Machine Learning with AWS

Explore the power of cloud services for your machine learning and artificial intelligence projects



Jeffrey Jackovich, Ruze Richards

Packt
www.packt.com

Machine Learning with AWS

Explore the power of cloud services for your machine learning and artificial intelligence projects

Jeffrey Jackovich, Ruze Richards

Packt

Machine Learning with AWS

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 authors, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

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

Authors: Jeffrey Jackovich, Ruze Richards

Managing Editor: Rina Yadav

Acquisitions Editor: Aditya Date

Production Editor: Samita Warang

Editorial Board: David Barnes, Ewan Buckingham, Simon Cox, Manasa Kumar, Alex Mazonowicz, Douglas Paterson, Dominic Pereira, Shiny Poojary, Saman Siddiqui, Erol Staveley, Ankita Thakur, and Mohita Vyas

First Published: November 2018

Production Reference: 1021118

ISBN: 978-1-78980-619-9

Table of Contents

Preface i

Introduction to Amazon Web Services 1

Introduction 2
What is AWS? 2
What is Machine Learning? 2
What is Artificial Intelligence? 2
What is Amazon S3? 3
Why use S3? 4
The Basics of Working on AWS with S3 4
AWS Free-Tier Account 4
Importing and Exporting Data into S3 5
How S3 Differs from a Filesystem 5
Core S3 Concepts 5
S3 Operations 7
Data Replication 8
REST Interface 8
Exercise 1: Using the AWS Management Console to Create an S3 Bucket 9
Exercise 2: Importing and Exporting the File with your S3 Bucket 12
AWS Command-Line Interface (CLI) 17
Exercise 3: Configuring the Command-Line Interface 17
Command Line-Interface (CLI) Usage 22
Recursion and Parameters 22
Activity 1: Importing and Exporting the Data into S3 with the CLI 23
Using the AWS Console to Identify Machine Learning Services 24

Exercise 4: Navigating the AWS Management Console 24
Activity 2: Testing the Amazon Comprehend's API Features 26
Summary 27

SummarizingText Documents Using NLP 29

Introduction 30
What is Natural Language Processing? 30
Using Amazon Comprehend to Inspect Text and Determine the Primary Language 31
Exercise 5: Detecting the Dominant Language Using the Command-Line Interface in a text document 32
Exercise 6: Detecting the Dominant Language in Multiple Documents by Using the Command-Line Interface (CLI) 35
Extracting Information in a Set of Documents 36
Detecting Named Entities – AWS SDK for Python (boto3) 36
DetectEntites – Input and Output 38
Exercise 7: Determining the Named Entities in a Document 38
DetectEntities in a Set of Documents (Text Files) 40
Detecting Key Phrases 41
Exercise 8: Determining the Key Phrase Detection. 41
Detecting Sentiments 42
Exercise 9: Detecting Sentiment Analysis 42
Setting up a Lambda function and Analyzing Imported Text Using Comprehend 44
What is AWS Lambda? 44
What does AWS Lambda do? 45
Lambda Function Anatomy 45
Exercise 10: Setting up a Lambda function for S3 46
Exercise 11: Configuring the Trigger for an S3 Bucket 52
Exercise 12: Assigning Policies to S3_trigger to Access Comprehend 56

Activity 3: Integrating Lambda with Amazon Comprehend to Perform Text Analysis 58
Summary 60

Perform Topic Modeling and Theme Extraction 63

Introduction 64
Extracting and Analyzing Common Themes 64
Topic Modeling with Latent Dirichlet Allocation (LDA) 64
Basic LDA example 65
Why Use LDA? 65
Amazon Comprehend–Topic Modeling Guidelines 66
Exercise 13: Topic Modeling of a Known Topic Structure 68
Exercise 14: Performing Known Structure Analysis 84
Activity 4: Perform Topic Modeling on a Set of Documents with Unknown Topics 87
Summary 88

Creating a Chatbot with Natural Language 91

Introduction 92
What is a Chatbot? 92
The Business Case for Chatbots 92
What is Natural Language Understanding? 93
Core Concepts in a Nutshell 93
Setting Up with Amazon Lex 96
Introduction 96
Exercise 15: Creating a Sample Chatbot to Order Flowers 97
Creating a Custom Chatbot 105
A Bot Recognizing an Intent and Filling a Slot 107
Exercise 16: Creating a Bot that will Recognize an Intent and Fill a Slot 108

Natural Language Understanding Engine 118
Lambda Function – Implementation of Business Logic 120
Exercise 17: Creating a Lambda Function to Handle Chatbot Fulfillment 121
Implementing the Lambda Function 123
Input Parameter Structure 124
Implementing the High-Level Handler Function 125
Implementing the Function to Retrieve the Market Quote 125
Returning the Information to the Calling App (The Chatbot) 126
Connecting to the Chatbot 127
Activity 5: Creating a Custom Bot and Configuring the Bot 129
Summary 129

Using Speech with the Chatbot 131

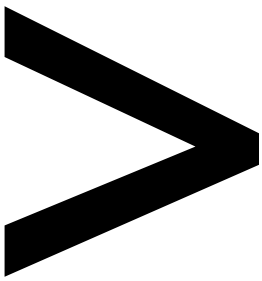
Introduction 132
Amazon Connect Basics 132
Free Tier Information 132
Interacting with the Chatbot 133
Talking to Your Chatbot through a Call Center using Amazon Connect 134
Exercise 18: Creating a Personal Call Center 135
Exercise 19: Obtaining a Free Phone Number for your Call Center 141
Using Amazon Lex Chatbots with Amazon Connect 143
Understanding Contact Flows 144
Contact Flow Templates 144
Exercise 20: Connect the Call Center to Your Lex Chatbot 145
Activity 1: Creating a Custom Bot and Connecting the Bot with Amazon Connect 154
Summary 155

Analyzing Images with Computer Vision 157

Introduction 158
Amazon Rekognition Basics 158
Free Tier Information on Amazon Rekognition 159
Rekognition and Deep Learning 159
Detect Objects and Scenes in Images 160
Exercise 21: Detecting Objects and Scenes using your own images 163
Image Moderation 166
Exercise 22: Detecting objectionable content in images 169
Facial Analysis 171
Exercise 23: Analyzing Faces in your Own Images 172
Celebrity Recognition 177
Exercise 24: Recognizing Celebrities in your Own Images 179
Face Comparison 182
Activity 1: Creating and Analyzing Different Faces in Rekognition 184
Text in Images 185
Exercise 25: Extracting Text from your Own Images 186
Summary 189

Appendix A 191

Index 235



Preface

About

This section briefly introduces the author, the coverage of this book, the technical skills you'll need to get started, and the hardware and software requirements required to complete all of the included activities and exercises.

About the Book

In this book, you will learn about the various artificial intelligence and machine learning services available on AWS. Through practical hands-on exercises, you will learn how to use these services to generate impressive results. By the end of this book, you will have a basic understanding of how to use a wide range of AWS services in your own projects.

About the Authors

Jeffrey Jackovich, is the author of this book, and a curious data scientist with a background in health-tech and mergers and acquisitions (M&A). He has extensive business-oriented healthcare knowledge, but enjoys analyzing all types of data with R and Python. He loves the challenges involved in the data science process, and his ingenious demeanor was tempered while serving as a Peace Corps volunteer in Morocco. He is completing a Masters of Science in Computer Information Systems, with a Data Analytics concentration, from Boston University.

Ruze Richards, is the author of this book, and a data scientist and cloud architect who has spent most of his career building high-performance analytics systems for enterprises and startups. He is especially passionate about AI and machine learning, having started life as a physicist who got excited about neural nets, then going on to work at AT&T Bell Labs in order to further pursue this area of interest. With the new wave of excitement along with the actual computing power being available on the cloud for anybody to actually get amazing results with machine learning, he is thrilled to be able to spread the knowledge and help people achieve their goals.

Objectives

- Get up and running with machine learning on the AWS platform
- Analyze unstructured text using AI and Amazon Comprehend
- Create a chatbot and interact with it using speech and text input
- Retrieve external data via your chatbot
- Develop a natural language interface
- Apply AI to images and videos with Amazon Rekognition

Audience

This book is ideal for data scientists, programmers, and machine-learning enthusiasts who want to learn about the artificial intelligence and machine learning capabilities of the Amazon Web Services.

Approach

This book takes a hands-on approach to teach you machine learning with AWS. It contains multiple activities that use real-life business scenarios for you to practice and apply your new skills in a highly relevant context.

Minimum Hardware Requirements

For an optimal student experience, we recommend the following hardware configuration:

- Processor: Intel Core i5 or equivalent
- Memory: 4GB RAM
- Storage: 35GB available space

Software Requirements

You'll also need the following software installed in advance:

1. OS: Windows 7 SP1 64-bit, Windows 8.1 64-bit or Windows 10 64-bit
2. Browser: Google Chrome, Latest Version
3. An AWS free tier account

Conventions

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: " The command form is, "**`s3://myBucketName/myKey.`**"

A block of code is set as follows:

```
aws comprehend detect-dominant-language ^
--region us-east-1 ^
--text "Machine Learning is fascinating."
```

New terms and important words are shown in bold. Words that you see on the screen, for example, in menus or dialog boxes, appear in the text like this: "Data stored in S3 is managed as objects using an **Application Programming** Interface (**API**) accessible via the internet (**HTTPS**)."

Installation and Setup

Before you start this book, you will need an AWS account. You will also need to set up the AWS command-line interface (AWXSCLI), the steps for which can be found below. You will also need Python 3.6, pip and an AWS Rekognition Account throughout the book.

AWS account

For an AWS free tier account, you will need a personal email address, and credit or debit card, and a cell phone that can receive text message so you can verify your account. To create a new account, follow this link https://aws.amazon.com/free/.

AWSCLI Setup

Install AWS CLI setup from the link https://s3.amazonaws.com/aws-cli/AWSCLISetup.exe. To download the AWS CLI setup file (*includes 32-bit and 64-bit MSI installers and will automatically install the correct version). To verify install was successful open a command prompt and type **`aws --version`**.

Installing Python

Install Python 3.6 following the instructions at: https://realpython.com/installing-python/.

Installing pip

1. To install pip, go to command prompt and type **`pip install awscli --upgrade --user`**. Verify the successful install with command "**`aws - -version`**"
2. After installing **`pip`**, add the AWS executable to your OS's PATH environment variable. With an MSI installation, this should occur automatically, but you may need to set it manually if the "**`aws - -version`**" command is not working.
3. To modify your **`PATH`** variable (Windows), type **`environment variables`**, and select **`Edit the system environment variables for your account`**, select the path, and add the path to the variable value field, separated by semicolons.

Installing Virtual Environment

Install the Anaconda version depending on your operating system from the following link https://www.anaconda.com/download/. Anaconda helps install what you need without conflicting packages.

1. To check the Anaconda Distribution is up to date, type **`conda update conda`**.
2. To create a virtual environment, type **`conda create -n yourenvname python=3.6 anaconda`** and press **`y`** to continue, this will install the Python version and all associated anaconda packaged libraries at **`path_to_you_anaconda_location/anaconda/envs/yourenvname`**.
3. To activate the account on macOS and Linux, type **`source activate yourenvname`** and for Windows type **`activate yourenvname`**.
4. To install the additional Python packages to a virtual environment, type **`conda install -n yourenvname [package]`**.
5. To deactivate the virtual environment type **`deactivate`**.

Configuration and Credential files

To locate the config file, see the operating specific commands below. For more information see: https://docs.aws.amazon.com/cli/latest/userguide/cli-config-files.html.

Amazon Rekognition Account

You will need to create a new Amazon Rekognition free tier account where customers can analyze up to 5,000 images free each month for the first 12 months. To create the free account, follow the link https://aws.amazon.com/rekognition/

Installing the Code Bundle

Additional Resources

The code bundle for this book is also hosted on GitHub at: https://github.com/TrainingByPackt/Machine-Learning-with-AWS.

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

1

Introduction to Amazon Web Services

Learning Objectives

By the end of this chapter, you will able to:

- Describe the basics of working on AWS using Amazon S3
- Import data from and export data to Amazon S3 using the AWS Management Console and Command-Line Interface (CLI)
- Use the AWS Management Console
- Identify Machine Learning services

This chapter describes the working of S3 using AWS management console and recognize machine learning services.

Introduction

This chapter will introduce you to the **Amazon Web Service (AWS)** interface, and will teach you how to store and retrieve data with **Amazon Simple Storage (S3)**.

Next, you will apply your S3 knowledge by importing and exporting text data via the management console and the **Command-Line Interface (CLI)**.

Lastly, you will learn how to locate and test **Artificial Intelligence (AI)** and **Machine Learning (ML)** services.

What is AWS?

AWS is a secure cloud platform that delivers on-demand computing power, database storage, applications, and other IT resources, through a cloud services platform via the internet with a pay-as-you-go pricing model. As of 2018, AWS dominates the cloud infrastructure services worldwide market with ~30% of the market share, compared to the trailing Microsoft (~15%) and Google (~5%), as per Canalys (https://www.canalys.com/static/press_release/2018/270418-cloud-infrastructure-market-grows-47-q1-2018-despite-underuse.pdf).

What is Machine Learning?

Machine Learning is a subset of Artificial Intelligence in the field of computer science that often uses statistical techniques to provide computers with the ability to learn with data without being programmed. Machine Learning explores the building and construction of algorithms that can learn from and make predictions about, data. The algorithms transcend static instructions and make data-driven predictions and decisions with a model from sample inputs.

Machine Learning is essential to learn in today's world, since it is an integral part of all industries' competitive and operational data strategies. More specifically, ML allows insights from **Natural Language Processing (NLP)** to power chatbots; fraud detection ML insights are used in the financial industry, and ML applications allow for efficient online recommendation engines, such as friend suggestions on Facebook, Netflix displaying movies, you probably like, and more items to consider on Amazon.

What is Artificial Intelligence?

Artificial Intelligence is intelligence that's demonstrated by machines; more specifically, any device that perceives its environment and takes actions that increases its chance of successfully achieving its goals. Contemporary examples are understanding human speech, competing at the highest levels of strategic games (such as Chess and Go), and autonomous cars.

Artificial Intelligence is important because it adds intelligence to existing products. Products that are currently used will be further improved with AI capabilities, for example, Siri was added to the new generation of Apple products. Conversational chatbots can be combined with large amounts of data to improve technologies at home and in the office.

Overall, this chapter will provide you with the foundational knowledge of AWS to build sophisticated AI and ML applications in your projects. This will help you with the tools to identify free-tier AWS services. Thus, you will be able to use them exclusively, or integrate them to analyze data, build a conversational chatbot, store and process an incredible amount of data, and bring your ideas to fruition.

This chapter will introduce you to the AWS interface and help you learn how to store and retrieve data with **Amazon Simple Storage Service (S3)**. You will apply your S3 knowledge by importing and exporting text data via the AWS Management Console and the CLI. You will also learn how to locate and test AI and ML services.

What is Amazon S3?

S3 is an online cloud object storage and retrieval service. Amazon S3 is a cloud object storage. Instead of data being associated with a server, S3 storage is server independent and can be accessed over the internet Data stored in S3 is managed as objects using an **Application Programming Interface** (**API**) that is accessible via the internet (HTTPS).

The benefits of using S3 are as follows:

- Amazon S3 runs on the largest global cloud infrastructure, to deliver 99.99% durability.
- It provides the widest range of options to transfer data.
- It allows you to run Big Data analytics without moving data into a separate analytics system.
- It supports security standards and compliance certificates.
- It offers a flexible set of storage management and administration capabilities.

> **Note**
>
> For more information refer*:* https://aws.amazon.com/s3/.

Why use S3?

The **S3** is a place to store and retrieve your files. It is recommended for storing static content such as text files, images, audio files, video files, and so on. For example, S3 can be used as a static web server if the website consists exclusively of HTML and images. The website can be connected to an FTP client to serve the static files. In addition, S3 can also be used to store user generated image and text files.

However, the two most important applications of it are as follows:

- To store static data from web pages or mobile apps
- To implement Big Data analytics

It can easily be used in conjunction with additional AWS Machine Learning and infrastructure services. For example, text documents imported to Amazon S3 can be summarized by code running in an AWS Lambda function that is analyzed using AWS Comprehend. We will cover both in *Chapter 2, Summarizing Text Document using NLP* and *Chapter 3, Perform Topic Modeling and Theme Extraction.*

The Basics of Working on AWS with S3

The first step to accessing S3 is to create an AWS Free-tier account, which provides access to the AWS Management Console. The AWS Management Console is a web application that provides one method to access all of AWS's powerful storage and ML/AI services.

The second step is to understand the access level. AWS defines **Identity and Access Management (IAM)**. The same email/password is used for accessing the IAM

AWS Free-Tier Account

AWS provides a free-tier (*within their individual free usage stipulations*) account, and one of the included storage services is *Amazon Simple Storage* (S3). Thus, you can maximize cost savings and reduce errors before making a large investment by testing services to optimize your ML and AI workflows.

Importing and Exporting Data into S3

AWS Import and Export is a service that you can use to transfer large amounts of data from physical storage devices into AWS. You mail your portable storage devices to AWS, and AWS Import/Export transfers data directly from your storage devices using Amazon's high-speed internal network. Your data load typically begins the next business day after your storage device arrives at AWS. After the data export or import completes, services return your storage device. For large datasets, AWS data transfer can be significantly faster than internet transfer and more cost-effective than upgrading your connectivity.

How S3 Differs from a Filesystem

S3 is used to store almost any type of file, thus, it can get confused with similarities to a traditional filesystem. However, S3 differs in a few ways from a traditional filesystem. Overall, the folders in a traditional file system are **Buckets** in S3; a file in a traditional filesystem is an **object** in S3. S3 uses objects, since you can store any data type (that is, more than files) in Buckets.

Another difference is how objects can be accessed. Objects stored in Buckets can be accessed from a web service endpoint (such as a web browser, for example, Chrome, Firefox, and so on), so each object requires a globally unique name. The name restrictions for objects are similar to the restrictions in selecting a URL when creating a new website. Obviously, you need to select a unique URL, according to the same logic that your house has a unique address.

For example, if you created a Bucket (with public permission settings) named **`myBucket`** and then uploaded a text file named **`pos_sentiment__leaves_of_grass.txt`** to the Bucket, the object would be accessible from a web browser via the corresponding subdomain.

Core S3 Concepts

The S3 hierarchy includes the following concepts:

Type of data storable: S3 is recommended for storing static content, such as text files, images, audio, video, and so on.

Objects: Objects are the most basic entities that are stored in S3. Every object contains data, metadata, and a key. Metadata is data about the data and provides basic information about the data stored in an object. Metadata is stored in a set of name-value pairs, used to describe the information associated with the object.

Keys: A key is the name assigned to an object that uniquely identifies an object inside a Bucket. All objects in a bucket have one key associated with them.

Bucket: Just like a folder, a Bucket is the container where you store objects. Buckets are created at root level, and do not have a filesystem hierarchy. More specifically, you can have multiple Buckets, but you cannot have sub-Buckets within a Bucket. Buckets are the containers for objects, and you can control (create, delete, and list objects in the Bucket) access to it, view access logs for it, and select the geographical region where Amazon S3 will store the Bucket.

