function getRandomInt(min, max) {
    return Math.floor(Math.random() * (max - min + 1)) + min;
}
```

```

device
.on('connect', function () {
console.log('Yaaa! We are connected!');
device.subscribe('Topic0');
console.log('Subscribed to Topic');

for (var i = 0; i < numbers.length; i++)
{
for (var j = 0; j < numbers.length; j++)
{
numbers[i] = getRandomInt(0,1);
uuid[i] = uuid.v4()
}
var msg = "{" + "uuid:" + uuid[i] + "," + "state:" +
numbers[i] + "}";
device.publish('Topic0', msg);
console.log ("Publishing message: " + uuid[i] + numbers[i]);
}

});

device
.on('message', function (topic, payload) {
console.log('Received following message: ', topic, payload.toString());
});

```

The following code snippet leverages the `node-uuid` module to randomly generate long strings of UUIDs. With each UUID record generated, a corresponding random value of either 0 or 1 is printed which denotes the *state* of the UUID. You can control the number of records generated by adjusting the value of the array object. By default, the following code will publish 10 records in a proper JSON to the MQTT topic, as shown in the following code snippet:

```

{
  "uuid": "357d6212-3444-4f55-9784-93a265905289",
  "state": 0
}
{
  "uuid": "ad8cb61b-f29d-4a84-a7eb-42633dd3a3c2",
  "state": 0
}
. . . . .
{
  "uuid": "19492c45-0cf7-4468-b275-5b7b1bdf3a64",
  "state": 1
}

```

Once the code is in place, simply execute it using the following command:

```
# node test4.js
```

```
yoyo@YoYoNux:~/myIoTDevice$
yoyo@YoYoNux:~/myIoTDevice$ node test4.js
Yaaa! We are connected!
Subscribed to Topic
Publishing message: 357d6212-3444-4f55-9784-93a265905289 0
Publishing message: ad8cb61b-f29d-4a84-a7eb-42633dd3a3c2 0
Publishing message: 19492c45-0cf7-4468-b275-5b7b1bdf3a64 0
Publishing message: 57fb4d49-7188-4e92-894d-43c7be78f852 1
Publishing message: 3c8bc37c-a0fa-40c1-a4f0-59a5d6719fa6 0
Publishing message: 56725fb4-ab7d-44d3-9d19-779c0a846c4f 1
Publishing message: b1d9a6e1-9eb0-43f2-9bfd-c34b4fdbb5ec 1
Publishing message: 6fcd6012-ff99-4dc3-952b-d83c3b71449a 1
Publishing message: b6d1a1c0-1e50-46a6-bbb0-79b8d022fa72 0
Publishing message: delf6633-c67d-4471-b1ca-8824b6b331e9 0
Received following message: Topic0 {"uuid":"357d6212-3444-4f55-9784-93a265905289","state":0}
Received following message: Topic0 {"uuid":"ad8cb61b-f29d-4a84-a7eb-42633dd3a3c2","state":0}
Received following message: Topic0 {"uuid":"19492c45-0cf7-4468-b275-5b7b1bdf3a64","state":0}
Received following message: Topic0 {"uuid":"57fb4d49-7188-4e92-894d-43c7be78f852","state":1}
Received following message: Topic0 {"uuid":"3c8bc37c-a0fa-40c1-a4f0-59a5d6719fa6","state":0}
Received following message: Topic0 {"uuid":"56725fb4-ab7d-44d3-9d19-779c0a846c4f","state":1}
Received following message: Topic0 {"uuid":"b1d9a6e1-9eb0-43f2-9bfd-c34b4fdbb5ec","state":1}
Received following message: Topic0 {"uuid":"6fcd6012-ff99-4dc3-952b-d83c3b71449a","state":1}
Received following message: Topic0 {"uuid":"b6d1a1c0-1e50-46a6-bbb0-79b8d022fa72","state":0}
Received following message: Topic0 {"uuid":"delf6633-c67d-4471-b1ca-8824b6b331e9","state":0}
```

With this, we come to the end of this particular section. In the next section, we will be looking at how we can integrate the AWS IoT Core with other AWS Services, using simple IoT rules.

Working with IoT rules

One of the most fascinating features recently provided with the AWS IoT Core is IoT rules. With IoT rules, you can basically provide your connected devices with the ability to interact with other AWS services. IoT rules provide a predefined set of rules that allow you to perform a variety of tasks, such as those listed here:

- Write data received from a device to an Amazon DynamoDB table
- Send a push notification to all users using Amazon SNS
- Publish data to an Amazon SQS queue
- Invoke a Lambda function to perform some data transformation
- Process data from devices using Amazon Kinesis, and much more

An IoT rule also provides you with an added functionality which enables you to query and filter device data, as well use simple SQL commands. Based on the SQL statement execution, you can then either invoke a success or an error action:

1. To get started with IoT rules, select the **Act** option from the AWS IoT Core console. Since this is our first time, click on **Create a rule** to proceed with the next steps.
2. On the **Create a rule** page, start off by providing a suitable **Name** and a **Description** for your rule. In this scenario, we will be creating an IoT rule that will write all device data to a DynamoDB table.
3. Next, from the **Using SQL version** drop-down list, select an appropriate SQL version for this rule. By default, the **2016-03-23** will be selected.
4. Next, we form the **Rule query statement**. This is used to filter out particular messages from a large set of device data. In this case, we want all the data from our demo device to be written to a DynamoDB table, so in the **Attribute** field, type in an ***** which indicates all fields followed by a valid topic name in the **Topic filter** option. You can also set an optional **Condition** to match your query.

Here's what the final query should look like:

Using SQL version ?

2016-03-23

Rule query statement