Region: Region refers to the geographical region where Amazon S3 stores a Bucket, based on the user's preference. The region can be selected when creating a Bucket. The location should be based on where the data will be accessed the most. Overall, specific region selection has the biggest impact if S3 is used to store files for a website that's exclusively accessed in a specific geographic region. The object storage in a Bucket with different forms is as follows:

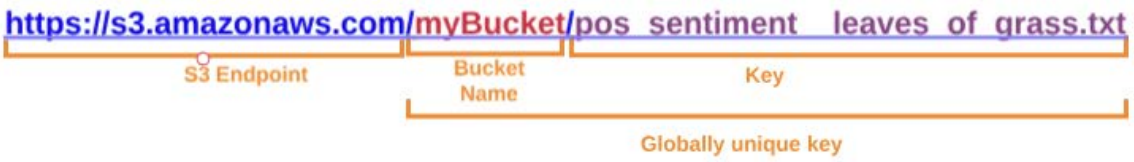


Figure 1.1: Object storage

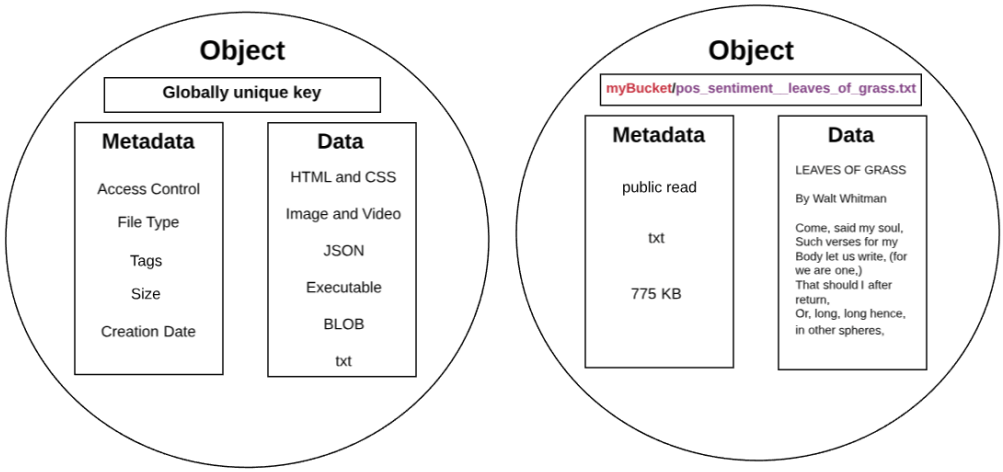


Figure 1.2: Object storage using a unique key and myBucket

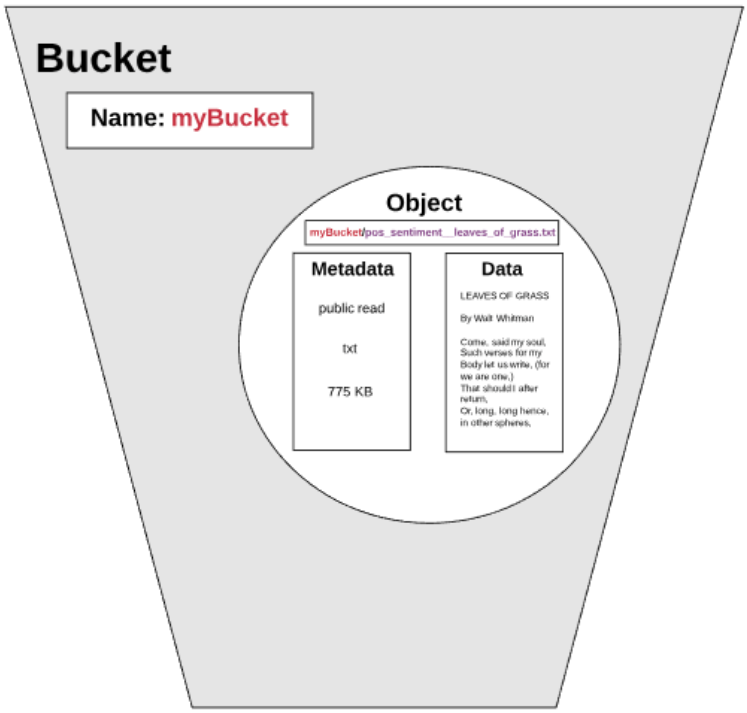


Figure 1.3: Object stored in myBucket

S3 Operations

The S3 Application Program Interface (API) is quite simple, and it includes the following operations for the respective entity:

- **Bucket**: Create, Delete, and List keys in a Bucket
- **Object**: Write, Read, and Delete

Data Replication

Amazon replicates data across the region in multiple servers located in Amazon's data centers. Data replication benefits include high availability and durability. More specifically, when you create a new object in S3, the data is saved in S3; however, the change needs to be replicated across the Amazon S3 regions. Overall, replication may take some time, and you might notice delays resulting from various replication mechanisms. Please consider the following when performing the indicated operations.

After deleting an object, replication can cause a lag time that allows the deleted data to display until the deletion is fully replicated. Creating an object and immediately trying to display it in the object list might be delayed as a result of a replication delay.

REST Interface

S3's native interface is a **Representational State Transfer (REST)** API. It is recommended to always use HTTPS requests to perform any S3 operations. The two higher-level interfaces that we will use to interact with S3 are the AWS Management Console and the AWS **Command-Line Interface** (**CLI**). Accessing objects with the API is quite simple, and includes the following operations for the respective entity:

- **Bucket**: Create, Delete, or List keys in a Bucket
- **Object**: Write, Read, or Delete

Exercise 1: Using the AWS Management Console to Create an S3 Bucket

In this exercise, we will import a text file into our S3 Bucket. To import a file, you need to have access to the Amazon S3 console:

1. Press the *Ctrl* key while clicking https://console.aws.amazon.com/console/home to open the AWS Management Console in a new browser tab.
2. Click inside the search bar located under AWS services:

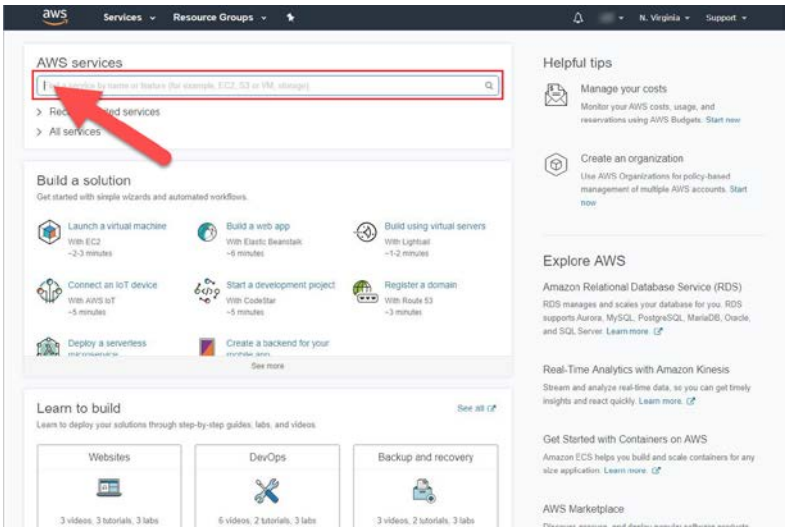


Figure 1.4: Searching AWS services

3. Type **S3** into the search bar, and an auto-populated list will display. Then, click on the **S3 Scalable Storage in the Cloud** drop-down option:

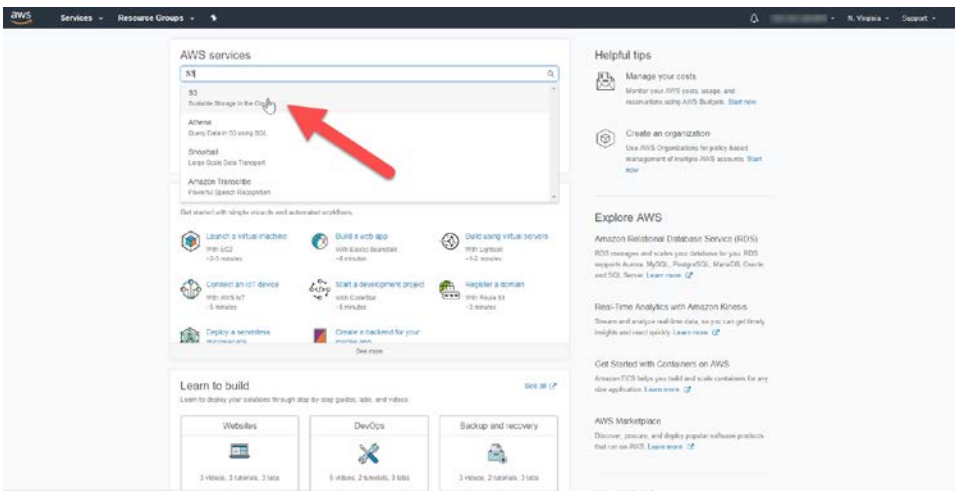


Figure 1.5: Selecting the S3 service

4. Now, we need to create an S3 Bucket. In the S3 dashboard, click the **Create Bucket** button.
5. If this is the first time that you are that you are creating a bucket, your screen will look as follows:

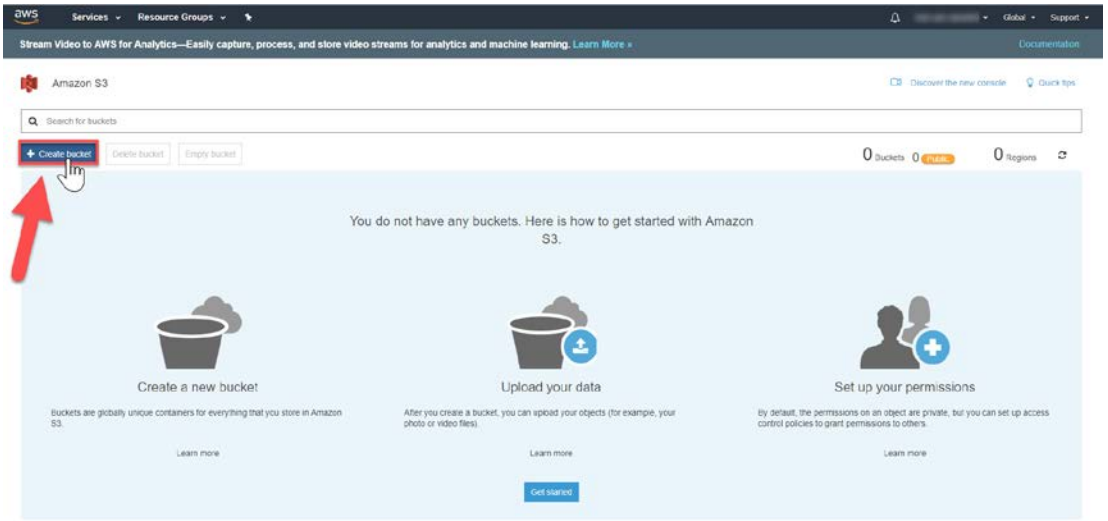


Figure 1.6: Creating the Bucket

> **Note**
>
> If you have already created S3 Buckets, your dashboard will list all of the Buckets you have created.

6. **Enter a unique Bucket name**: Bucket names must be unique across all existing Bucket names in Amazon S3. That If you encounter a naming issue, please refer to https://docs.aws.amazon.com/AmazonS3/latest/dev/BucketRestrictions.html.
7. **Region**: If a default region is auto-populated, then keep the default location. If it is not auto-populated, select a region near your current location.

8. Click the **Next** button to continue for the creation of bucket:

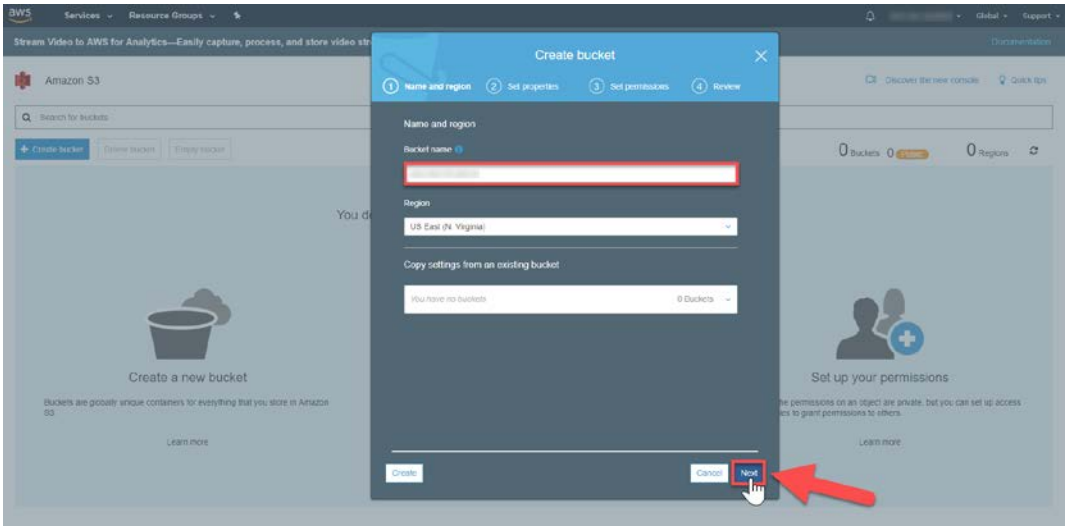


Figure 1.7: The Create Bucket window

9. An S3 Bucket provides the property options Versioning, Server Access Logging, Tags, Object-Level Logging, and Default Encryption. However, we will not enable them.

10. Your Bucket will be displayed in the bucket list, like so:

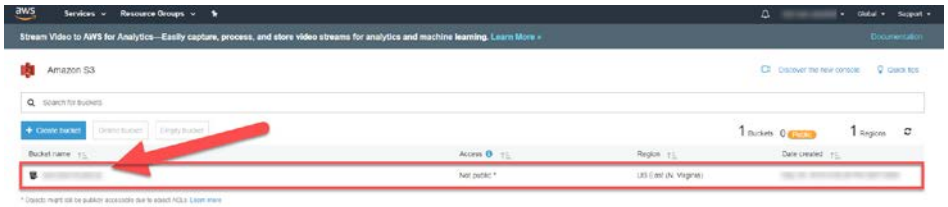


Figure 1.8: The Bucket has been created

Exercise 2: Importing and Exporting the File with your S3 Bucket

In this exercise, we will import and export the file with the S3 Bucket. The following are the steps for completion:

Importing a File:

1. You will import a file to your Amazon S3 Bucket.
2. Click the Bucket's name to navigate to the Bucket:

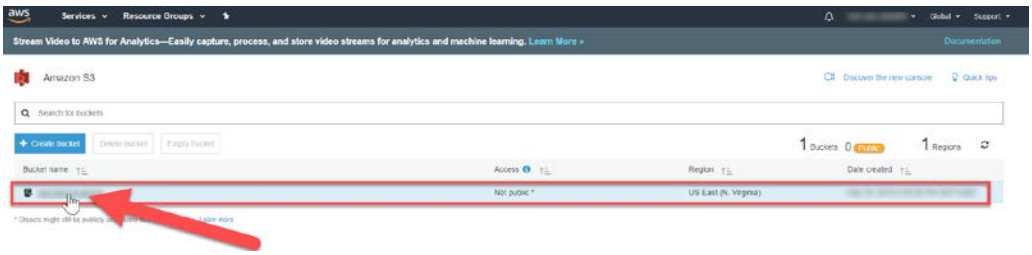


Figure 1.9: Navigate to Bucket

3. You are in the Bucket's home page. select **Upload**:

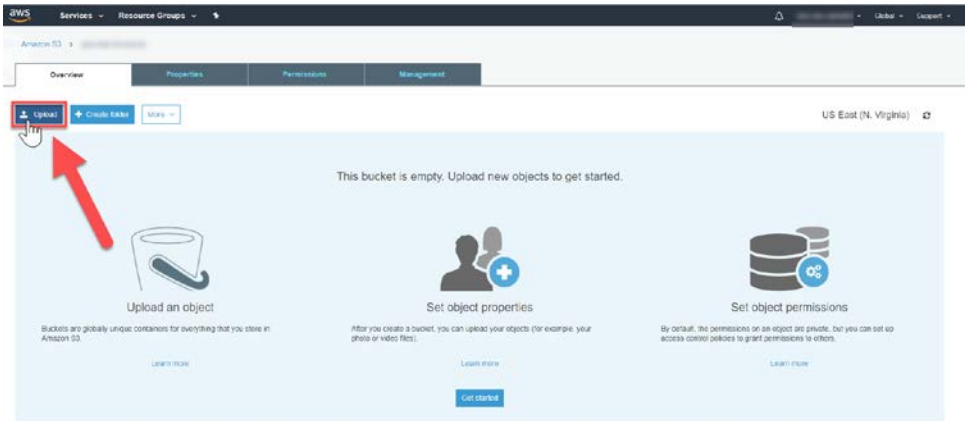


Figure 1.10: Uploading the file into the Bucket

4. To select a file to upload, click **Add files**. Navigate to the **`pos_sentiment__leaves_of_grass.txt`** location and select the sample file that you want to store:

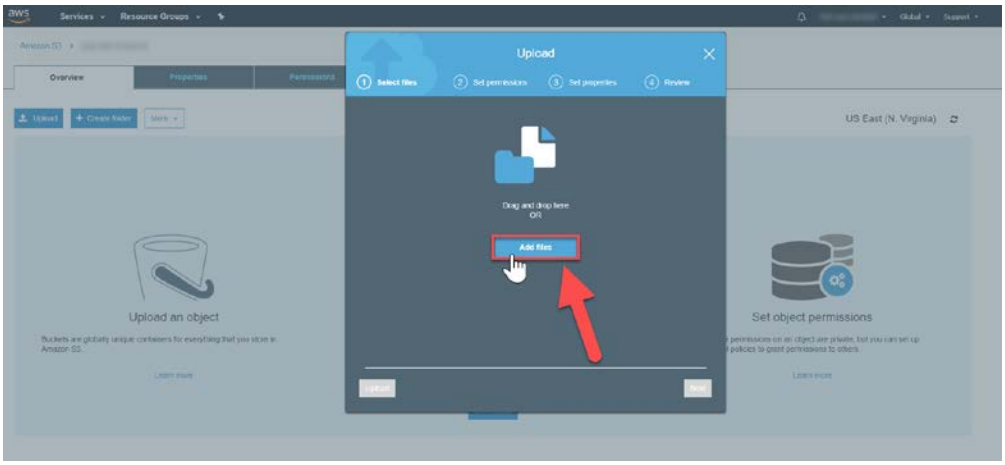


Figure 1.11: Adding a new file to the Bucket

5. After selecting a file to upload, select **Next**:

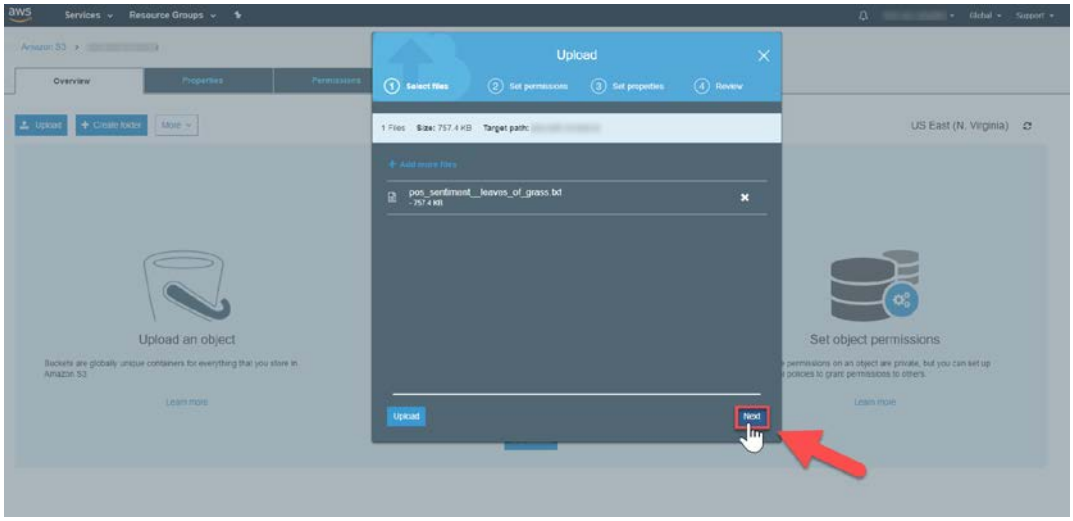


Figure 1.12: Select the file to upload to the Bucket

6. Click on the **Next** button and leave the default options selected:

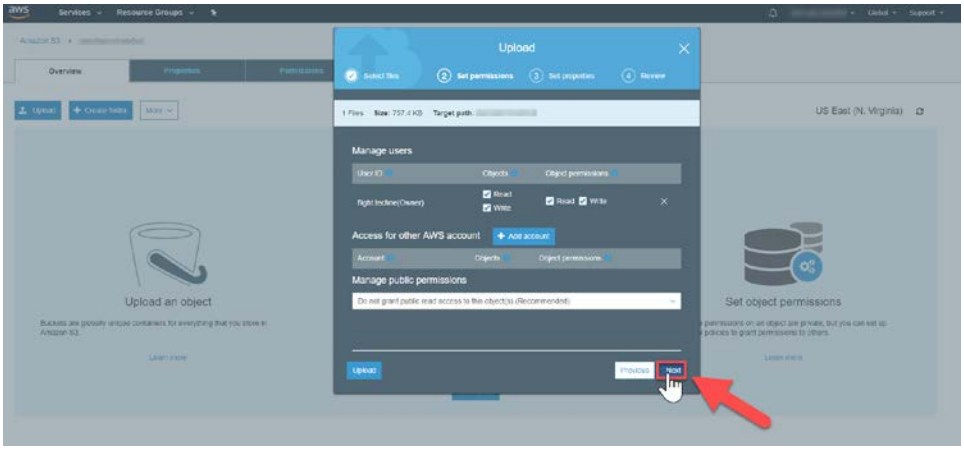


Figure 1.13: Default Options page while uploading the file

7. You have the ability to set property settings for your object, such as **`storage class`**, **`encryption`**, and **`metadata`**. However, leave the default values as-is, and then click on the **Next** button:

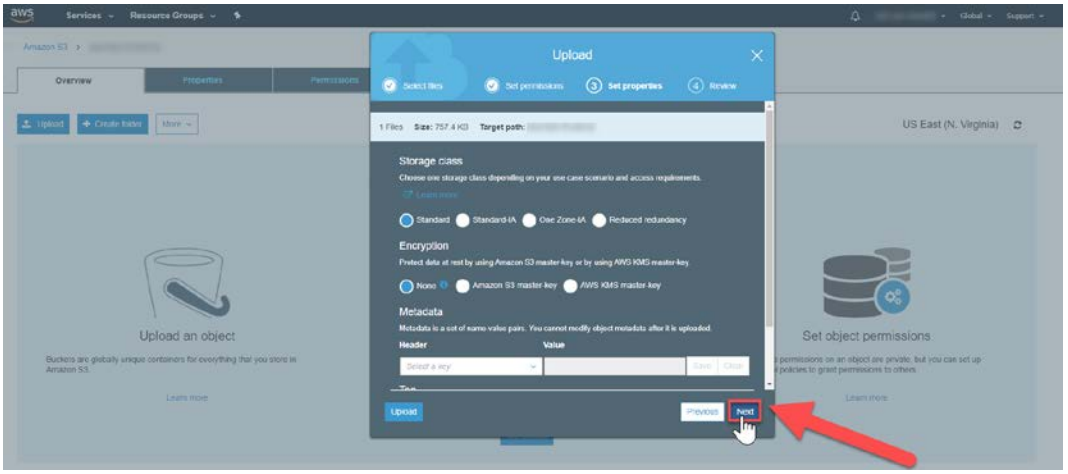


Figure 1.14: Setting properties

8. Click on the **Upload** button to upload the files:

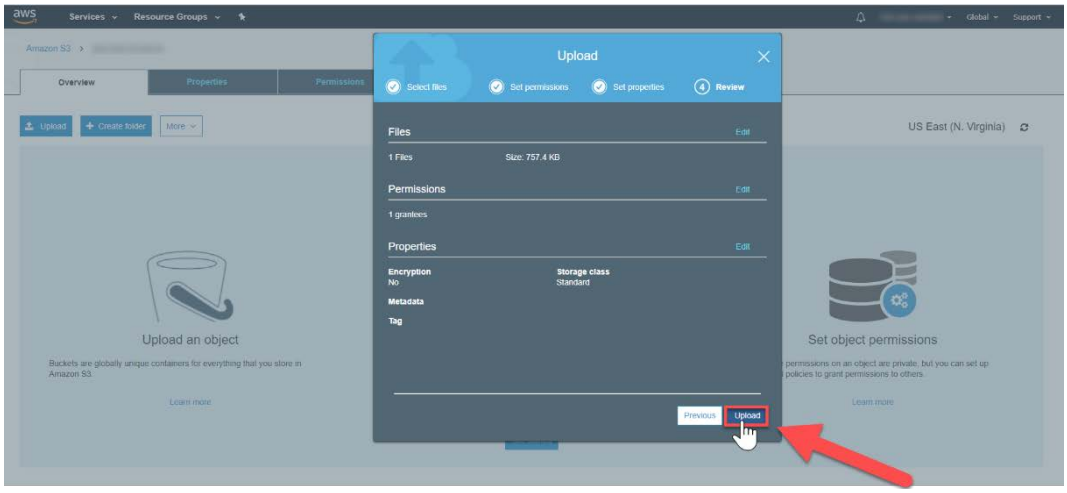


Figure 1.15: Uploading the files

9. You will be directed to your object in your bucket's home screen:

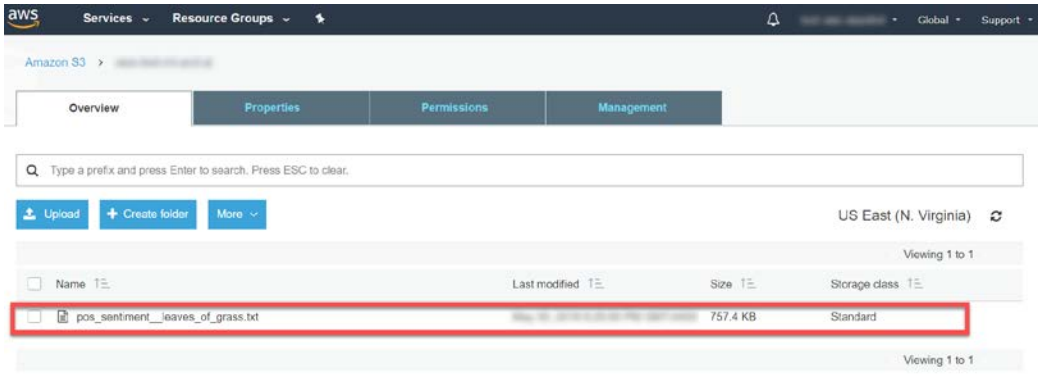


Figure 1.16: Files uploaded to the Bucket

Export File:

1. Select the checkbox next to the file to export (*Red Marker #1 – see the following screenshot*). This populates the file's information display screen. Click on **Download** (*Red Marker #2 – see the following screenshot*) to retrieve the text file:

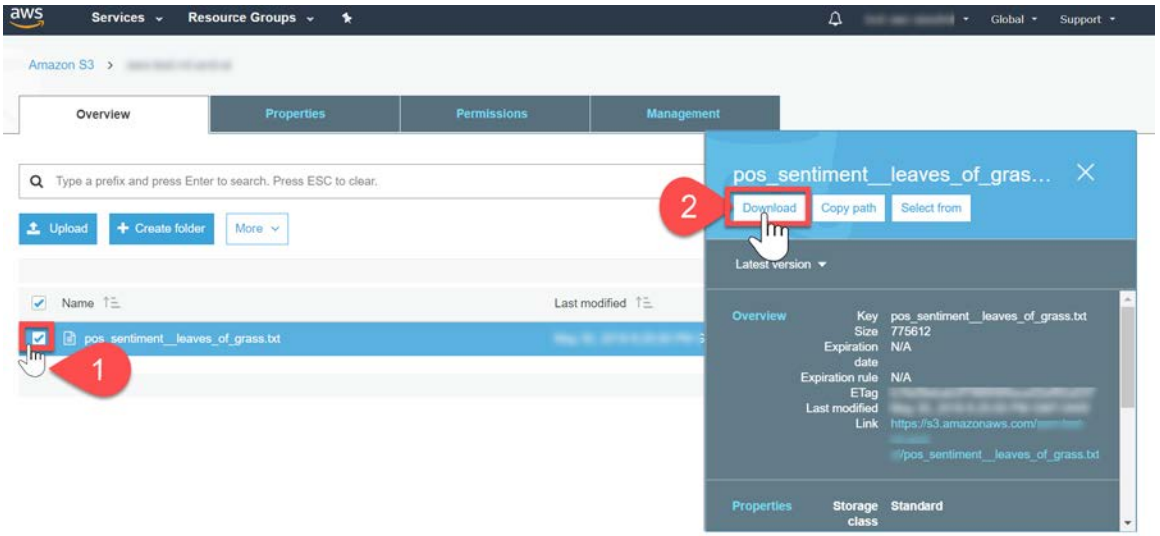


Figure 1.17: Exporting the file

2. The file will download, as shown in the lower left-hand corner of the screen:

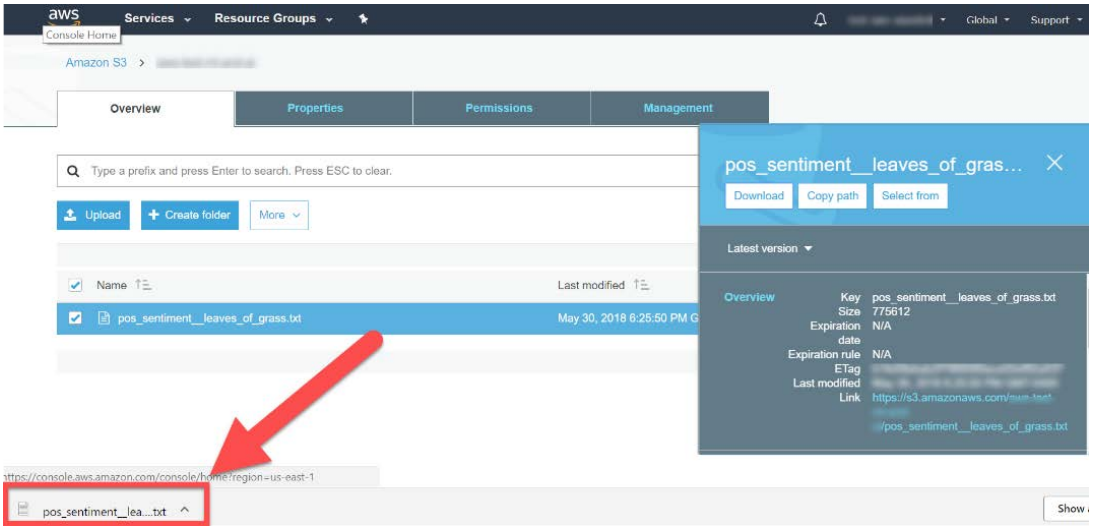


Figure 1.18: Downloading the file to export

AWS Command-Line Interface (CLI)

The CLI is an open source tool built on the AWS SDK for Python (Boto) to perform setups, determine if calls work as intended, verify status information, and so on. The CLI provides another access tool for all AWS services, including S3. Unlike the Management Console, the CLI can be automated via scripts.

To authenticate your AWS account to the CLI, you must create a configuration file to obtain your public key and secret key. Next, you will install, and then configure, the AWS CLI.

Exercise 3: Configuring the Command-Line Interface

In this exercise, we will configure the CLI with our respective AWS Access Key ID and AWS Secret Access Key. The following are the steps for completion:

1. Go to.: https://console.aws.amazon.com/console/home and then, click on **Users**:

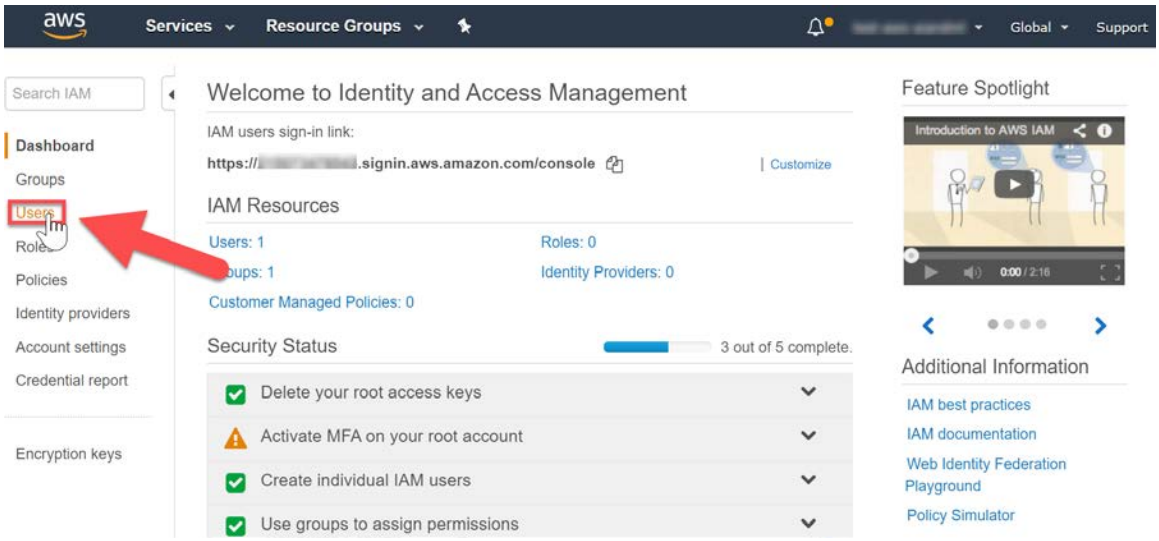


Figure 1.19: The Amazon Console home page with the Users option highlighted

2. In the upper-right corner of the signed-in AWS Management Console, click on **My Security Credentials**:

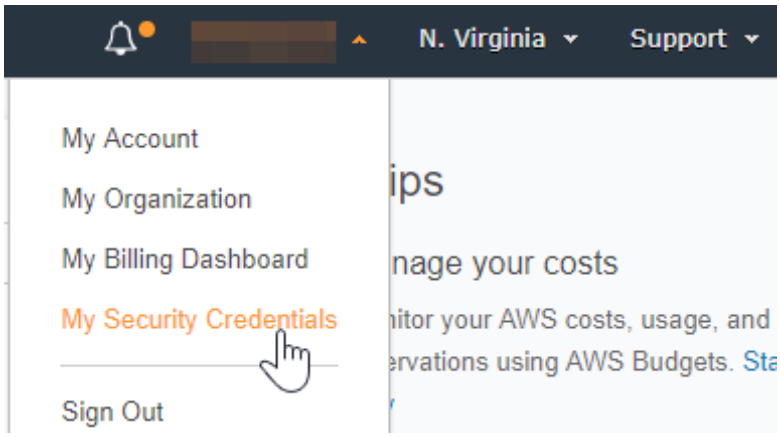


Figure 1.20: Selecting My Security Credentials

3. Next, click on **Continue to Security Credentials**:

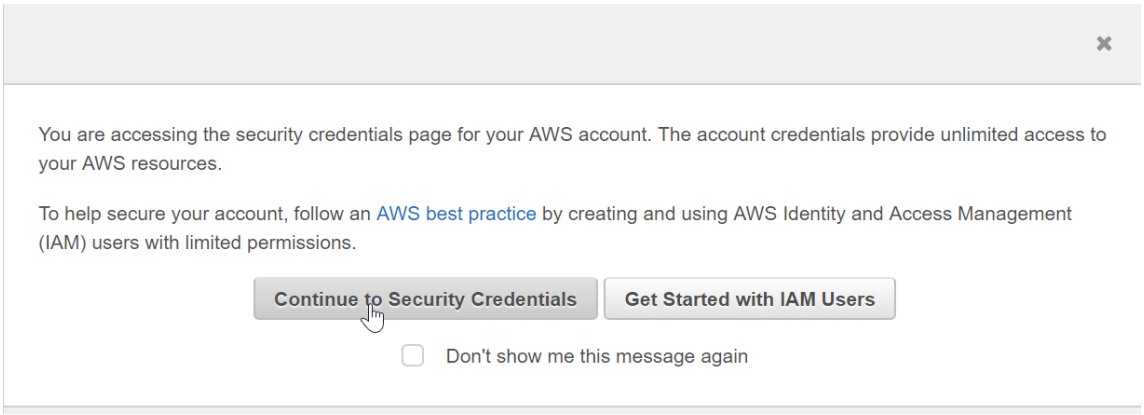


Figure 1.21: Security Credentials

4. Click on the **Access keys (access key ID and secret access key)** option:

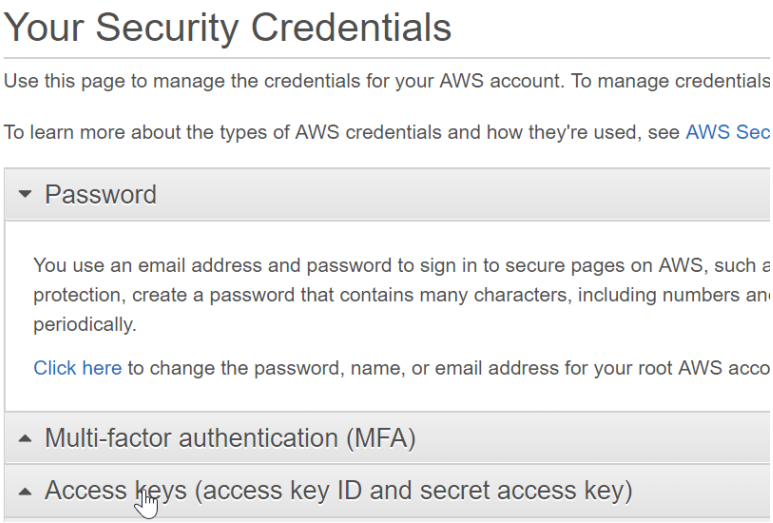


Figure 1.22: Access key generation

5. Then, click on **Create New Access Key:**



Figure 1.23: Creating a new access key

6. Click on **Download Key File** to download the key file:

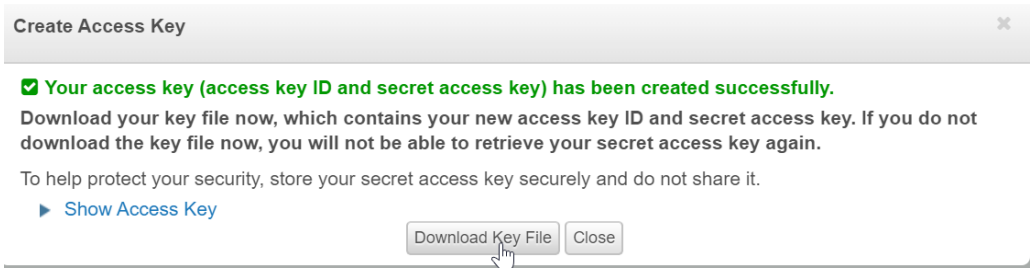


Figure 1.24: Downloading the key file

7. The **`rootkey.csv`** that contains the keys will be downloaded. Click it to view the the details:

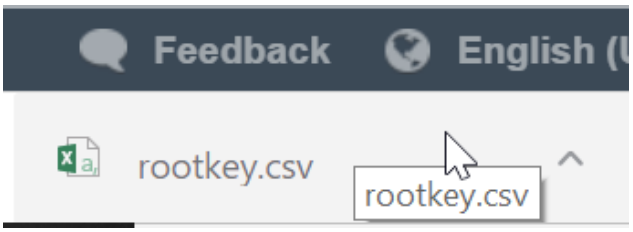


Figure 1.25: The downloaded key file

8. Store the keys in a safe location. Protect your AWS account, and never share, email, or store keys in a non-secure location. An AWS representative will never request your keys, so be vigilant when it comes to potential phishing scams.
9. Open the Command Prompt and type **`aws configure`**:
10. You will be prompted for four input variables, one by one type your respective information, then press *Enter* after each input:

    `AWS Access Key ID`

    `AWS Secret Access Key`

    `Default region`

    `Default output format (json)`

11. The name is obtained in your console (**N. Virginia** is displayed here, but yours is determined by your unique location):

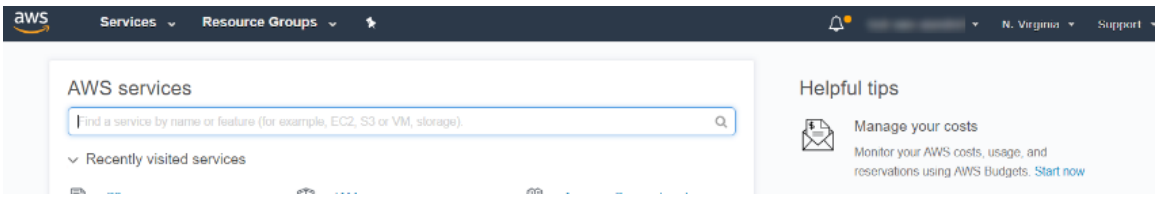


Figure 1.26: Location search

12. The code is obtained from the following **Available Regions** list:

Code	Name
us-east-1	US East (N. Virginia)
us-east-2	US East (Ohio)
us-west-1	US West (N. California)
us-west-2	US West (Oregon)
ca-central-1	Canada (Central)
eu-central-1	EU (Frankfurt)
eu-west-1	EU (Ireland)
eu-west-2	EU (London)
eu-west-3	EU (Paris)
ap-northeast-1	Asia Pacific (Tokyo)
ap-northeast-2	Asia Pacific (Seoul)
ap-northeast-3	Asia Pacific (Osaka-Local)
ap-southeast-1	Asia Pacific (Singapore)
ap-southeast-2	Asia Pacific (Sydney)
ap-south-1	Asia Pacific (Mumbai)
sa-east-1	South America (São Paulo)

Figure 1.27: List of available regions

13. The Command Prompt 's final input variable will look as follows. Then, press *Enter*:

```
(aws-env) C:\Users\[redacted]>aws configure
AWS Access Key ID [****************JERA]: AJSJKLEWEWNKLQKDDW
AWS Secret Access Key [****************GXzN]: fasfgfjfjydusgstrstsafdsa
Default region name [us-east-1]: us-east-1
Default output format [json]: json
```

Figure 1.28: The last step in AWS CLI configuration in the Command Prompt

Command Line-Interface (CLI) Usage

When using a command, specify at least one path argument. The two path arguments are LocalPath and S3Uri:

LocalPath: This represents the path of a local file or directory, which can be written as an absolute or relative path.

S3Uri: This represents the location of an S3 object, prefix, or Bucket. The command form is **`s3://myBucketName/myKey`**. The path argument must begin with **`s3://`**, to indicate that the path argument refers to an S3 object.

The overall command structure is **`aws s3 <Command> [<Arg> …]`**. The following table shows the different commands , with a description and an example:

Commands	Description	Example usage
mb	Creates a bucket	aws s3 mb s3://mybucket
cp	Copies a local file to an S3 bucket	aws s3 cp myNewFile.txt s3://mybucket
cp	Copies an s3 object to a local file	aws s3 cp s3://mybucket/myNewFile.txt myNewFile2.txt
ls	Lists the contents of an S3 bucket	aws s3 ls
rm	Deletes an S3 object	aws s3 rm s3://mybucket/myNewFile.txt
rb	Deletes an empty S3 bucket. To note: a bucket must be completely empty of objects before it can be deleted.	aws s3 rb s3://mybucket

Figure 1.29: Command list

Recursion and Parameters

Importing files one at a time is time-consuming, especially if you have many files in a folder that need to be imported. A simple solution is to use a recursive procedure. A recursive procedure is one that has the ability to call itself and saves you, as the user, from entering the same import command for each file.

Performing a recursive CLI command requires passing a parameter to the API. This sounds complicated, but it is incredibly easy. First, a parameter is simply a name or option that is passed to a program to affect the operation of the receiving program. In our case, the parameter is **recursive**, and the entire command to perform the recursive command is as follows:

```
aws s3 cp s3://myBucket . --recursive
```

With the command, is all of the s3 objects in a respective Bucket are copied to a specified directory:

Parameter	Description	Example usage
-recursive	If used with the cp command, all S3 objects in a respective bucket are copied to a specified directory	aws s3 cp s3//mybucket. –recursive

Figure 1.30: Parameter List

Activity 1: Importing and Exporting the Data into S3 with the CLI

In this activity, we will be using the CLI to create a Bucket in S3 and import a second text file. Suppose that you are an entrepreneur and you are creating a chatbot. You have identified text documents that contain content that will allow your chatbot to interact with customers more effectively. Before the text documents can be parsed, they need to be uploaded to an S3 Bucket. Once they are in S3, further analysis will be possible. To ensure that this has happened correctly, you will need to have installed Python, have an environment set up, and have a user authenticated with the CLI:

1. Configure the Command-Line Interface and verify that it is able to successfully connect to your AWS environment.
2. Create a new S3 Bucket.
3. Import a text file into the Bucket.
4. Export the file from the Bucket and verify the exported objects.

> **Note**
>
> To refer to the detailed steps, go to the *Appendix A* at the end of this book on Page no. 192

Using the AWS Console to Identify Machine Learning Services

The AWS Console provides a web-based interface to navigate, discover, and utilize AWS services for AI and ML. In this topic, we will explore two ways to use the Console to search Machine Learning services. In addition, we will test an ML API with text data retrieved from a website.

Exercise 4: Navigating the AWS Management Console

In this exercise, we will navigate the AWS Management Console to locate Machine Learning services. Starting from the console https://console.aws.amazon.com/console/ and only using console search features, navigate to the Amazon Lex https://console.aws.amazon.com/lex/ service information page:

1. Click on https://console.aws.amazon.com/console/ to navigate to the AWS Console. Then, click on **Services**:

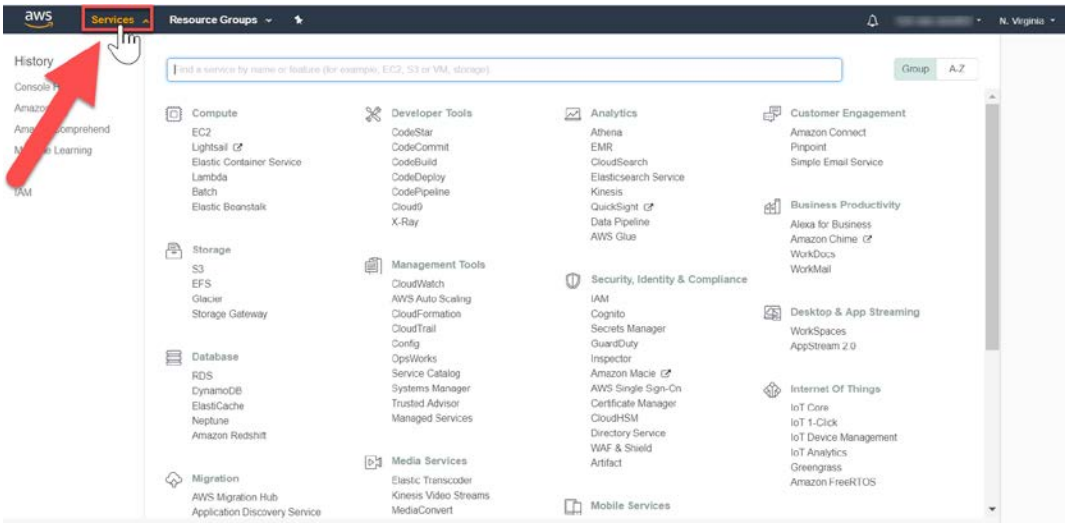


Figure 1.31: AWS Console

2. Scroll down the page to view all of the Machine Learning services. Then, click on **Amazon Lex**:

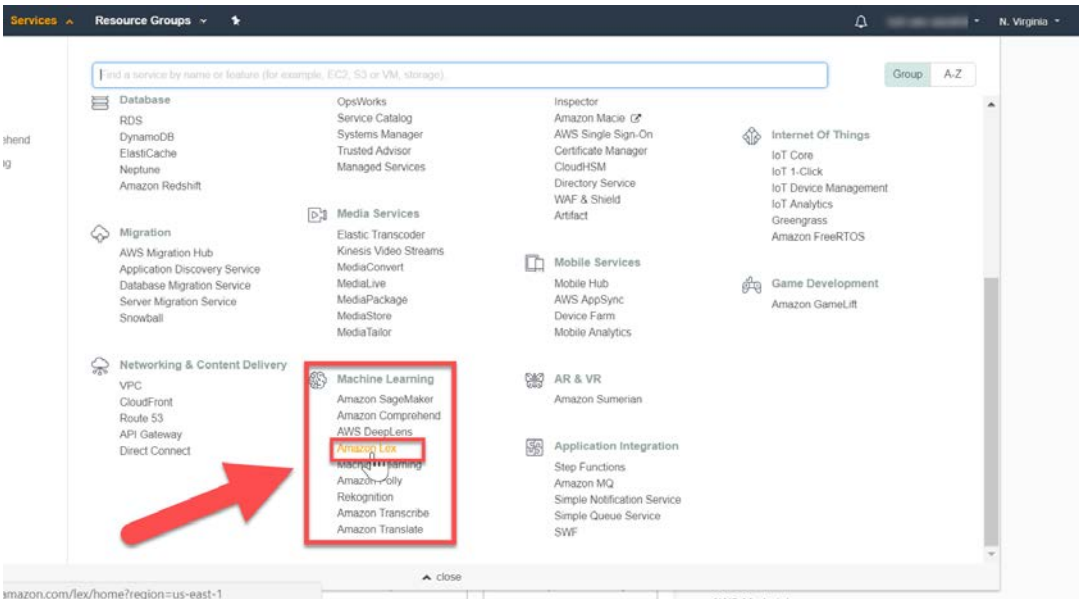


Figure 1.32: Options for Machine Learning

3. You will be redirected to the Amazon Lex home screen:

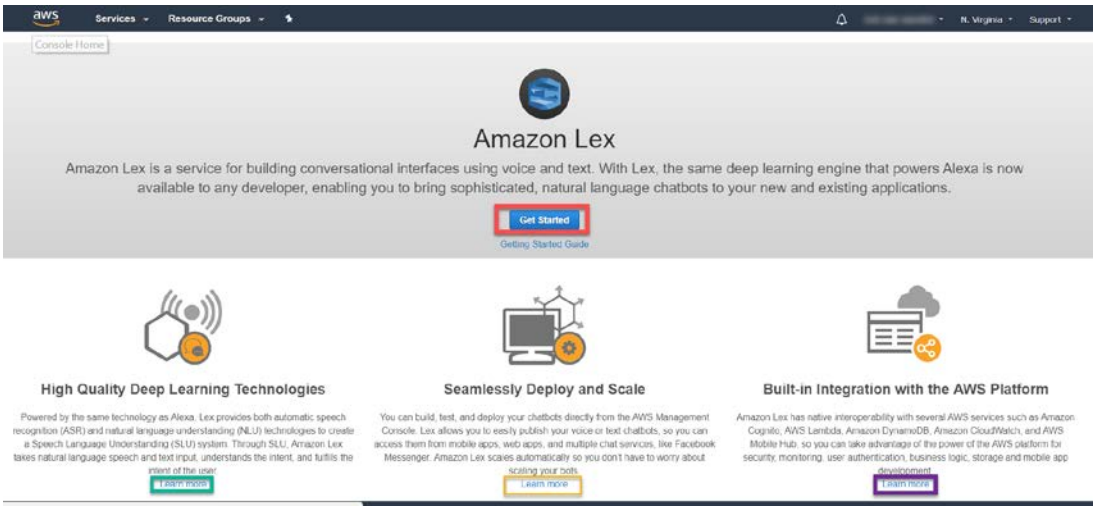


Figure 1.33: Amazon Lex home screen

Locating new AWS Services is an essential skill for discovering more tools to provide solutions for your data projects. Now, let's review another way to locate Machine Learning resources via the **Search bar**.

Activity 2: Testing the Amazon Comprehend's API Features

In this activity, we will display text analysis output by using a partial text file input in the API explorer. Exploring an API is a skill that saves development time by making sure that the output is in a desired format for your project. Thus, we will test Comprehend's text analysis features.

Suppose that you are an entrepreneur creating a chatbot. You have identified a business topic and the corresponding text documents, with content that will allow the chatbot to make your business successful. Your next step is to identify/verify an AWS service to parse the text document for sentiment, language, key phrases, and entities. Before investing time in writing a complete program, you want to test the AWS service's features via the AWS Management Console's interface. To ensure that this happens correctly, you will need to search the web for an article (written in English or Spanish) that contains the subject matter (sports, movies, current events, and so on) that you're interested in. The AWS Management Console is also accessible via the root user's account.

You are aware that exploring APIs is a skill that can save development time by ensuring that the output is in a desired format for your project. The following are the steps of completion:

1. Identify an AWS service, via the AWS Management Console, to accomplish your objectives.
2. Navigate to your web page of choice, which contains articles in English and Spanish.
3. Copy the text from the article written in English or Spanish, in order to identify the following features: sentiment, language, key phrases, and entities.
4. Obtain a score representing the articles: sentiment, language, key phrases, and entities.

> **Note**
>
> To refer to the detailed steps, go to the *Appendix A* at the end of this book on Page no. 194

Summary

At the beginning of this chapter, we explained what Amazon Web Services is, what Machine Learning is, and what Artificial Intelligence is. Following this, you learned what Amazon S3 is, and why Amazon S3 is used. You also explored the basic requirements for working with AWS using S3. With this, you used IAM (Identity and Access Management).

Next, you learned how to import and export data into S3. Following that, you explored the components of S3. At the same time, you learned about the REST Interface. In the last part of this chapter, we looked at the AWS command line and its usages. Finally, we explored the concepts of recursion and parameters, and how to use the AWS Console to identify Machine Learning services.

In the next chapter, you will learn how to summarize text documents by using Natural Language Processing (NLP). Researching new AWS services is essential for discovering additional solutions to solve any machine learning problems that you are working on.

2

Summarizing Text Documents Using NLP

Learning Objectives

By the end of this chapter, you will able to:

- Use Amazon Comprehend to examine text, in order to determine its primary language
- Extract information such as entities (people or places), key phrases (noun phrases that are indicative of the content), emotional sentiments, and topics in a set of documents
- Set up a Lambda function to process and analyze the imported text using Comprehend

This chapter describes the use of Amazon Comprehend to summarize the text documents and creating Lambda function to analyze the text.

Introduction

The Amazon Comprehend service continually learns from new data from Amazon.com product descriptions and consumer reviews, and thus, it perpetually improves its ability to understand a variety of topics from *Government*, *Health*, *Media*, *Education*, *Advertising*, and so on. Overall, Amazon Comprehend can analyze a collection of text documents and can organize the articles by topic, identify the most frequently mentioned features, and group articles by subject matter, to enable personalized recommendations to website visitors.

In the first part of this chapter, you learned how to use Amazon Comprehend to extract insights through using **Natural Language Processing (NLP)** from the contents of documents. Now, you will learn how to use the Amazon Comprehend API to produce insights by recognizing the language, entities, key phrases, sentiments, and topics in a document. This will allow you to understand deep learning-based NLP to build more complex applications, which we will do on Day 2.

In the second part of this chapter, you will learn about AWS Lambda, and how to integrate this service with Amazon Comprehend. You will also integrate a database to provide the foundation to build scalable NLP processing applications.

What is Natural Language Processing?

Amazon Comprehend is a Natural Language Processing (NLP) service. The overall goal of an NLP service is to make machines understand our spoken and written language. Virtual Assistants, such as Alexa or Siri, use NLP to produce insights from input data. The input data is structured by a language, which has a unique grammar, syntax, and vocabulary. Thus, processing text data requires identifying the language first, to apply subsequent rules to identify the document's information. NLP's general task is to capture this information as a numeral representation. The general task is further specified into specific tasks, such as identifying languages, entities, key phrases, emotional sentiments, and topics.

Amazon Comprehend processes any text file in UTF-8 format. It uses a pre-trained model to examine a document or set of documents, in order to gather insights about the document set. Amazon continuously trains the model so that there is no need to provide training data.

Using Amazon Comprehend to Inspect Text and Determine the Primary Language

Amazon Comprehend is used to gather insights from a variety of topics (Health, Media, Telecom, Education, Government, and so on) and languages in text data. Thus, the first step to analyze text data and utilize more complex features (such as topic, entity, and sentiment analysis) is to determine the dominant language. Determining the dominant language ensures the accuracy of more in-depth analysis.

To examine the text in order to determine the primary language, there are two operations (**`DetectDominantLanguage`** and **`BatchDetectDominantLanguage`**).

`DetectDominantLanguage` accepts a **`UTF-8`** text string that is at least 20 characters in length and must contain fewer than 5,000 bytes of UTF-8 encoded characters. **`BatchDetectDominantLanguage`** accepts an array of strings as a list. The list can contain a maximum of 25 documents. Each document should have at least 20 characters, and must contain fewer than 5,000 bytes of UTF-8 encoded characters.

The response includes what language was identified using a two-letter code. The following table shows the language codes for different countries:

> **Note**
>
> Check out https://docs.aws.amazon.com/comprehend/latest/dg/how-languages.html for an updated list of supported languages.

Code	Language	Code	Language	Code	Language
af	Afrikaans	hy	Armenian	ps	Pushto
am	Amharic	ilo	Iloko	qu	Quechua
ar	Arabic	id	Indonesian	ro	Romanian
as	Assamese	is	Icelandic	ru	Russian
az	Azerbaijani	it	Italian	sa	Sanskrit
ba	Bashkir	jv	Javanese	si	Sinhala
be	Belarusian	ja	Japanese	sk	Slovak
bn	Bengali	kn	Kannada	sl	Slovenian
bs	Bosnian	ka	Georgian	sd	Sindhi
bg	Bulgarian	kk	Kazakh	so	Somali
ca	Catalan	km	Central Khmer	es	Spanish
ceb	Cebuano	ky	Kirghiz	sq	Albanian
cs	Czech	ko	Korean	sr	Serbian
cv	Chuvash	ku	Kurdish	su	Sundanese
cy	Welsh	la	Latin	sw	Swahili
da	Danish	lv	Latvian	sv	Swedish
de	German	lt	Lithuanian	ta	Tamil
el	Greek	lb	Luxembourgish	tt	Tatar
en	English	ml	Malayalam	te	Telugu

Figure 2.1: Amazon Comprehend supported languages

The reaction also includes a score that indicates the certainty level that Amazon Comprehend has that a specific language is the dominant language in the document (see the following screenshot). The language scores are independent of other scores, and so reaction does not provide the percentage of the document that is represented by a particular language:

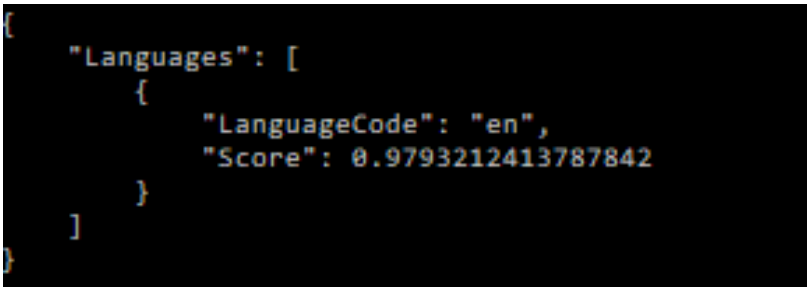


Figure 2.2: Dominant language score confidence output

Exercise 5: Detecting the Dominant Language Using the Command-Line

Interface in a text document

In this exercise, you will learn how to detect Comprehend's using **`detectDominantLanguage`** function. The following steps describe how to detect the dominant language:

> **Note**
>
> The source code is available via GitHub in the repository at: https://github.com/TrainingByPackt/Machine-Learning-with-AWS/blob/master/lesson2/topic_a/detect_dominant_languages.py.

1. First, we must import the AWS SDK for Python (boto3) http://boto3.readthedocs.io/en/latest/:

```
import boto3
```

2. Then, import the JSON module to serialize the JSON https://docs.python.org/3.6/library/json.html :

```
import json
```

3. Replace <**`input region`**> with your unique region (for example, **`us-east-1`**). The following instantiates a new Comprehend client:

```
comprehend = boto3.client(service_name='comprehend', region_name='<input
region>')
```

4. Next, we assign English and Spanish strings to be analyzed by Comprehend:

```
english_string = 'Machine Learning is fascinating.'
spanish_string = 'El aprendizaje automático es fascinante.'
```

5. Next, we print a string to indicate the respective variable that our script is about to execute:

```
print('Calling DetectDominantLanguage')
print('english_string result:')
```

6. Lastly, call Comprehend's **`detect_dominant_language`** method with the **`englih_string`** and **`spanish_string`** variables https://docs.aws.amazon.com/comprehend/latest/dg/API_DetectDominantLanguage.html . **`json.dumps()`** writes the JSON data to a Python string in the terminal:

```
print(json.dumps(comprehend.detect_dominant_language(Text = english_
string), sort_keys=True, indent=4))
```

7. Save the changes to the file. Open a command prompt, if you haven't already, and activate your virtual environment.

8. Navigate to the **detect_dominant_languages.py** location. Type **python detect_dominant_languages.py** in the command prompt. Executing this command will produce the following output (see the following screenshot):

   As expected, the **english_text** string is identified as English (with the "**en**" language code) with a ~0.99 confidence score (see the following output).

   Also as expected, the **spanish_text** string is identified Spanish (with the "**es**" language code) with a ~0.99 confidence score (see the following output):



Figure 2.3: Detecting the dominant language output – English and Spanish

Exercise 6: Detecting the Dominant Language in Multiple Documents by Using the Command-Line Interface (CLI)

In this exercise, you will learn how to detect Comprehend's **detectDominantLanguage** operation for multiple documents. The following steps describe how to detect the dominant language:

> **Note**
>
> The source code is available via GitHub in the repository at: https://github.com/TrainingByPackt/Machine-Learning-with-AWS/blob/master/lesson2/topic_a/batch_detect_dominant_languages.py.

1. First, we import the AWS SDK for Python (boto3) http://boto3.readthedocs.io/en/latest/:

   ```
   import boto3
   ```

2. Then, we import the JSON module to serialize the JSON https://docs.python.org/3.6/library/json.html :

   ```
   import json
   ```

3. Replace <**input region**> with your unique region (for example, 'us-east-1'). The following instantiates a new Comprehend client:

   ```
   comprehend = boto3.client(service_name='comprehend', region_name='<input
   region>')
   ```

4. Next, assign a list of English and Spanish strings to be analyzed by Comprehend:

   english_string_list = ['Machine Learning is fascinating.', 'Studying Artificial Intelligence is my passion.']

   spanish_string_list = ['El aprendizaje automático es fascinante.', 'Estudiar Inteligencia Artificial es mi pasión.']

5. Lastly, we call Comprehend's "**batch_detect_dominant_language**" method with the **englighstring_list** and **spanish_string_list** variables https://docs.aws.amazon.com/comprehend/latest/dg/API_DetectDominantLanguage.html. Then, **json.dumps()** writes the JSON data to a Python string to the terminal:

   ```
   print(json.dumps(comprehend.batch_detect_dominant_language(Text = english_
   string_list), sort_keys=True, indent=4))
   ```

The important concepts to remember are that Comprehend has the ability to detect different languages and can take text input as a single string or in batch format as a list of strings.

In this chapter, we reviewed how Comprehend's **`DetectDominantLanguage`** method is structured, and how to pass in both strings and a list of strings. Next, we will extract entities, phrases, and sentiments from a set of documents.

Extracting Information in a Set of Documents

At a business level, knowing if and why a customer is angry or happy when they text a Virtual assistant is extremely important, in order to retain the customer. At an NLP level, this requires more information to extract and a more complex algorithm. The additional information to extract, and quantify are **`entities`**, **`key phrases`**, **`emotional sentiment`**, and **`topics`**.

Detecting Named Entities – AWS SDK for Python (boto3)

An entity is a textual reference to the unique name of a real-world object, such as people, places, commercial items, and precise references to measurements such as dates and quantities. For example, in the text "Martin lives at 27 Broadway St.", **Martin** might be detected as a **PERSON**, while **27 Broadway St** might be detected as a **LOCATION**.

Entities also have a score to indicate the confidence level that the entity type was detected correctly. The following table shows a complete list of entity types and descriptions:

Type	Description
COMMERCIAL_ITEM	A branded device, product, merchandise, etc.
DATE	A full date (for example, 10/22/2018), day (Wednesday), month (June), or time (10:30 a.m.)
EVENT	An event, such as a celebration, ceremony, holiday, etc.
LOCATION	A specific location, such as a state, city, pond, apartment, etc.
ORGANIZATION	Large organizations, such as an institution, government, assembly, company, religion, sports team, etc.
OTHER	Entities that don't fit into any of the other entity categories
PERSON	Individuals, customers, groups of people, nicknames, fictional characters
QUANTITY	A quantified amount, such as currency, percentages, numbers, bytes, etc.
TITLE	An official name given to any creation or creative work, such as painting, article, play, etc.

Figure 2.4: AWS Comprehend entity types and descriptions

DetectEntites – Input and Output

DetectEntites takes a **`LanguageCode`** and string of text as an input, and then provides the following information about each entity within the input text: **`BeginOffset`**, **`EndOffset`**, **`Score`**, **`Text`**, and **`Type`**. The following table shows a complete list of AWS Comprehend DetectEntities, types, and descriptions:

Type	Description	Type	Required
Begin Offset	A character offset in the input text that shows where the entity begins (the first character is at position 0). The offset returns the position of each UTF-8 code point in the string.	Integer	No
EndOffset	A character offset in the input text that shows where the entity ends. The offset returns the position of each UTF-8 code point in the string. A code point is the abstract character from a particular graphical representation.	Integer	No
Score	The level of confidence that Amazon Comprehend has in the accuracy of the detection.	Float	No
Text	The text of the entity.	String	No
Type	The entity's type. (For example: PERSON, LOCATION, ORGANIZATION, COMMERCIAL_ITEM, EVENT, DATE, QUANTITY, TITLE, or OTHER)	String	No

Figure 2.5: AWS Comprehend entity types and descriptions

Exercise 7: Determining the Named Entities in a Document

In this exercise, we will determine the named entities in a document. For this, we will use Amazon Comprehend's **`DetectEntities`** operation. The following are the steps for detecting the Named Entities:

> **Note**
>
> The source code is available via GitHub in the repository at: https://github.com/TrainingByPackt/Machine-Learning-with-AWS/blob/master/lesson2/topic_b/detect_entities.py.

1. Navigate to the **`detect_entities.py`** location, replace **`<input region>`** with your specific region, and save the file.

Now, import the AWS SDK for python (boto3) https://boto3.amazonaws.com/v1/documentation/api/latest/index.html by using the following command:

```
import boto3
```

2. Now, import the **JSON** module to serialize **JSON** from https://docs.python.org/3.6/library/json.html by using the following command:

```
import json
```

3. Now, instantiate a new Comprehend client:

```
comprehend = boto3.client(service_name='comprehend', region_name='<input
region>')
```

4. Now, after instantiating a new Comprehend, provide **English** text to analyze **english_string = "I study Machine Learning in Seattle on Thursday.":**

```
print('Calling DetectEntities')
```

5. Now, **json.dumps()** writes JSON data to a Python string:

```
print(json.dumps(comprehend.detect_entities(Text = english_string,
LanguageCode='en'), sort_keys=True, indent=4))
print('End of DetectEntities\n')
```

6. Run the code by executing the **detect_entities.py** command with **python**. The output of the preceding code is shown in the following screenshot:

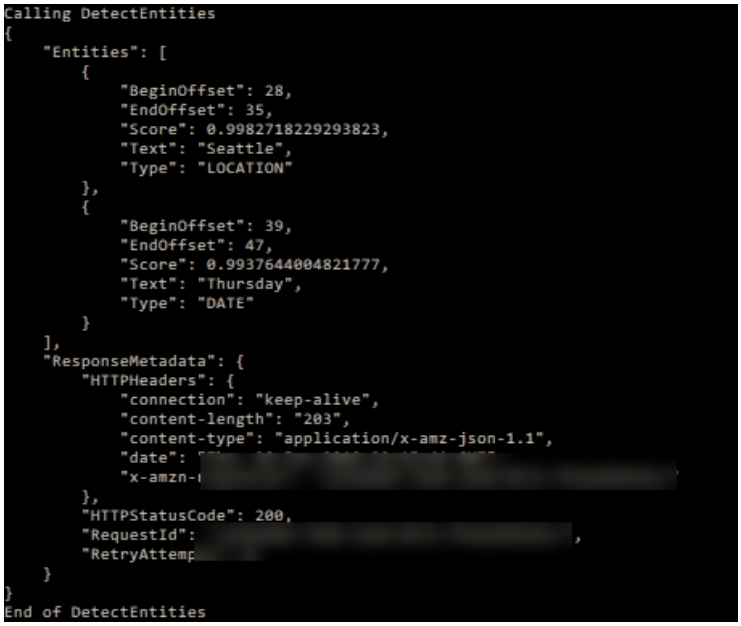


Figure 2.6: AWS Comprehend DetectEntities output

The confidence scores were both ~0.99, as the inputs were simple examples. As expected, **Seattle** was detected as a **LOCATION**, and **Thursday** was detected as the **DATE**:

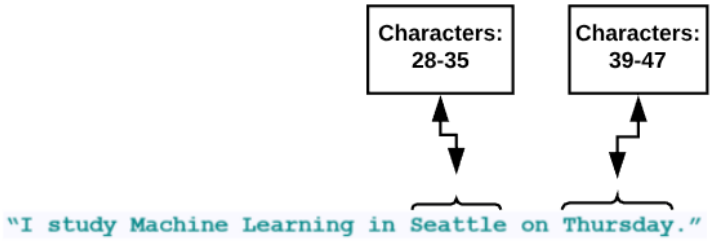


Figure 2.7: AWS Comprehend BeginOffset and EndOffset review

DetectEntities in a Set of Documents (Text Files)

We are now going to extract entities from a set of documents (text files). Navigate to the https://github.com/TrainingByPackt/Machine-Learning-with-AWS/blob/master/lesson2/topic_b/detect_entities_from_documents.py location, replace '<**input region**>' with your specific region, and save the file. Run the code by executing the command with **`python detect_key_phrases.py`**:

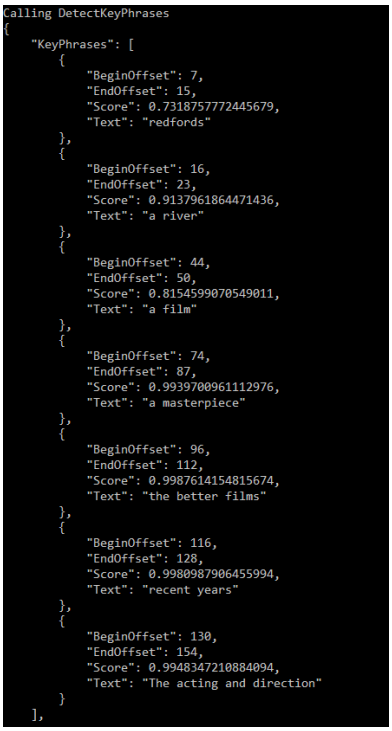


Figure 2.8: DetectKeyPhrases output

Detecting Key Phrases

A key phrase for AWS is analogous to a noun phrase, which represents an actual thing. In English when we put together different words that represent one concrete idea we call it a noun phrase. For example, "**A fast machine**" is a noun phrase because it consists of "**A**", the article, "**fast**", an adjective, and "**machine**" which is a noun. AWS looks for appropriate word combinations and gives scores that indicates the confidence that a string is actually a noun phrase.

Exercise 8: Determining the Key Phrase Detection.

In this exercise, we will determine the key phrase detection. To do so we will use Amazon Comprehend's **`DetectKeyPhrase`** operation. The following are the steps for detecting the Named Entities:

> **Note**
>
> The source code is available via GitHub in the repository at: https://github.com/TrainingByPackt/Machine-Learning-with-AWS/blob/master/lesson2/topic_b/detect_key_phrases.py.

1. Navigate to the **`detect_key_phrases.py`** location, replace **`<input region>`** with your specific region, and save the file. Import the AWS SDK for python (boto3) http://boto3.readthedocs.io/en/latest/ by using the following command:

```
import boto3
```

2. Now, import the JSON module to serialize the JSON from https://docs.python.org/3.6/library/json.html by using the following command:

```
import json
```

3. Now, instantiate a new Comprehend client by using the following code:

```
comprehend = boto3.client(service_name='comprehend', region_name='<input
region>')
```

4. Now, provide **English** text to analyze, using the following code:

```
english_string = 'I study Machine Learning in Seattle on Thursday.'
print('Calling DetectKeyPhrases')
# json.dumps() writes JSON data to a Python string
print(json.dumps(comprehend.detect_entities(Text = english_string,
LanguageCode='en'), sort_keys=True, indent=4))
print('End of DetectKeyPhrases\n')
```

5. Run the code by executing the command with **`python detect_key_phrases.py`**. You will see the following output:

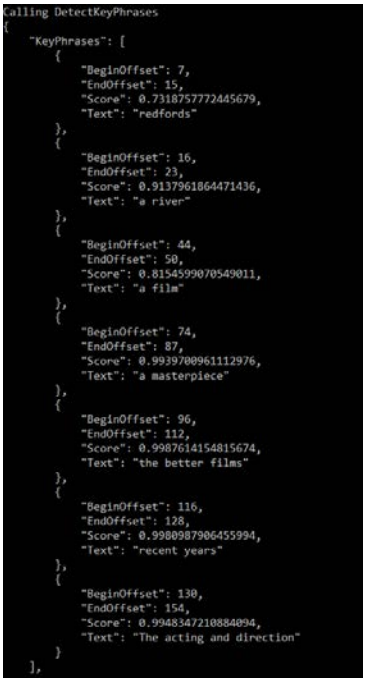


Figure 2.9: AWS Comprehend DetectKeyPhrases output

Detecting Sentiments

Amazon Comprehend can be used to determine the sentiment of a document. You can determine whether the sentiment is positive, negative, neutral, or mixed. For example, you can use sentiment analysis to determine the sentiments of comments on a blog post, to determine whether your readers liked the post.

Exercise 9: Detecting Sentiment Analysis

In this exercise, we will determine the sentiment analysis. To do so, we will use Amazon Comprehend's **`DetectSentiment`** operation. The following are the steps for detecting Sentiment Analysis:

> **Note**
>
> The source code is available via Github in the repository at: https://github.com/TrainingByPackt/Machine-Learning-with-AWS/blob/master/lesson2/topic_b/detect_sentiment.py.

1. Navigate to the **detect_sentiment.py** location, replace '<**input region**>' with your specific region, and save the file. Import the **AWS SDK** for Python (boto3) from http://boto3.readthedocs.io/en/latest/ by using the following command:

```
import boto3
```

2. Now, import the **JSON** module to serialize JSON from https://docs.python.org/3.6/library/json.html by using the following command:

```
import json
```

3. Now, instantiate a new comprehend client, using the following code:

```
comprehend = boto3.client(service_name='comprehend', region_name='<input
region>')
```

4. Then, provide a text string to analyze, using the following code:

```
english_string = 'Today is my birthday, I am so happy.'

print('Calling DetectSentiment')
json.dumps() writes JSON data to a Python string
print('english_string results:')
print(json.dumps(comprehend.detect_sentiment(Text = english_string,
LanguageCode='en'),
sort_keys=True,
      indent=4))
print('End of DetectSentiment\n')
```

Run the code by executing the command with: **`python detect_seniment.py`**. The output is shown as follows:



Figure 2.10: AWS Comprehend – DetectSentiment output

Setting up a Lambda function and Analyzing Imported Text Using Comprehend

In this topic, we will be integrating AWS Lambda functions to Comprehend, which provides a more powerful, scalable infrastructure. You can use AWS Lambda to run your code in response to events, such as changes to data in an Amazon S3 Bucket.