```
SELECT * FROM 'Topic0'
```

Attribute ?

*

Topic filter ?

Topic0

Condition ?

e.g. temperature > 75

5. With the query in place, the next step is to associate one or more actions with your IoT rule. From the **Set one or more actions** section, select the **Add action** option to get started.
6. On the **Select an action** page, you can browse and select one of the predefined action templates. In this case, we will be selecting the **Insert a message into a DynamoDB table** action, which allows you to write all, or part of, an MQTT message to a DynamoDB table. Once selected, click on the **Configure action** to proceed.
7. This will take you to the **Configure action** page where you can either select an existing DynamoDB table, or alternatively create one as well. For this use case, we will be relying on the *UUID and state* code that we last executed on our sample device. In order to split and write the message data to their individual columns, ensure that the DynamoDB table that you create matches the following settings:
 - **Partition key:** `uuid`
 - **Sort key:** `state`
 - **Read capacity units:** 5
 - **Write capacity units:** 5

Create DynamoDB table

Tutorial ?

DynamoDB is a schema-less database that only requires a table name and primary key. The table's primary key is made up of one or two attributes that uniquely identify items, partition the data, and sort data within each partition.

Table name* ⓘ

Primary key* Partition key

String ⓘ

☒ Add sort key

String ⓘ

8. Once the table is created, select it from the **Table name** drop-down list. Next, fill in the corresponding expressions in the **Hash key value** and **Range key value** fields, as depicted in the following screenshot. These expressions will write the message values into their corresponding DynamoDB table columns:

The table must contain Hash and Range keys.

*Table name		
uuid-state-scoredata		Create a new resource
*Hash key	*Hash key type	*Hash key value
uuid	STRING	\${uuid}
Range key	Range key type	Range key value
state	STRING	\${state}

9. Ensure that you also create and assign an IAM Role that will grant write AWS IoT access to the DynamoDB table. With the table and IAM Role assigned, click on **Add action** to complete configuring the action. You can correspondingly use the same process to add multiple actions to a single IoT rule. For example, write the device messages to a DynamoDB table as well as to an SQS queue, and so on. With the rule ready, click on **Create rule** to complete the process.
10. Now, to test the rule. Run the `uuid-state` code that we ran in the earlier section. Ensure that the topic names in both the code as well as in the IoT rule match, otherwise the code will simply not write anything to the DynamoDB table:

```
# node test4.js
```

With the code executing, check the DynamoDB table for the data. You should see a similar output as shown in the following screenshot:



Scan: [Table] uuid-state-scoredata: uuid, state ^ Viewing 1 to 10 items

Scan [Table] uuid-state-scoredata: uuid, state ^

+ Add filter

Start search

	uuid	state	payload
☐	7cb60fc6-faf2-4d56-b010-e0a535e04a57	0	{ "state" : { "N" : "0" }, "uuid" : { "S" ...
☐	5b58f557-5a4b-465f-804a-65e5b7fc5f02	0	{ "state" : { "N" : "0" }, "uuid" : { "S" ...

With this, we come to the end of AWS IoT Core. Make sure you clean up and delete the DynamoDB table once the testing completes, to avoid any unnecessary costs. In the next section, we will be exploring yet another powerful AWS IoT service called AWS Greengrass.

Introducing AWS Greengrass

AWS Greengrass is a form of edge computing service that extends the cloud's functionality to your IoT devices by allowing data collection and analysis closer to its point of origin. This is accomplished by executing AWS Lambda functions locally on the IoT device itself, while still leveraging the cloud for management and analytics purposes.

How does this help a business? Well to start with, using AWS Greengrass you are now able to respond to locally generated events in near real time! With Greengrass, you can program your IoT devices to locally process and filter data and only transmit the important chunks back to AWS for analysis. This also has a direct impact on the costs as well as the amount of data transmitted back to the cloud.

Here's a brief look at a few of the necessary components that go into the workings of AWS Greengrass:

- **Greengrass Core (GGC) software:** The Greengrass Core software is a packaged module that consists of a runtime to allow executions of Lambda functions, locally. It also contains an internal message broker and a deployment agent that periodically notifies the AWS Greengrass service about the device's configuration, state, available updates, and so on. The software also ensures that the connection between the device and the IoT service is secure with the help of keys and certificates.
- **Greengrass groups:** A Greengrass group is a collection of Greengrass Core settings and definitions that are used to manage one or more Greengrass-backed IoT devices. The groups internally comprise a few other components, namely:
 - **Greengrass group definition:** A collection of information about your Greengrass group
 - **Device definition:** A collection of IoT devices that are a part of a Greengrass group
 - **Greengrass group settings:** Contains connection as well as configuration information along with the necessary IAM Roles required for interacting with other AWS services
 - **Greengrass Core:** The IoT device itself
- **Lambda functions:** A list of Lambda functions that can be deployed to the Greengrass Core.
- **Subscriptions:** A collection of a message source, a message target and an MQTT topic to transmit the messages. The source or targets can be either the IoT service, a Lambda function or even the IoT device itself.
- **Greengrass Core SDK:** Greengrass also provides an SDK which you can use to write and run Lambda functions on Greengrass Core devices. The SDK currently supports Java 8, Python 2.7, and Node.js 6.10.

With this key information in mind, let's go ahead and deploy our very own Greengrass Core on an IoT device.

Connecting a device to Greengrass Core

The steps required to connect an IoT device with AWS Greengrass are very similar to those we performed during the setup of the AWS IoT Core. In this section, we are going to extend our dummy IoT device (Ubuntu Server on a virtual machine) with Greengrass using the AWS Management Console:

1. To get started, from the AWS IoT console, select the **Greengrass** option from the navigation pane.
2. Setting up Greengrass involves a three-step process starting with creating and configuring a **Greengrass group**, followed by adding a **Greengrass Core** to the group and finally, by adding the IoT device to the group. To get going, click on the **Get Started** option under the **Define a Greengrass Group** tile.
3. On the **Set up your Greengrass Group** page, select the **Use easy creation** option. This process will automatically provision a Core in the registry, use default settings to generate a new group, and provide your core with a new certificate and a key pair.
4. Type in a suitable group **Name** for the Greengrass group and click on **Next** to proceed.
5. You can optionally choose to apply a type to this group by selecting the **Thing Type** from the drop-down box, as shown in the following screenshot. In this case, we already have a **Thing Type** defined from our previous exercises so we are going to use this. Click on **Next** to continue:

SET UP YOUR GREENGRASS GROUP

Every Group needs a Core to function

Every Greengrass Group requires a device running Core software. It enables communication between Devices, local Lambda functions, and AWS cloud computing services. Adding information to the Registry is the first step in provisioning a device as your Greengrass Core.

Name

MyGreengrassGrp_Core

Hide optional configuration ▲

Apply a type to this thing

Using a thing type simplifies device management by providing consistent registry data for things that share a type. Types provide things with a common set of attributes, which describe the identity and capabilities of your device, and a description.

Thing Type

dummyIoTDevice ▼

Create a type

6. Since we are using the *easy creation* method, AWS runs a scripted action that basically performs the following set of tasks for us:
 - Create a new Greengrass group in the cloud
 - Provision a new core in the IoT registry and add to the group
 - Generate a public and private key set for your core
 - Generate a new security certificate for the core using the keys
 - Attach a default security policy to the certificate
7. Click on **Create Group and Core** to proceed with the scripted install.
8. Finally, on the **Connect your Core device** page, download the core-specific certificates and config file as a TAR resource by selecting the **Download these resources as a tar.gz** option. You will also need to download an appropriate version of the Greengrass Core software to run on your IoT device. Since we are performing all of these activities on a Ubuntu-based virtual machine, select the **x86_64_Ubuntu** option from the **Greengrass Core software** drop-down list and download it. Once done, click **Finish** to exit the setup.

With both the Greengrass Core software and the necessary Greengrass certificates downloaded, we now have to transfer them to our IoT device using any SCP tool. Once transferred, run the following set of commands to set up and start the Greengrass Core:

1. First up, untar the Greengrass Core software using the following command.