Executing code in response to events provides a real-world solution for developing scalable software architecture. Overall, this increases our data pipeline and provides the ability to handle more complex Big Data volumes and NLP operations.

What is AWS Lambda?

AWS Lambda is a compute service that runs code without provisioning or managing servers. AWS Lambda executes the code only when needed, and scales automatically. AWS Lambda runs your code on high availability compute infrastructure, which performs the administration of the compute service. More specifically, AWS Lambda performs the following: server and operating system maintenance, capacity provisioning and automatic scaling, code monitoring, and logging.

Overall, the goal of a Lambda is to make short, simple, modular code segments that you can tie together into a larger processing infrastructure.

What does AWS Lambda do?

Lambda allows users to run small segments of code (Java, Node, or Python) to complete a specific task. These specific tasks can be storing and then executing changes to your AWS setup, or responding to events in S3 (we will explore the latter later on in this topic). Before Lambda, you would typically need a separate EC2 server to run your entire code; however, Lambda allows small segments of the code to run without the need for EC2.

Lambda Function Anatomy

AWS Lambda provides two options for implementing Python code. First, you can upload a complete Python code file. Second, you can use the Lambda function editor entirely in-line, which means that you can enter and modify the code directly, without having to upload any files to AWS. The code that you enter will be executed when the Lambda function is invoked. The second option will allow for easier testing, so we will use it.

Let's examine the structure of the Lambda function:

- When you create a function (for example, **`s3_trigger`**), AWS creates a folder named the same, with a Python file named **`Lambda_function.py`** within the **folder**. This file contains a stub for the **`Lambda_handler`** function, which is the entry point of our Lambda function. The entry point takes two parameters as arguments:
- The event argument provides the value of the payload, which is sent to the function from the **`calling`** process. It typically takes the form of a Python **`dict`** type, although it could also be one of list, **`str`**, **`int`**, **`float`**, or **`NoneType`**.
- The context argument is of the type **LambdaContext** and contains runtime information. You will be using this parameter for an exercise in a later section. The return value of the function can be any type that is JSON serializable. This value gets returned to the calling application, after serializing.

We will incorporate Lambda, S3, and Amazon Comprehend, in order to automatically perform document analysis when a text document is uploaded to S3. The architecture of a Lambda function is as follows:

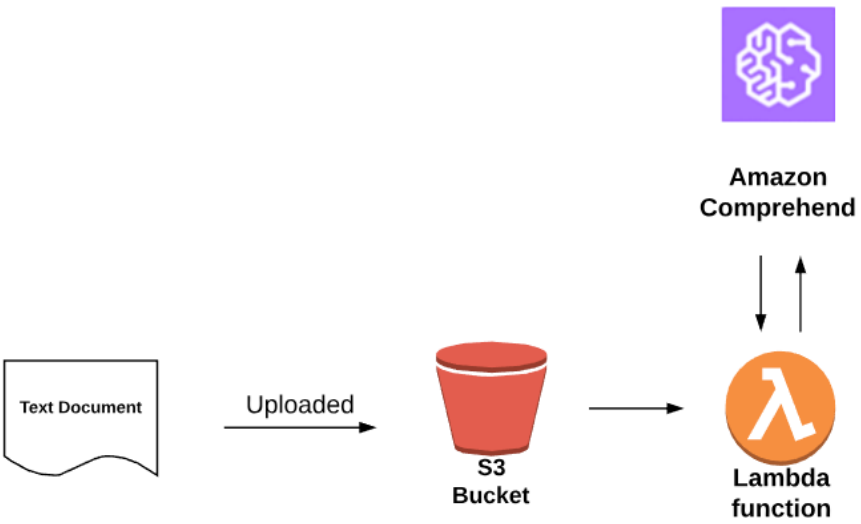


Figure 2.11: Architecture diagram

Exercise 10: Setting up a Lambda function for S3

In this exercise, we will integrate the following AWS services: S3, Lambda, and Amazon Comprehend. For performing this exercise, the architecture should be recollected. Upload a file (**`test_s3trigger_configured.txt`**) to S3 and view the results from Comprehend's analysis. The following are the steps for setting up a Lambda function.

Creating the S3 Bucket

1. First, navigate to the Amazon S3 service, https://console.aws.amazon.com/s3/, and click on **Create bucket**:

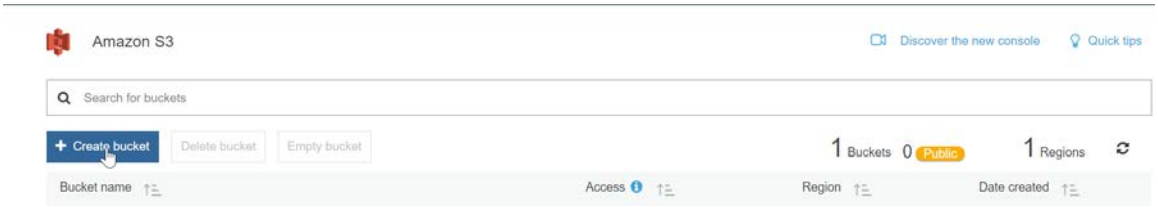


Figure 2.12: S3 Bucket creation for the Lambda trigger

2. For the Bucket name, type **`aws-ml-s3-trigger`**, and then click on **Create**:

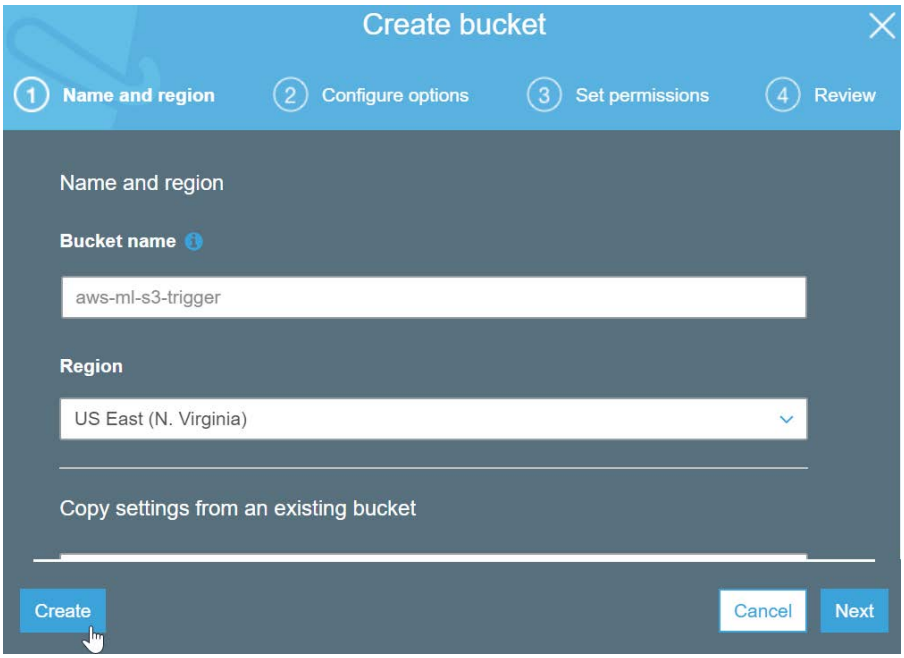


Figure 2.13: Creating an S3 Bucket

3. Your **`Bucket`** will be created, and you will be redirected to the **Bucket list**:

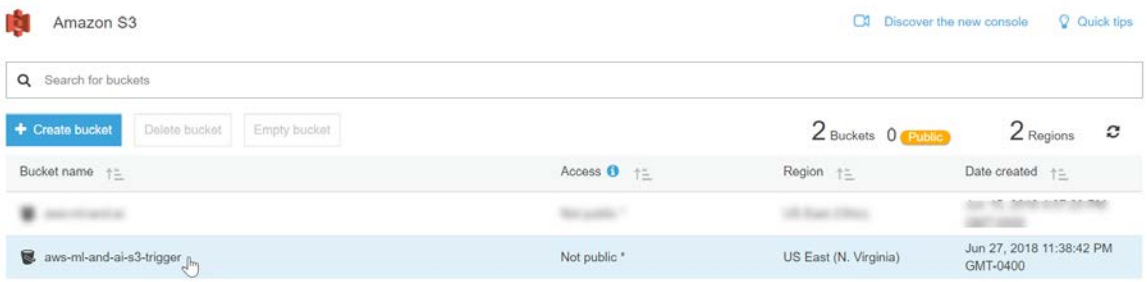


Figure 2.14: S3 Bucket list screen

4. Next, navigate to https://console.aws.amazon.com/Lambda/ and click on **Create a function**:

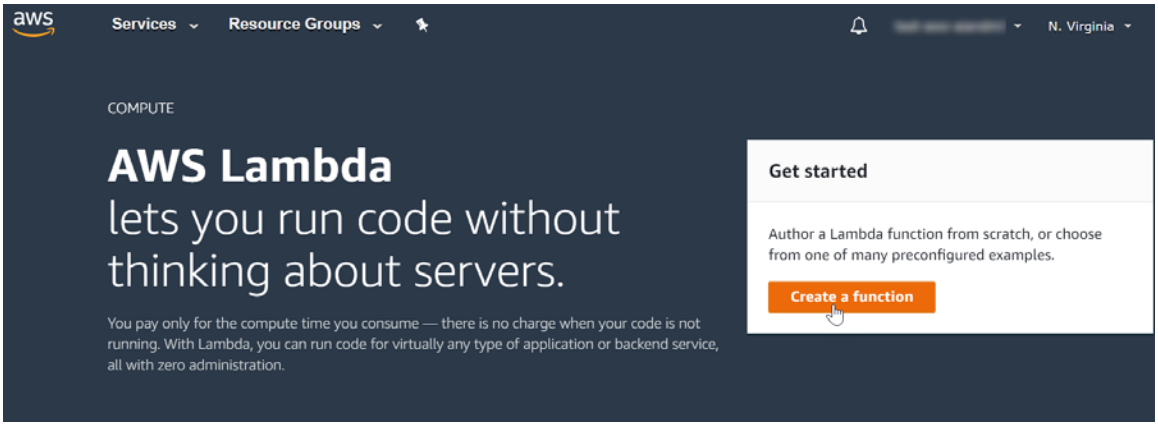


Figure 2.15: AWS Lambda home screen.

5. Choose Author from scratch from the options. For Name, **`type s3_trigger`**:

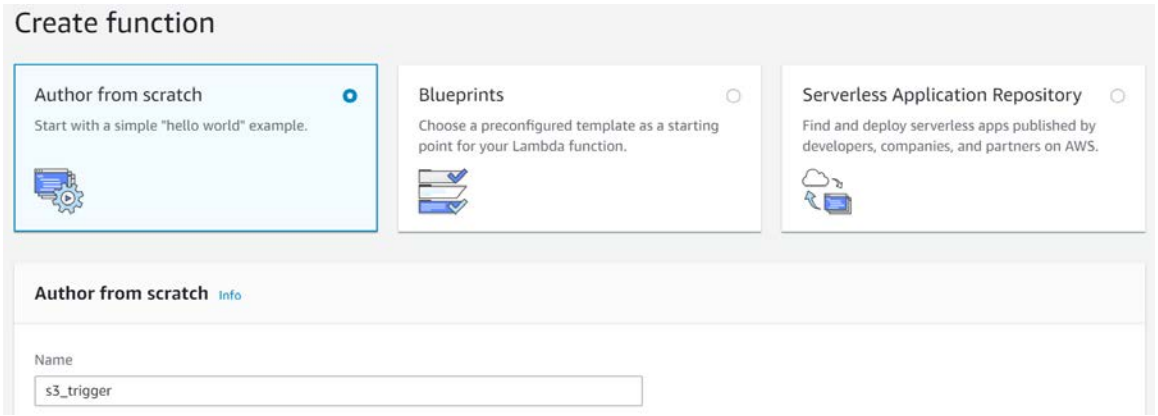


Figure 2.16: AWS Lambda – Creating a function with the "author from scratch" selection

6. For the runtime options, choose **`Python 3.6`** from the list:

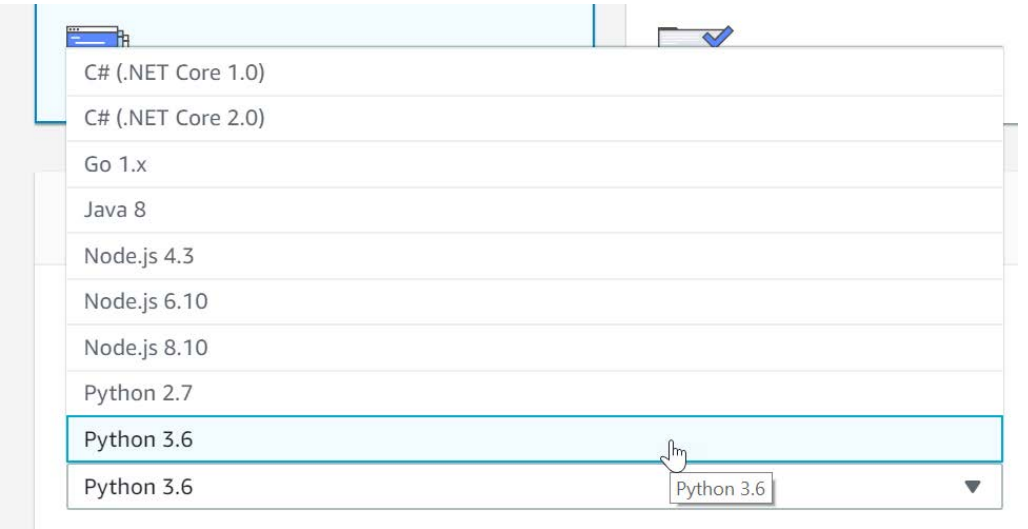


Figure 2.17: AWS Lambda – Python 3.6 selection

7. For the Role field, choose **Create new role from template(s)** from the drop-down menu, and enter the name **`s3TriggerRole`** in the Role name field. Then, click on the **Create function** button to create the **`Lambda function`** in AWS:

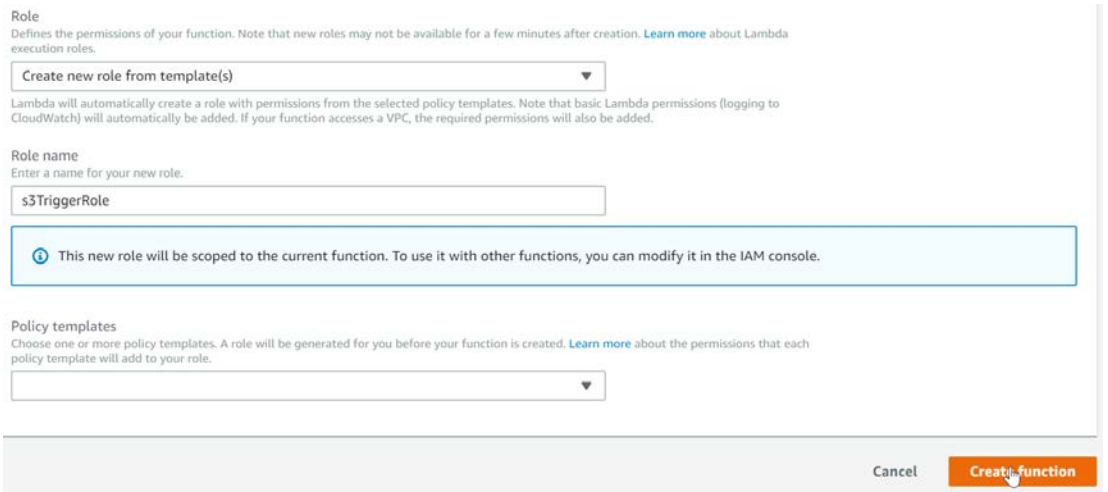


Figure 2.18: AWS Lambda – Create function screen

8. Then, click on the drop-down menu under **Policy templates**:

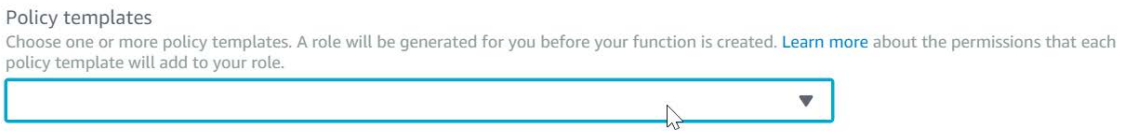


Figure 2.19: Policy template selection drop-down menu

9. Select **Amazon S3** object read-only permissions:

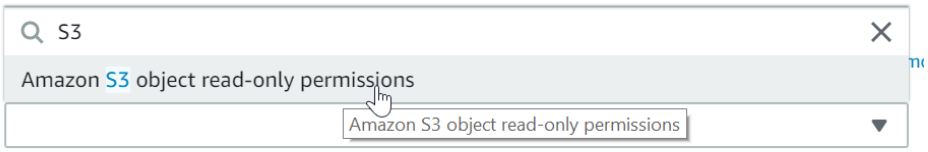


Figure 2.20: Amazon S3 object read-only permissions selection

10. Then, click on the **Create function** button to create the **`Lambda function`**:

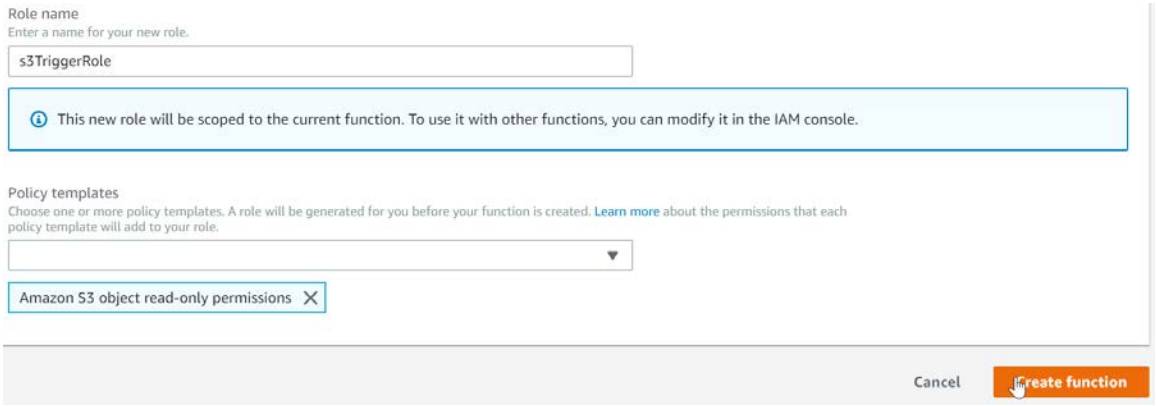


Figure 2.21: Clicking on Create function

> **Note**
>
> If you receive a **`NoSuchEntity`** error, this is a temporary warning that occurs when Amazon creates the service role for the **`s3_trigger`**. AWS has a reference to the possible temporary issues under the Roles heading. This will not affect your ability to continue with the chapter. Refresh your screen, and in a few minutes the warning message should disappear.