```
# sudo tar -xzf greengrass-<PLATFORM-VERSION>.tar.gz -C /
```

2. Next, run the following command to untar and place the security files and certificates in the `greengrass` directory:

```
# sudo tar -xzf <UID>-setup.tar.gz -C /greengrass
```

3. Once the contents of both the TAR files are extracted, run the following command to download the Root CA certificate from Symantec:

```
# cd /greengrass/certs/  
# sudo wget -O root.ca.pem  
http://www.symantec.com/content/en/us/enterprise/verisign/roots/Ver  
iSign-Class%203-Public-Primary-Certification-Authority-G5.pem
```

Here is the final folder structure for your reference:

```
yoyo@YoYoNux:/greengrass$ tree -L 2
.
├── certs
│   ├── e93ddc4782.cert.pem
│   ├── e93ddc4782.private.key
│   ├── e93ddc4782.public.key
│   ├── README
│   └── root.ca.pem
├── config
│   └── config.json
├── ggc
│   ├── core -> packages/1.3.0
│   ├── deployment
│   ├── packages
│   └── var
└── ota
    ├── ota_agent -> ota_agent_v1.0.0
    └── ota_agent_v1.0.0
```

4. Once completed, run the following set of commands to create a dedicated user and group for Greengrass Core software:

```
# sudo adduser --system ggc_user
# sudo addgroup --system ggc_group
```

5. Next, update the host operating system and install a `sqlite3` package on it using the following commands:

```
# sudo apt-get update
# sudo apt-get install sqlite3
```

6. With all the pieces of the puzzle in place, we are now ready to finally start the Greengrass Core service on our IoT device. Type in the following command as shown:

```
# cd /greengrass/ggc/packages/1.3.0/
# sudo ./greengrassd start
```

You should get the following output on your Terminal, as shown in the following screenshot:

```
yoyo@YoYoNux:~$  
yoyo@YoYoNux:~$ cd /greengrass/ggc/packages/1.3.0/  
yoyo@YoYoNux:/greengrass/ggc/packages/1.3.0$  
yoyo@YoYoNux:/greengrass/ggc/packages/1.3.0$ sudo ./greengrassd start  
Setting up greengrass daemon  
Validating hardlink/softlink protection  
Validating execution environment  
Found cgroup subsystem: cpuset  
Found cgroup subsystem: cpu  
Found cgroup subsystem: cpuacct  
Found cgroup subsystem: blkio  
Found cgroup subsystem: memory  
Found cgroup subsystem: devices  
Found cgroup subsystem: freezer  
Found cgroup subsystem: net_cls  
Found cgroup subsystem: perf_event  
Found cgroup subsystem: net_prio  
Found cgroup subsystem: hugetlb  
Found cgroup subsystem: pids  
  