11. You will know that the issues have been resolved when the **Amazon CloudWatch Logs** service becomes available as a resource for the function's role:

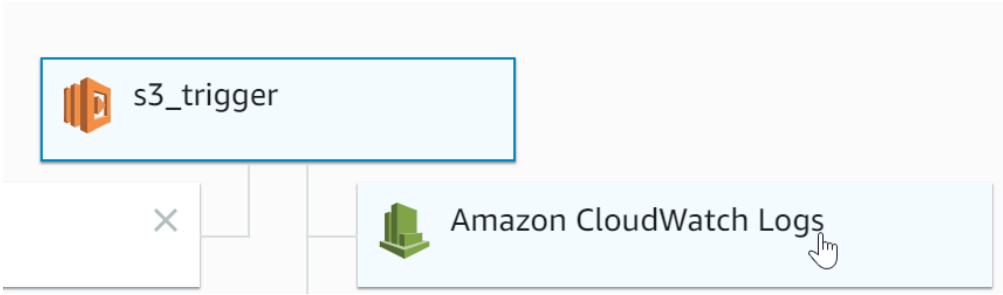


Figure 2.22: Amazon CloudWatch Logs

12. You should see a configuration screen, as follows:

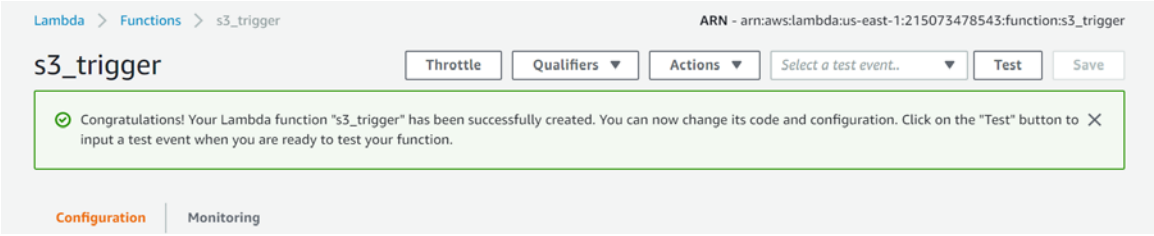


Figure 2.23: AWS Lambda successful s3_trigger verification screen

13. Now, let's add S3 as a trigger. Under **Add triggers**, scroll to S3:

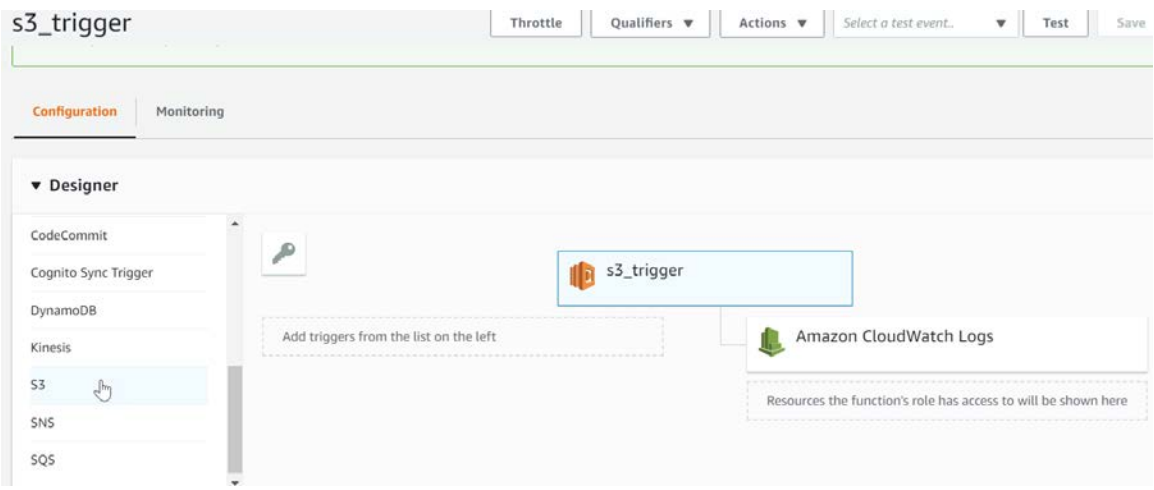


Figure 2.24: AWS Lambda – Adding S3 as a trigger selection screen

14. Click **S3,** and it will auto-populate under the **s3_trigger**. After clicking **S3**, your screen will look as follows:

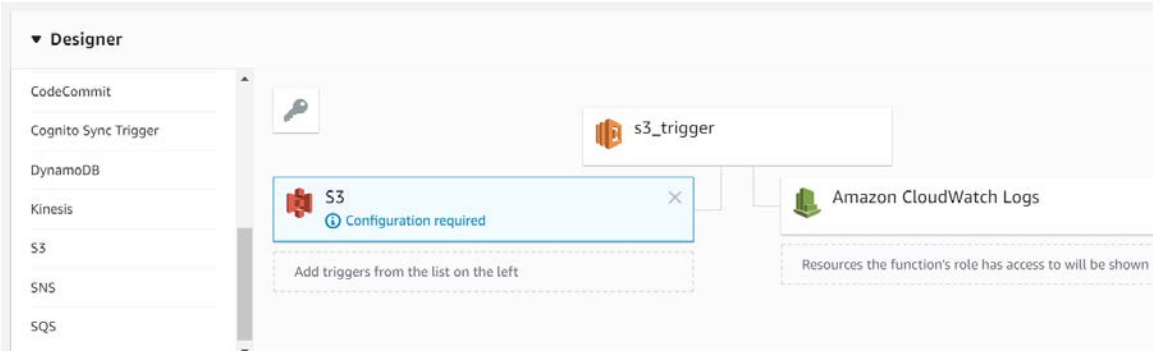


Figure 2.25: AWS Lambda – S3 trigger selection, configuration required

Exercise 11: Configuring the Trigger for an S3 Bucket

In this exercise, we will configure the trigger for the **Bucket** created in the preceding exercise. To configure the trigger, follow these steps:

1. Scroll down the screen to the configure triggers section from **Bucket** and choose the Bucket that you created, that is, **aws-ml-s3-trigger**.:

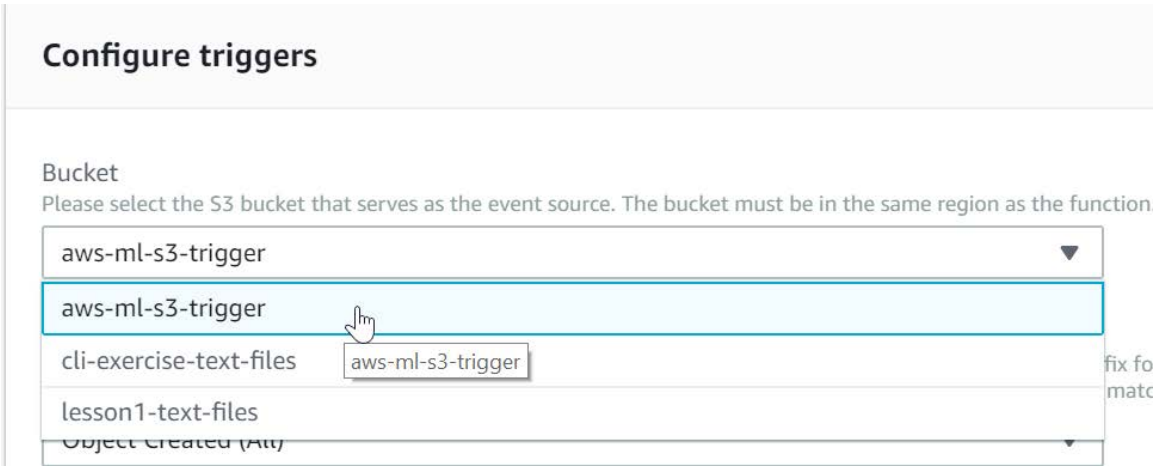


Figure 2.26: AWS Lambda – Configuring S3 trigger, S3 Bucket selection

2. Leave the remaining default settings. Next, scroll down the screen and click on **Add**:

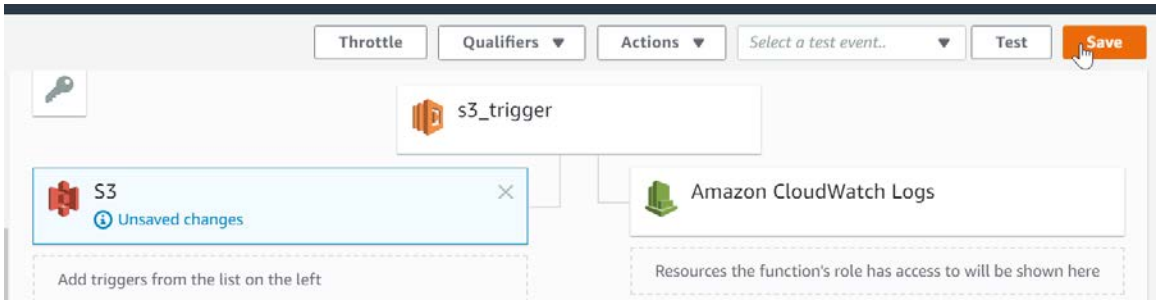
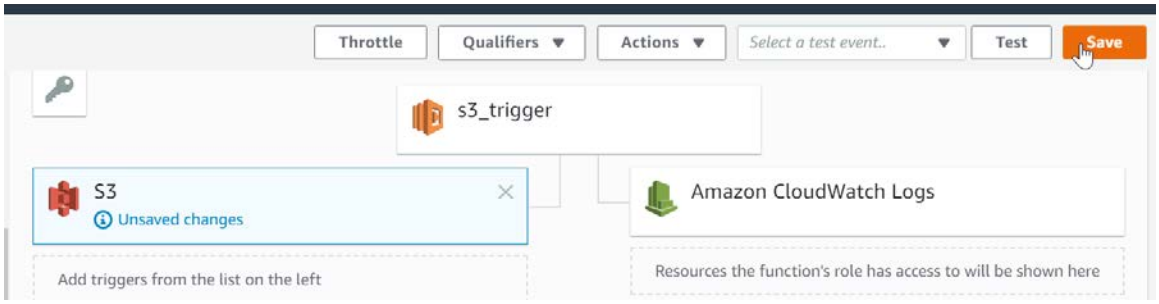


Figure 2.27: AWS Lambda – Adding the S3 Bucket as a trigger

3. The next step is to click on the **Save** button:



AWS Lambda – Saving the S3 trigger

4. Next, scroll down the screen to the **Function code** section. The default code will be the same as or similar to, the following:

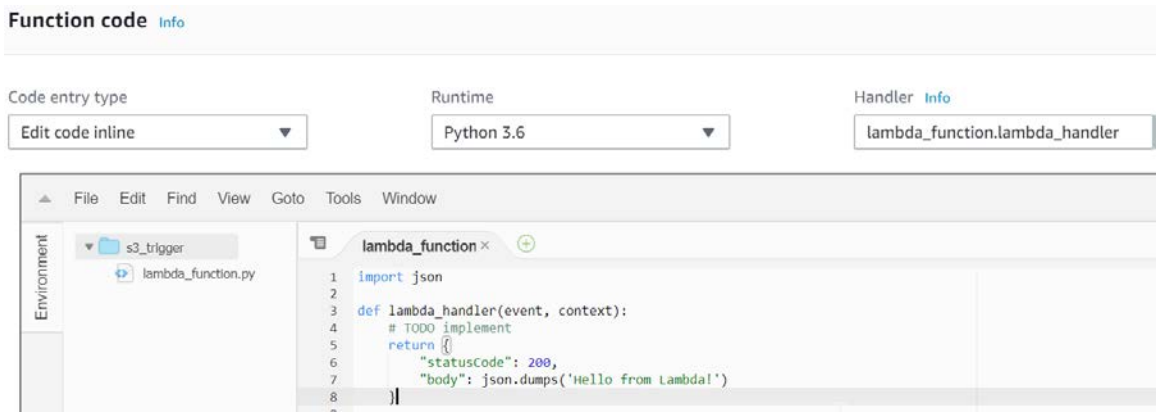


Figure 2.28: AWS Lambda The default Lambda_function screen

Here, we can enter and edit our code entirely within the Lambda function screen (as long as the Code entry type is set to Edit code inline, which is the default value in the drop-down menu).

> **Note**
>
> For this step, you may either follow along and type in the code, or obtain it from the source code folder at https://github.com/TrainingByPackt/Machine-Learning-with-AWS/blob/master/lesson2/topic_c/s3_trigger.py file.

5. First, we import the **AWS SDK** for Python (boto3) http://boto3.readthedocs.io/en/latest/:

```
import boto3
```

6. Next, create a function that takes two parameters-event and context:

```
def Lambda_handler(event, context):
```

7. Next, create the s3 client object:

```
s3 = boto3.client("s3")
```

8. Add an **if** event to check whether an event occurs.

9. Next, replace **<input Bucket name>** with the Bucket you created (**aws-ml-s3-trigger**, in my example):

```
Bucket = "<input Bucket name>"
```

10. Next, access the event **Records** first index to obtain the text file object:

```
text_file_obj = event["Records"][0]
```

11. Next, assign the text **filename** to a variable, and print the filename:

```
filename = str(text_file_obj['s3']['object']['key'])
print("filename: ", filename)
```

12. Next, create the file object by getting the Bucket and key:

```
file_obj = s3.get_object(Bucket = Bucket, Key = filename)
```

13. Assign the text to the **body_str_obj** variable:

```
body_str_obj = str(file_obj['Body'].read())
```

14. Replace <**input region name**> with your specific region. In addition, create the comprehend variable (us-east-1, in my example):

```
comprehend = boto3.client(service_name="comprehend", region_name='<input
region_name>')
```

15. The next three lines of code call the respective comprehend functions to detect the sentiment, entities, and key phrases from the text document. Then, the output is printed to the console:

```
sentiment_response = comprehend.detect_sentiment(Text = body_str_obj,
LanguageCode = "en")
print("sentiment_response: \n", sentiment_response)
entity_response = comprehend.detect_entities(Text = body_str_obj,
LanguageCode = "en")
print("\n\nentity_response: \n", entity_response)
key_phases_response = comprehend.detect_key_phrases(Text = body_str_obj,
LanguageCode = "en")
print("\n\nkey_phases_response: \n", key_phases_response)
```

16. The final statement returns the string 'Hello from Lambda', like so:

```
return 'Hello from Lambda'
```

17. Now, click on the **Save** button:

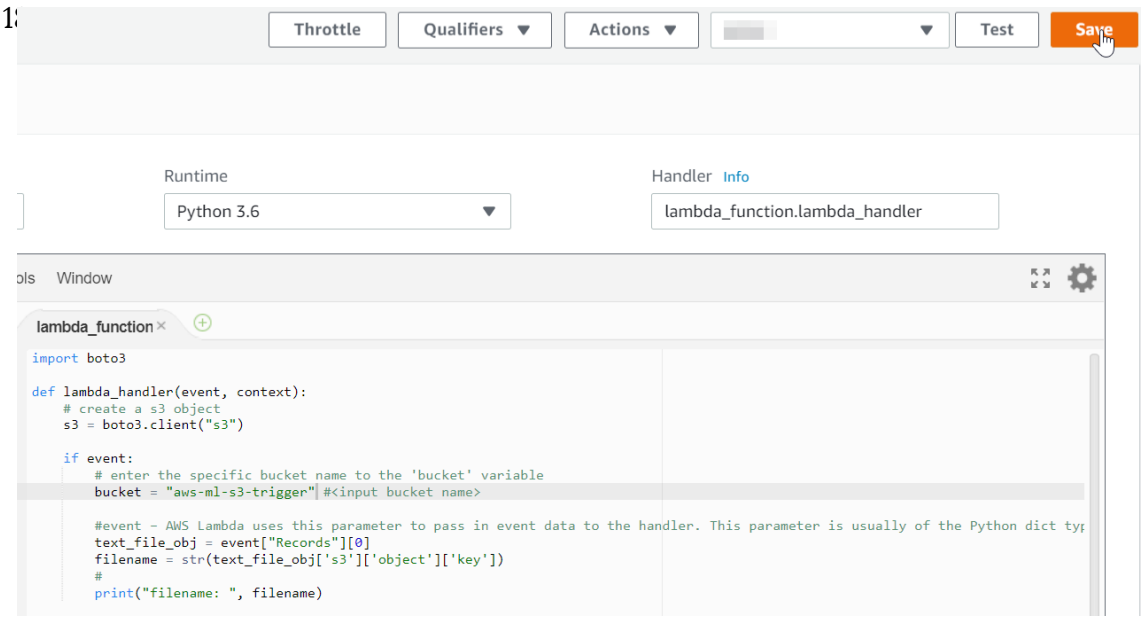


Figure 2.29: AWS Lambda – Save screen

From this exercise, the **`s3_trigger`** function has access to S3, but not Amazon Comprehend. We need to attach a policy to the **`s3_trigger`** function to allow it to access Amazon Comprehend to execute the text analysis functions (**`detect_sentiment`**, **`detect_entities`**, and **`detect_key_phrases`**).