Starting greengrass daemon  
greengrass successfully started with PID: 2569  
yoyo@YoYoNux:/greengrass/ggc/packages/1.3.0$
```



In case of errors in connecting, you can also check the Greengrass runtime log file at the following location:

`/greengrass/ggc/var/log/system/runtime.log`.

Simple, isn't it! With two out of three steps completed, the final step left in completing the Greengrass connectivity is adding a device to the Greengrass group that we have created:

1. To do so, from the AWS IoT console, select the **Groups** option provided under the **Greengrass** section from the navigation pane. You should see your newly created **Greengrass group** present here. Select it.
2. Select **Devices** and click on the **Add your first Device** option to continue.
3. Here in the **Add a Device** page, you can opt to either **Create a new Device** or optionally **Use an existing IoT Thing as an Device**. Since we already have the IoT device registered from our earlier IoT setup, select the **Select an IoT Thing** option to proceed.
4. Select the name of the added IoT device and click on **Finish** to complete the process.

There you have it! You have successfully installed and connected your IoT device with AWS Greengrass! In the next section, we will test this deployment by running a simple Lambda function on it.

Running Lambda functions on AWS Greengrass

With the Greengrass Core software up and running on your IoT device, we can now go ahead and run a simple Lambda function on it! For this particular section, we will be leveraging an AWS Lambda blueprint that prints a simple Hello World message:

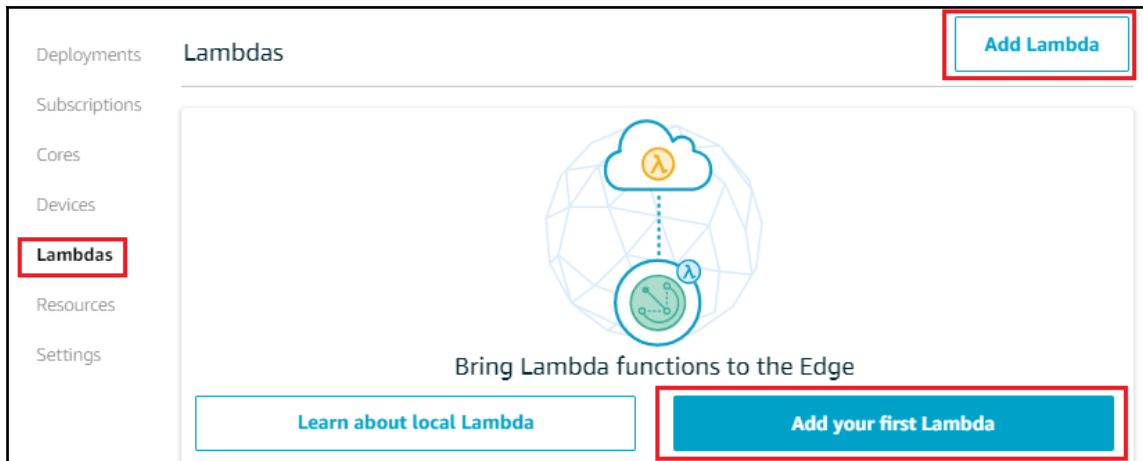
1. To get started, first we will need to create our Lambda function. From the AWS Management Console, filter out the **Lambda** service using the **Filter** option or alternatively, select this URL: <https://console.aws.amazon.com/lambda/home>.



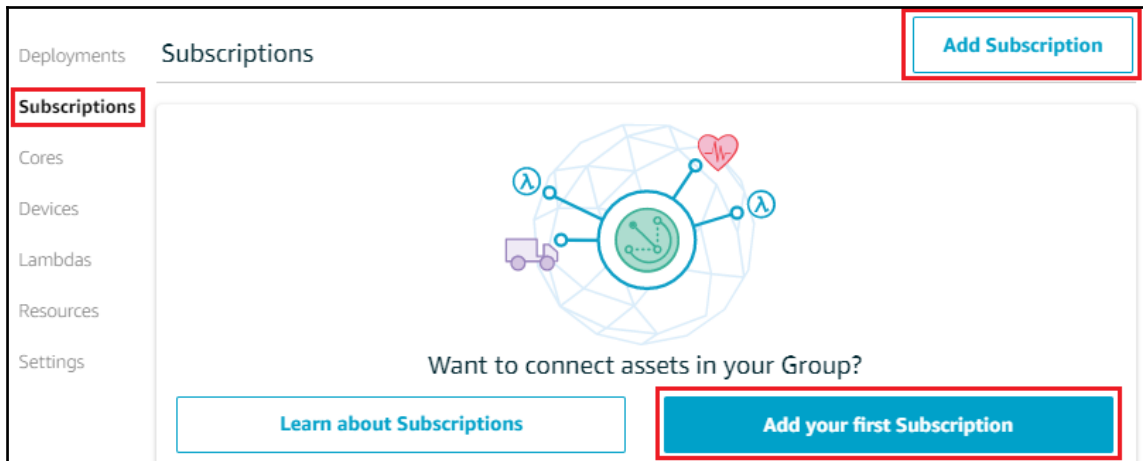
Ensure that the Lambda function is launched from the same region as that of the AWS Greengrass. In this case, we are using the US-East-1 (N. Virginia) region.

2. On the AWS Lambda console landing page, select the **Create function** option to get started.
3. Since we are going to be leveraging an existing function blueprint for this use case, select the **Blueprints** option provided on the **Create function** page.
4. Use the filter to find a blueprint with the name `greengrass-hello-world`. There are two templates present to date that match this name, one function is based on Python while the other is based on Node.js. For this particular section, select the `greengrass-hello-world` Python function and click on **Configure** to proceed.
5. Fill out the required details for the new function, such as a **Name** followed by a valid **Role**. For this section, go ahead and select the **Create new role from template** option. Provide a suitable **Role name** and finally, from the **Policy templates** drop-down list, select the **AWS IoT Button Permissions** role.
6. Once completed, click on **Create function** to complete the function's creation process. But before you move on to associating this function with your AWS Greengrass, you will also need to create a new *version* out of this function. Select the **Publish new version** option from the **Actions** tab.

7. Provide a suitable **Version description** text and click on **Publish** once done. Your function is now ready for AWS Greengrass.
8. Now, head back to the AWS IoT dashboard and select the newly deployed Greengrass group from the **Groups** option present on the navigation pane.
9. From the **Greengrass group** page, select the **Lambdas** option from the navigation pane followed by the **Add Lambda** option, as shown in the following screenshot:

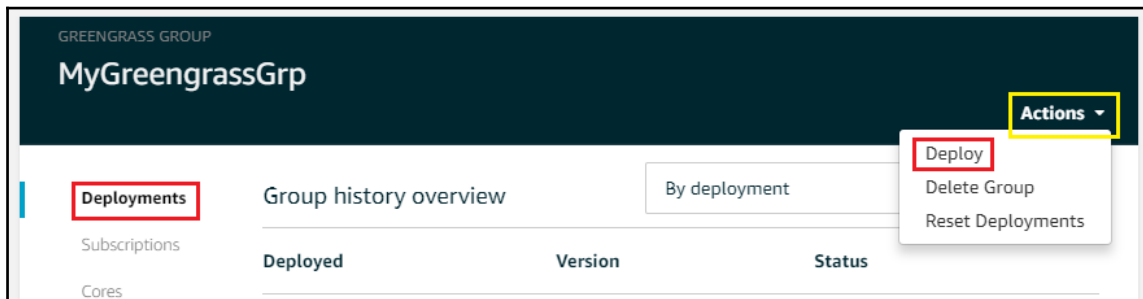


10. On the **Add a Lambda to your Greengrass group**, you can choose to either **Create a new Lambda function** or **Use an existing Lambda function** as well. Since we have already created our function, select the **Use existing function** option.
11. In the next page, select your **Greengrass Lambda function** and click **Next** to proceed. Finally, select the version of the deployed function and click on **Finish** once done.
12. To finish things, we will need to create a new subscription between the Lambda function (source) and the AWS IoT service (destination). Select the **Subscriptions** option from the same **Greengrass group** page, as shown. Click on **Add Subscription** to proceed:



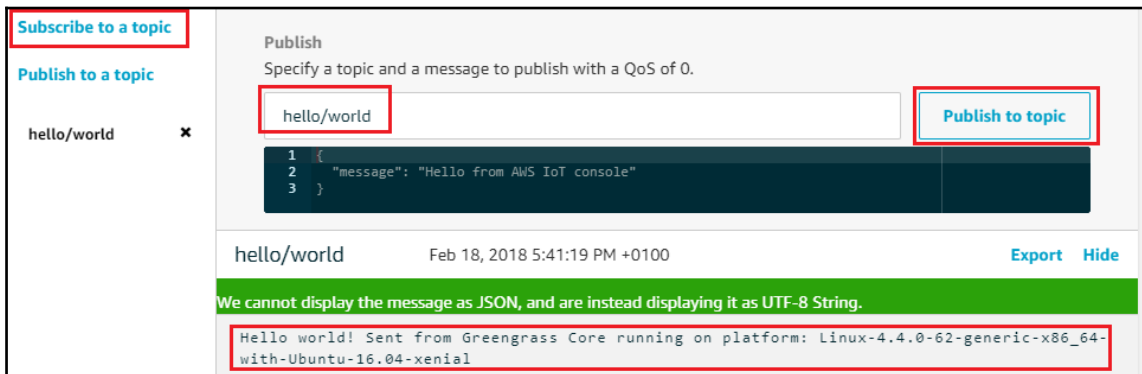
13. On the **Select your source and target** page, select the newly deployed Lambda function as the source, followed by the IoT cloud as the target. Click on **Next** once done. You can provide an **Optional topic filter** as well, to filter messages published on the messaging queue. In this case, we have provided a simple `hello/world` as the filter for this scenario. Click on **Finish** once done to complete the subscription configuration.

With all the pieces in place, it's now time to deploy our Lambda function over to the Greengrass Core. To do so, select the **Deployments** option and from the **Actions** drop-down list, select the **Deploy** option, as shown in the following screenshot:



The deployment takes a few seconds to complete. Once done, verify the status of the deployment by viewing the **Status** column. The **Status** should show **Successfully completed**.

With the function now deployed, test the setup by using the MQTT client provided by AWS IoT, as done before. Remember to enter the same `hello/world` topic name in the subscription topic field and click on **Publish to topic** once done. If all goes well, you should receive a custom Hello World message from the Greengrass Core as depicted in the following screenshot:

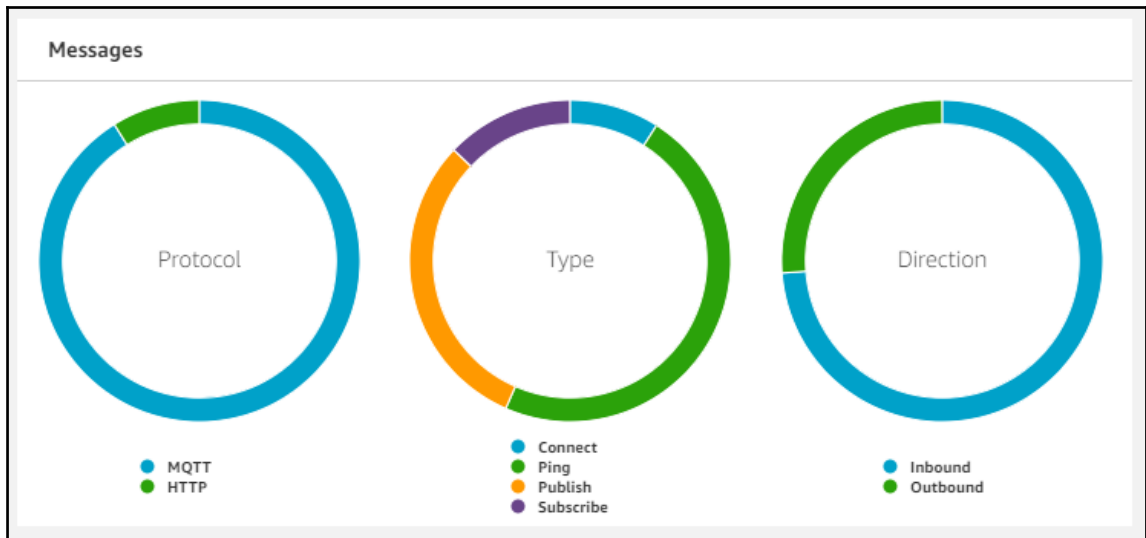


This was just a high level view of what you can achieve with Greengrass and Lambda. You can leverage Lambda for performing all kinds of preprocessing on data on your IoT device itself, thus saving a tremendous amount of time, as well as costs.

With this, we come to the end of this section. In the next section, we will be looking at a few simple ways of effectively monitoring your IoT devices.

Monitoring AWS IoT devices and services

AWS offers a variety of methods for monitoring both your IoT devices, as well as the IoT service and its calls. To get things started, let's first look at the simple device monitoring functionality provided by the AWS IoT dashboard itself. On the AWS IoT console page, select the **Monitor** option. Here, you can view a variety of graphs and data, such as the *number of successful connections* made to the AWS IoT service over the past hour, day, or week. You can even check the number of *messages* that were transmitted using either the MQTT or the HTTP protocol, as shown in the following screenshot:



You can also use the **Monitor** page to view the number of *messages published*, *rules executed*, and *shadow updates* performed.

In addition to this, you also have an option to enable logging for your AWS IoT service. To do so, select the **Settings** option from the navigation pane of the AWS IoT console. By default, logging of AWS IoT is *disabled*, however you can easily switch it on by selecting the **Edit** option provided under the **Logs** section. As messages from your IoT devices pass through the message broker and the rules engine, you can use the AWS IoT logs to process events and in turn, troubleshoot issues, both at the device as well as at the service's end.

You can choose between **Debug (most verbose)**, **Info**, **Warning**, and **Errors (least verbose)** levels of verbosity, depending on your logging requirements.

Summary

Well, like all good journeys, this book too has come to its end! I just wanted to take this time to say that it has really been a wonderful journey and experience writing this book!

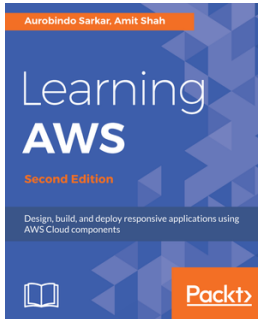
Although the book may seem a lot to read and grasp, trust me, this is all just a drop in the ocean! AWS continuously strives to evolve its services by adding more and more features to it, so much so, that today you have ready-to-use services for almost anything, including game development, AI, customer engagement, business productivity, just to name a few!

To quickly summarize all that we have learned so far: we started our journey with some interesting hands-on with the EC2 Systems Manager, shortly followed by two of my favorite services, the Elastic Beanstalk and the Elastic File System. We also covered a lot on security in the form of AWS WAF, AWS Shield, AWS CloudTrail, and AWS Config! Towards the end, we started exploring a few developer-based services in the form of AWS CodeDeploy, AWS CodeCommit, and AWS CodePipeline, to name a few. Finally, we ended the last few chapters on a high note by looking at the IoT and Analytics services in Amazon Redshift, Amazon EMR, AWS Data Pipeline and last, but not the least, AWS IoT!

Till next time, cheers!

Other Books You May Enjoy

If you enjoyed this book, you may be interested in these other books by Packt:



Learning AWS - Second Edition

Aurobindo Sarkar, Amit Shah

ISBN: 9781787281066

- Set up your AWS account and get started with the basic concepts of AWS
- Learn about AWS terminology and identity access management
- Acquaint yourself with important elements of the cloud with features such as computing, ELB, and VPC
- Backup your database and ensure high availability by having an understanding of database-related services in the AWS cloud
- Integrate AWS services with your application to meet and exceed non-functional requirements
- Create and automate infrastructure to design cost-effective, highly available applications



AWS Certified Developer - Associate Guide

Vipul Tankariya, Bhavin Parmar

ISBN: 9781787125629

- Create and manage users, groups, and permissions using AWS Identity and Access Management services
- Create a secured Virtual Private Cloud (VPC) with Public and Private Subnets, Network Access Control, and Security groups
- Get started with Elastic Compute Cloud (EC2), launching your first EC2 instance, and working with it
- Handle application traffic with Elastic Load Balancing (ELB) and monitor AWS resources with CloudWatch
- Work with AWS storage services such as Simple Storage Service (S3), Glacier, and CloudFront
- Get acquainted with AWS DynamoDB – a NoSQL database service
- Coordinate work across distributed application components using Simple Workflow Service (SWF)

Leave a review - let other readers know what you think

Please share your thoughts on this book with others by leaving a review on the site that you bought it from. If you purchased the book from Amazon, please leave us an honest review on this book's Amazon page. This is vital so that other potential readers can see and use your unbiased opinion to make purchasing decisions, we can understand what our customers think about our products, and our authors can see your feedback on the title that they have worked with Packt to create. It will only take a few minutes of your time, but is valuable to other potential customers, our authors, and Packt. Thank you!

Index

A

- Amazon CloudWatch metrics
 - used, for monitoring Amazon SNS 219
- Amazon CloudWatch
 - about 112
 - AllowedRequests 112
 - BlockedRequests 112
 - CountedRequests 112
- Amazon Elasticsearch
 - about 134
 - used, for analyzing CloudTrail Logs 135
- Amazon EMR
 - about 238
 - Apache Spark 241
 - applications 241
 - benefits 238
 - cluster connection 249
 - clusters, monitoring 254, 255
 - frameworks 240
 - job, executing on cluster 251
 - programs 241
 - storage 240
 - terminologies 239
 - working with 241, 243, 245, 248
- Amazon Redshift
 - about 256
 - cluster connection 261, 263
 - databases, working with 263, 267
 - executing 257, 260
 - tables, working with 263, 267
 - terminologies 256
 - URL, for COPY command 265
- Apache Spark
 - URL 241
- Application Load Balancer (ALB) 85
- application Load Balancer-based template

- URL 107
- AppSpec file
 - files 190
 - hooks 190
 - OS 190
 - setting up 190
 - version 190
- automation documents
 - triggering, with CloudWatch events 43, 45
 - triggering, with CloudWatch schedules 43, 45
 - used, for Linux instance patching 41, 43
 - working with 39
- AWS analytics suite of services
 - about 236
 - Amazon Athena 236
 - Amazon Elasticsearch Service 236
 - Amazon EMR 236
 - Amazon Kinesis 236
 - Amazon QuickSight 236
 - Amazon Redshift 236
 - AWS Data Pipeline 237
 - AWS Glue 237
- AWS CLI
 - used, for creating FIFO queue 227
- AWS Cloud9
 - URL 202
- AWS CloudTrail
 - about 112, 117
 - captured CloudTrail Logs and Events, filtering 122
 - captured CloudTrail Logs, viewing 122
 - CloudWatch alarms, deployment automating 132
 - custom metric filters, creating 130
 - key benefits 117
 - logs, analyzing with Amazon Elasticsearch 135
 - logs, monitoring with alarms 130
 - logs, monitoring with CloudWatch 126, 129, 130

- terminologies 118
- Trail, creating 119, 122
- Trail, modifying with AWS CLI 124, 126
- working with 119
- AWS Code Suite 172
- AWS CodeBuild 172
- AWS CodeCommit
 - about 172, 175, 178, 180
 - branches, working with 180
 - commits, working with 180
 - triggers, working with 180
- AWS CodeDeploy
 - about 173, 184
 - agent, configuring 187, 189
 - agent, installing 187, 189
 - application specification files 186
 - application, creating 193
 - applications 185
 - AppSpec file, setting up 190
 - CodeDeploy agent 186
 - deployment group, creating 193
 - deployments 185
 - terminologies 185
- AWS CodePipeline
 - about 173, 195
 - continuous delivery pipeline, creating 196
 - implementing 198, 201, 202
- AWS CodeStar
 - URL 202
- AWS Command Line Interface
 - reference link 64
- AWS Config
 - about 139, 141, 145
 - custom config rules, creating 145, 148
 - terminologies 140
- AWS Data Pipeline
 - about 270, 273, 275, 278
 - actions 272
 - activities 272
 - components 270
 - data nodes 271
 - definition files, working with 280, 283
 - resources 272
 - task runners 271
 - used, for backing up data 288, 291, 294

- used, for executing remote commands 284, 288
- AWS Greengrass
 - about 318
 - components 319
 - device, connecting 320, 322
 - Lambda function, executing 324, 327
- AWS IAM
 - enhancements 151
 - features 151
 - IAM policies, testing with IAM Policy Simulator 156
 - visual editor, used for creating IAM policies 152
- AWS IoT Core
 - AWS IoT Device SDK 308, 312
 - device gateway 302
 - device shadow 302
 - device, connecting 303, 308
 - groups 302
 - message broker 302
 - registry 302
 - rules 302
 - rules, working with 314, 317
- AWS IoT
 - devices, monitoring 327
 - services, monitoring 327
- AWS Organizations
 - about 159, 161, 166, 169
 - implementing 169
 - terminologies 160
- AWS Partner Network (APN) 202
- AWS Shield 113
- AWS Shield Advanced 113
- AWS Shield Standard 113
- AWS WAF dashboard 111
- AWS-UpdateLinuxAmi
 - createImage 41
 - launchInstance 40
 - stopInstance 40
 - terminateInstance 41
 - updateOSSoftware 40

B

- benefits, Amazon EMR
 - integration 238
 - pricing 238

reliability 238

scalability 238

benefits, Elastic Beanstalk

control 54

costs 54

deployment support 54

developer friendly 54

platform support 54

C

cloning environments 70

CloudFormation

used, for automating WAF Web ACL

deployments 105, 110

CloudFront CDN-based template

URL 107

CloudFront distributions

WAF Web ACL, assigning 99

CloudFront

URL 90

CloudWatch events

automation, triggering 43, 45

CloudWatch schedules

automation, triggering 43, 45

CloudWatch

about 12

dashboard widgets 13

events 12

high-resolution custom metrics 13

reference link 43

Systems Manager (SSM) agent, configuring 29

used, for monitoring WAF 111

compliance

used, for instance patches managing 46, 50

components, AWS Greengrass

Greengrass Core (GGC) software 319

Greengrass Core SDK 319

Greengrass groups 319

Lambda functions 319

subscription 319

D

database as a service (DBaaS) 256

DDoS Response Team (DRT) 113

dead letter queue 224

Distributed Denial of Services (DDoS) 85

E

EC2 instance

about 20

commands, execution 32

EC2 Systems Manager

about 20

automation 19

inventory 19

patch management 19

run command 19

State Manager 19

EFS Performance

reference link 82

Elastic Beanstalk CLI

working with 64

Elastic Beanstalk Saved Configurations

reference link 82

Elastic Beanstalk

about 54, 58

application 55

application versions 55

cloning environments 70

concepts 55, 57

environment dashboard 67, 70

environments 55

production environment, configuring 71

terminologies 55, 57

web server environment 55

worker environment 57

Elastic Compute Cloud (EC2) 8

Elastic File System (EFS)

about 14, 75

creating 77, 79

extending, to Elastic Beanstalk 79

reference link 77

working 75

Elastic Load Balancer (ELB) 13, 186

Elastic MapReduce (EMR) 16, 236

EMR File System (EMRFS) 240, 267

environment dashboard

about 67, 70

Health 68

- Instances 68
- NOTE 68
- notifications 68
- scaling 68
- software configuration 68
- environment manifest 81
- Environment Manifest (env.yaml)
 - reference link 82
- ETL (Extract, Transform and Load) 237

F

- fan out pattern 230
- features, AWS IAM
 - access mechanisms 151
 - AWS products, integration 151
 - identity federation 151
 - multi-factor authentication 151
- FIFO queue
 - creating, with AWS CLI 227

H

- Hadoop Distributed File System (HDFS)
 - about 239
 - URL 240
- HIPAA compliance
 - URL 161
- Hive 241
- Host Manager 55

I

- IAM Management Console
 - URL 152
- IAM policies
 - action 152
 - condition 152
 - creating, with visual editor 152
 - effect 152
 - resources 152
 - testing, with IAM Policy Simulator 156
- IAM Policy Simulator
 - URL 156
 - used, for testing IAM policies 156
- Identity and Access Management Service (IAM) 171

- instance maintenance
 - simplifying, with System Manager Automation 39
- instance patches
 - managing, with compliance 46, 50
 - managing, with patch baseline 46, 50
- Internet of Things (IoT)
 - about 299
 - requisites 299
- Inventory Management
 - about 50
- inventory service
 - enabling, for Dev instance 50
- IoT suite of services
 - Amazon FreeRTOS 301
 - AWS Greengrass 301
 - AWS IoT Analytics 301
 - AWS IoT Button 301
 - AWS IoT Core 300
 - AWS IoT Device Defender 301
 - AWS IoT Device Management 300

L

- Linux instance
 - patching, with automation 41, 43
- log management
 - elasticsearch 135
 - Kibana 135
 - logstash 135
- log parsing protection 106
- long polling 224

M

- maintenance window
 - associating, with patch baseline 49
- messaging services
 - about 205
 - Amazon Kinesis 207
 - Amazon SNS 205
 - Amazon SQS 206
- MySQL DB instance
 - reference link 71

P

- patch baseline
 - maintenance window set up 47
 - used, for instance patches managing 46, 50
- pipeline definition file
 - fields 281
 - objects 281
- plan of attack 17
- production environment
 - configuring 71
- projects 202
- Proofpoint
 - URL 106
- Publisher-Subscriber 205

R

- requisites, IoT
 - application 300
 - gateways 300
 - processors 300
 - sensors 299
 - things 299
- Run Command 31

S

- sandbox 162
- server-side encryption 224
- services
 - CloudWatch 12
 - EC2 Systems Manager 14
 - Elastic Beanstalk 14
 - Elastic Compute Cloud (EC2) 8
 - Elastic File System (EFS) 14
 - Elastic Load Balancer 13
 - enhancements 8
 - FPGAs, availability 9
 - GPUs, availability 9
 - shield 14
 - Simple Storage Service (S3) 9
 - Virtual Private Cloud (VPC) 11
 - WAF 14
- Shield
 - cloudtrail 15
 - CodeBuild 15

- CodeCommit 15
- CodeDeploy 15
- cognito 15
- Config 15
- data pipeline 16
- EMR 16
- IoT and Greengrass 16
- Redshift 16
- SNS 15
- SQS 15
- short polling 224
- Simple Notification Service (SNS)
 - about 15, 204, 205, 208, 210
 - integrating, with Amazon SQS 229
 - monitoring, with Amazon CloudWatch metrics 219
 - used, for sending text messages 211, 214
 - using, as triggers 215, 218
- Simple Queue Service (SQS)
 - about 15, 204, 206, 221
 - FIFO queue, creating with AWS CLI 227
 - FIFO queues 221
 - integrating, with Amazon SNS 229
 - queue, creating 222
 - standard queue 221
- Simple Storage Service (S3)
 - about 9
 - Amazon S3 transfer acceleration 11
 - dashboard 10
 - S3 Analytics 9
 - S3 CloudWatch metrics 10
 - S3 Inventory 9
 - S3 Object Tagging 9
- SQL Workbench/J 261
- SSM Agent on Linux Instances
 - reference link 27
- State Manager
 - automation 35
 - command 35
 - policy 35
 - working with 35, 38
- stream logs
 - Systems Manager (SSM) agent, configuring 29
- System Manager Automation
 - about 39

- used, for instance maintenance simplifying 39

Systems Manager (SSM) agent

- about 20
- configuring, to CloudWatch 29
- configuring, to stream logs 29
- IAM policies, configuring 20, 23, 26
- IAM Roles, configuring 20, 23, 26
- installing 27

T

terminologies, Amazon EMR

- clusters 239
- nodes 239
- steps 239

terminologies, Amazon Redshift

- clusters 256
- databases 256
- node slices 256

terminologies, AWS Config

- config rules 140
- configuration history 140
- configuration items 140
- configuration snapshot 140

terminologies, AWS Organizations

- accounts 161
- consolidated billing 160
- features 160
- Organizational Unit (OU) 160
- organizations 160
- root 160
- Service Control Policy (SCP) 161

terminologies, CloudTrail

- events 118
- logs 118
- trails 118

terminologies, WAF

- conditions 86
- rules 87
- web ACLs 88

transaction per second (TPS) 221

triggers 184

V

Virtual Private Cloud (VPC)

- about 11
- DNS resolution, for VPC peering 12
- endpoints, for DynamoDB 12
- IPv6, support 11

W

Web ACL deployments

- automating, with CloudFormation 105, 110
- components 105

Web ACL

- assigning, to CloudFront distributions 99

Web Application Firewall (WAF)

- about 14, 84, 85, 89
- advanced traffic filtering 85
- conditions, creating 92
- cost effective security solution 86
- cross-site scripting conditions, working with 100
- enhanced protection 85
- management 86
- monitoring, with CloudWatch 111
- rules, creating 94
- SQL injection, working with 100

terminologies 86

URL 90, 104

WAF Web ACL deployments, automating with CloudFormation 105

WAF Web ACL, assigning to CloudFront distributions 99

Web ACL deployments, automating with CloudFormation 110

web ACL, creating 90

web server environment 55

WordPress site

- Dev environment, creating 58, 62, 63

worker environment 55

Y

Yet Another Resource Negotiator (YARN) 241

Z

Zabbix

- reference link 32