Exercise 12: Assigning Policies to S3_trigger to Access Comprehend

In this exercise, we will attach the policies to the **`S3_trigger`** function, in order to allow it to access comprehend. The steps for completion for assigning the policies are as follows:

1. Navigate to the Identity and Access Management dashboard at https://console.aws.amazon.com/iam:

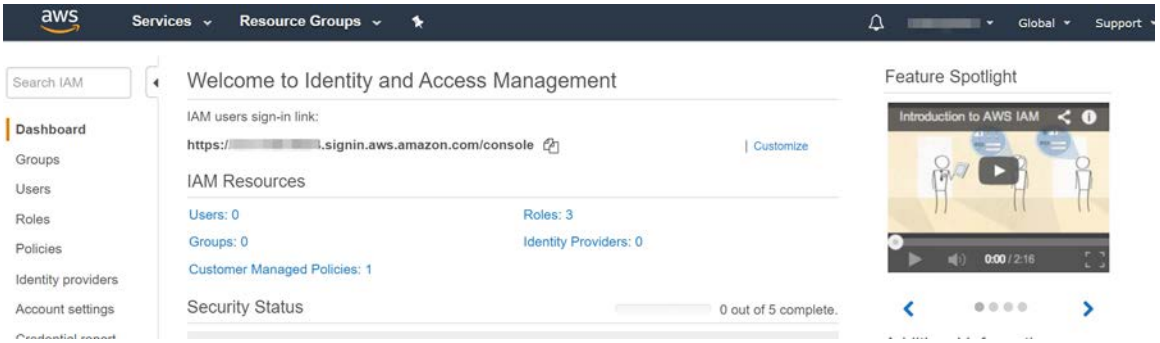


Figure 2.30: IAM dashboard

2. Now, once you get to the IAM dashboard, click on **Roles**:

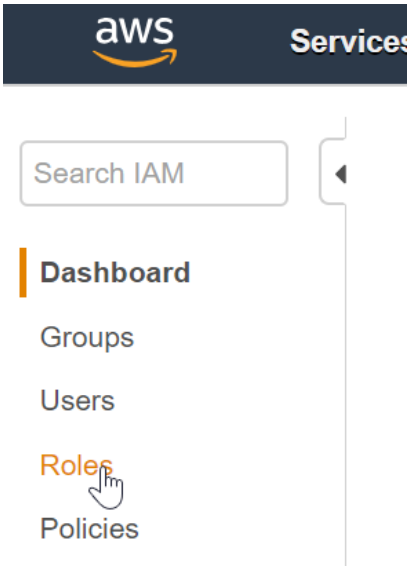


Figure 2.31: Left-hand side of the IAM dashboard

3. Now, the screen will be populated with the Role list. Click on **`s3TriggerRole`** in the Role list:

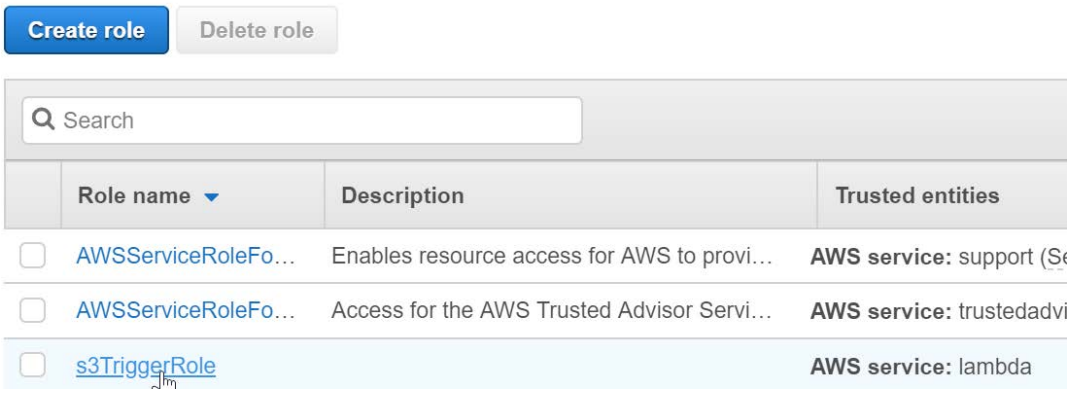


Figure 2.32: Role list Selecting s3TriggerRole

4. The option of **`s3TriggerRole`** will be enabled. Then, click on **Attach policies**:

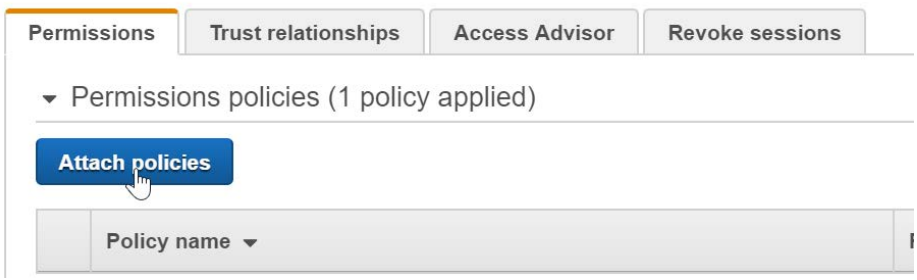


Figure 2.33: Permissions tab for s3TriggerRole

5. Type **`Comprehend`** to filter the policies. Then, click the checkbox next to **`ComprehendFullAccess`**:

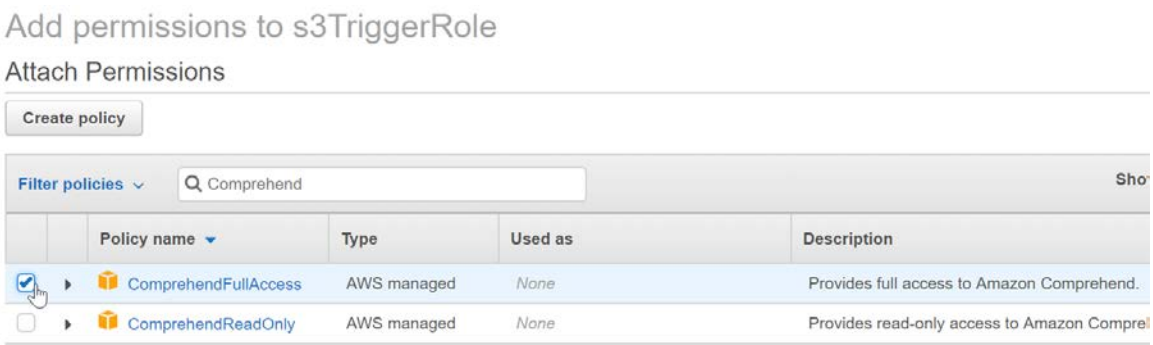


Figure 2.34: ComprehendFullAccess policy selection

6. Once you have selected the checkbox, click on **Attach policy** (located in the lower right-hand side of the screen):

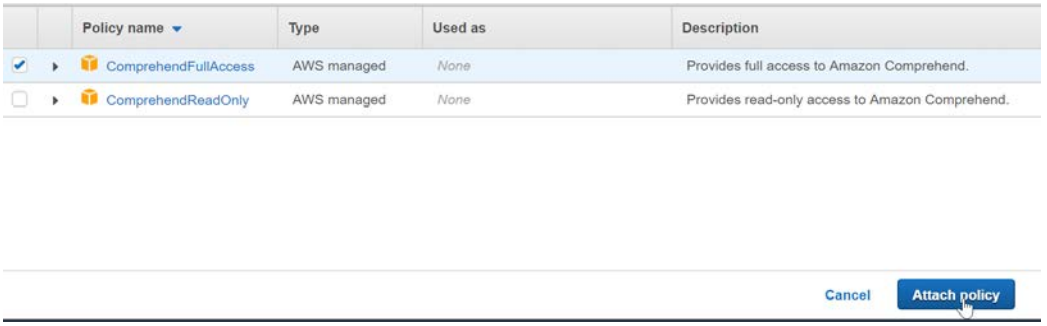


Figure 2.35: Attaching the selected policies

7. You will be redirected to the **`s3TriggerRole`** screen, and you will receive the following message:

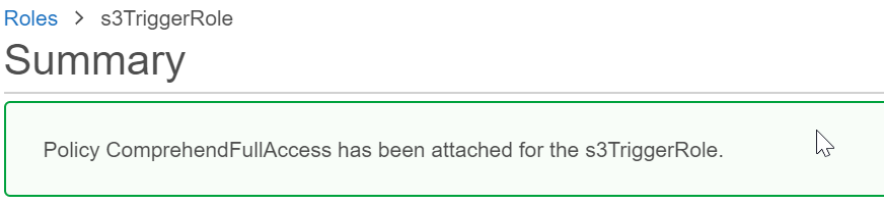


Figure 2.36: Successfully attached polices message

Activity 3: Integrating Lambda with Amazon Comprehend to Perform Text Analysis

In this activity, we will integrate the Lambda function with Comprehend to perform text analysis (**`detect_sentiment`**, **`detect_entities`**, and **`detect_key_phrases`**) when a document is uploaded to S3.

Suppose that you are an entrepreneur creating a chatbot. You have identified a business topic and the corresponding text documents, with content that will allow the chatbot to make your business successful. Your next step is to integrate the Lambda function with Comprehend, for sentiment, key phrases, and entities. To ensure that this happens correctly, you will need to have **`test_s3trigger_configured.txt`**. Before you execute the **`s3_trigger`**, consider the output, based on the following aspects of the text: sentiment (positive, negative, or neutral), entities (quantity, person, place, and so on), and key phrases:

1. First, navigate to the **`S3_trigger`** Lambda function.
2. Add **`test_s3trigger_configured.txt`** to the S3 Bucket, in order to verify the Lambda **`S3_trigger`** function.

3. Now, upload the file into the Bucket and monitor the file.
4. Next, click on View logs in the **CloudWatch** by using the log stream.
5. Now, expand the output in a text format.
6. The following will be the output:

   Sentiment_response -> Classified as 60.0% likely to be a Positive

   Sentiment_response:

```
{'Sentiment': 'POSITIVE','SentimentScore':{'Positive':
0.6005121469497681,'Negative': 0.029164031147956848, 'Neutral':
0.3588017225265503, 'Mixed': 0.01152205839753151},
```

   entity_response --> Classified as 70.5% likely to be a Quantity

   entity_response:

```
{Entities':[{'Score':0.7053232192993164, 'Type': 'QUANTITY','Text': '3
trigger', 'BeginOffset': 35, 'EndOffset': 44}],
```

   key_phases_response -> Classified as 89.9% likely "a test file" and 98.5% likely 'the s3 trigger" are the key phrases:

   key_phases_response:

```
 {'KeyPhrases': [{'Score': 0.8986637592315674, 'Text': 'a test file',
'BeginOffset': 8, 'EndOffset': 19}, {'Score': 0.9852105975151062, 'Text':
'the s3 trigger', 'BeginOffset': 30, 'EndOffset': 44}],
```

> **Note**
>
> To refer to the detailed steps, go to the *Appendix A* at the end of this book on Page no.198

Summary

In this chapter, you learned how Comprehend's **`DetectDominantLanguage`** method is structured, and how to pass in both strings and a list of strings. You learned how to extract entities, sentiments, key phrases, and topics, which provide the data for complex NLP processing. This allows Amazon Comprehend to become more efficient, by automating text analysis upon a text document that's been uploaded to S3.

Overall, the culmination of these independent functions provides the foundation for building complex machine learning-based NLP applications (for example, Siri, Alexa, and so on). Knowing how and why the individual functions operate will allow you to build your own AWS-based NLP applications.

In the next chapter, we will explore Topic Modeling and perform theme extraction.

3

Perform Topic Modeling and Theme Extraction

Learning Objectives

By the end of this chapter, you will be able to:

- Extract and analyze common themes through topic modeling with Amazon Comprehend
- Describe the basics of topic modeling analysis
- Perform topic modeling on a set of documents and analyze the results

This chapter describes Topic Modeling on common themes using Amazon Comprehend analyzing the result for document set.

Introduction

In the first part of this chapter, you will learn how to analyze Topic modeling output from Amazon Comprehend. Specifically, you will learn the fundamentals of the algorithm used for Topic modeling, Latent Dirichlet Allocation (LDA). Learning LDA will allow you to apply Topic modeling to a multitude of unique business use cases.

You will then perform Topic modeling on two documents with a known Topic structure. The first is the story **Romeo and Juliet** and the second is **War of the Worlds**. Lastly, you will analyze topics from 1,000 text documents containing negative movie reviews with Amazon Comprehend.

Extracting and Analyzing Common Themes

You can also utilize Amazon Comprehend to analyze a corpus of archives to locate the normal topics contained inside the corpus. Amazon Comprehend inspects reports in the corpus and, afterward, restores the most noticeable themes and the reports that are related to every subject. Subject displaying is an offbeat procedure: you present an arrangement of records for preparation and later get the outcomes when handling is finished. Amazon Comprehend performs point displaying on huge report sets. For the best results, you ought to incorporate around 1,000 records when you present a subject demonstrating work.

Topic Modeling with Latent Dirichlet Allocation (LDA

The subjects or **common themes** of a set of documents can be determined with Amazon Comprehend. For example, you have a movie review website with two message boards, and you want to determine which message board is discussing two newly released movies (one about sport and the other about a political Topic). You can provide the message board text data to Amazon Comprehend to discover the most prominent topics discussed on each message board.

The machine learning algorithm that Amazon Comprehend uses to perform Topic Modeling is called latent Dirichlet allocation (LDA). LDA is a learning-based model that's used to determine the most important topics in a collection of documents.

The way that LDA works is it considers every document to be a combination of topics, and each word in the document is associated to one of these topics.

For example, if the first paragraph of a document consists of words like **eat**, **chicken**, **restaurant**, and **cook** then you conclude that the Topic can be generalized to **Food**. And if the second paragraph of a document contains words like **ticket**, **train**, **kilometer**, and **vacation** then you can conclude that the Topic is **Travel**.

Basic LDA example

Topic modeling can seem complex, and understanding the fundamental steps of how LDA determines Topics is essential to performing Topic modeling on more complex business use cases. Thus, let's deconstruct LDA with the following simple example.

You have one document with five sentences. Your goal is to determine the two most common topics present in the document:

- I like to eat bread and bananas.
- I ate a bread and banana smoothie for breakfast.
- Puppies and kittens are cute.
- My brother adopted a puppy yesterday.
- Look at this cute opossum munching a piece of broccoli.

LDA discovers the topics that these sentences contain. For example, given the above sentences and asked for two topics, LDA might produce the following:

Sentences 1 and 2: 100% Topic A

Sentences 3 and 4: 100% Topic B

Sentence 5: 60% Topic A, 40% Topic B

Topic A: 30% bread, 15% banana, 10% breakfast, 10% munching, (Thus, you can assume Topic A is about food)

Topic B: 20% Puppies, 20% kittens, 20% cute, 15% opossum, (This, you can assume Topic B is about cute animals).

Why Use LDA?

LDA is useful when you have an arrangement of records that you need to find designs inside, without thinking about the reports themselves. LDA can be utilized to create subjects to comprehend an archives, general Topic. This is, usually utilized in suggestion frameworks, report arrangement, and record synopsis. In conclusion, LDA is helpful in preparing prescient models with subjects and events.

LDA has many use cases. For example, you have 30,000 user emails and want to determine the most common topics to provide group-specific recommended content based on the most prevalent topics. Manually reading, or even outsourcing the manual reading of, 30,000 emails, would take an excessive investment in terms of time and money, and the accuracy would be difficult to confirm. However, Amazon Comprehend can seamlessly provide the most common topics present in 30,000 emails in a few steps with incredible accuracy. First, convert the emails to text files, upload them to an S3 bucket, then imitate a Topic modeling job with Amazon Comprehend. The output is two `CSV` with the corresponding Topics and terms.

Amazon Comprehend–Topic Modeling Guidelines

The most accurate results are given if you provide Comprehend with the largest possible corpus. More specifically:

- You should use no less than 1,000 records in every subject demonstrating work
- Each report ought to be something like three sentences in length
- If a record comprises of for the most part numeric information, you should expel it from the corpus

Currently, Topic Modeling is limited to two document languages: **English** and **Spanish**.

A Topic modeling job allows two format types for input data (see the following table 1). This allows users to process both collections of large documents (for example, newspaper articles or scientific journals), and short documents (for example, tweets or social media posts).

Input Format Options:

Format	Description
One document per file	Each file contains one input document. This is best for collections of large documents.
One document per line	The input is a single file. Each line in the file is considered a document. This is best for short documents, such as social media postings.

Figure3.1: AWS Comprehend– Topic modeling input format options

Output Format Options:

Files	Description
topic-terms.csv	List of topics in the collection. For each topic, the list includes, by default, the top terms by topic according to their weight.
doc-topics.csv	Lists the documents associated with a topic and the proportion of the document that is concerned with the topic. If you specified ONE_DOC_PER_FILE the document is identified by the file name

Figure 3.2: AWS Comprehend– Topic modeling output files description

After Amazon Comprehend processes your document collection, the modeling outputs two CSV files: Topic-terms.csv (see Figure 1) and **`doc-topics.csv`**.

The **`topic-terms.csv`** file provides a list of topics in the document collection with the terms, respective Topic and weight. For example, if you gave Amazon Comprehend two hypothetical documents, **learning to garden** and **investment strategies**, it might return the following to describe the two topics in the collection:

Topic	Term	Weight
0	learn	0.22
0	garden	0.21
0	soil	0.1
0	dig	0.09
0	shovel	0.08
0	season	0.07
0	grow	0.06
0	sun	0.01
0	seeds	0.009
0	growth	0.0008
1	money	0.28
1	stock	0.19
1	returns	0.1
1	risk	0.07
1	hedge	0.06
1	algorithmic	0.05
1	strategy	0.03
1	loss	0.01
1	software	0.009
1	profit	0.008

Figure 3.3: Sample Topic modeling output (`topic-terms.csv`) for two document's input

The **doc-topics.csv** file provides a list of the documents provided for the Topic modeling job with the document names, and the respective topics and their proportions in each document. Given two hypothetical documents, **learning_to_garden.txt** and **investment_strategies.txt,** you can expect the following output:

Docname	Topic	Proportion
learning_to_garden.txt	0	1
investment_strategies.txt	1	1

Figure 3.4: Sample Topic modeling output (**doc-topics.csv**) for two document's input

Exercise 13: Topic Modeling of a Known Topic Structure

In this exercise, we will use Amazon Comprehend to perform Topic modeling on two documents with known topics (**Romeo and Juliet** and **War of the Worlds**). We are using two known topics to better understand LDA. Before proceeding to the exercise, just look at an overview of the data pipeline architecture:

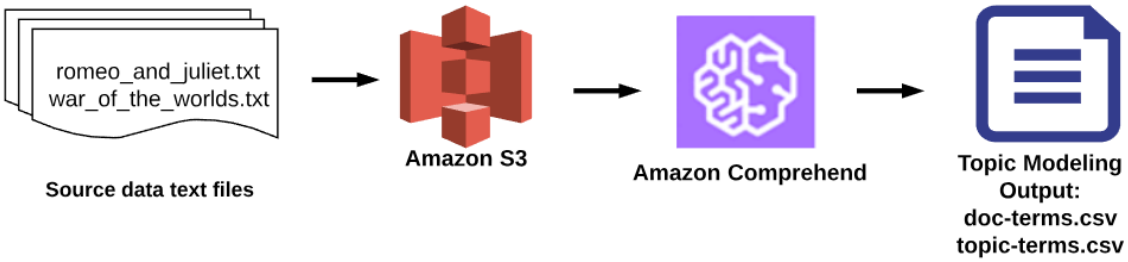


Figure 3.5: Data pipeline architecture overview

The following are the steps to complete the Topic modelling of a known Topic structure:

1. We need an input and output S3 bucket. Let's create both. Navigate to https://s3.console.aws.amazon.com/s3/.
2. Now, click on the **Create bucket** button to create a bucket:

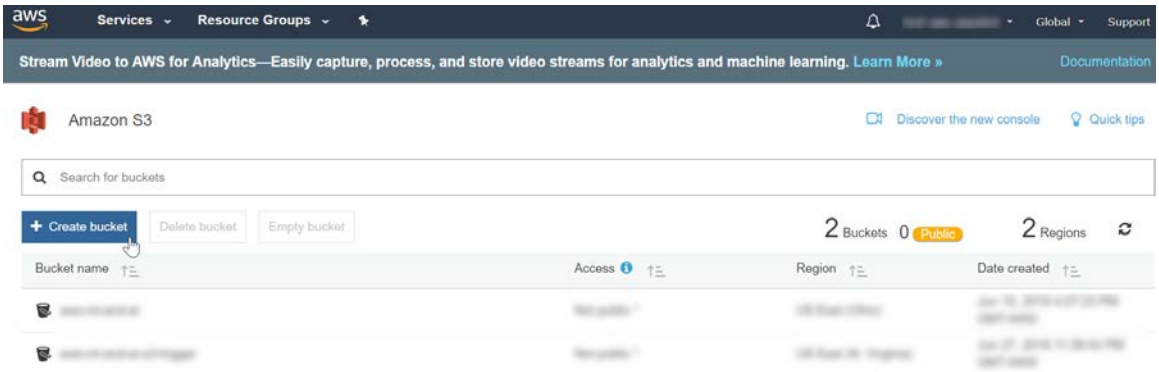


Figure 3.6: Creating a bucket

3. For Bucket name, enter a unique name that describes the function. Here, the name **`aws-ml-input-for-topic-modeling`** is used. Click on the **Create** button:

> **Note**
>
> Clicking Create versus Next uses all default settings for: properties and permissions.

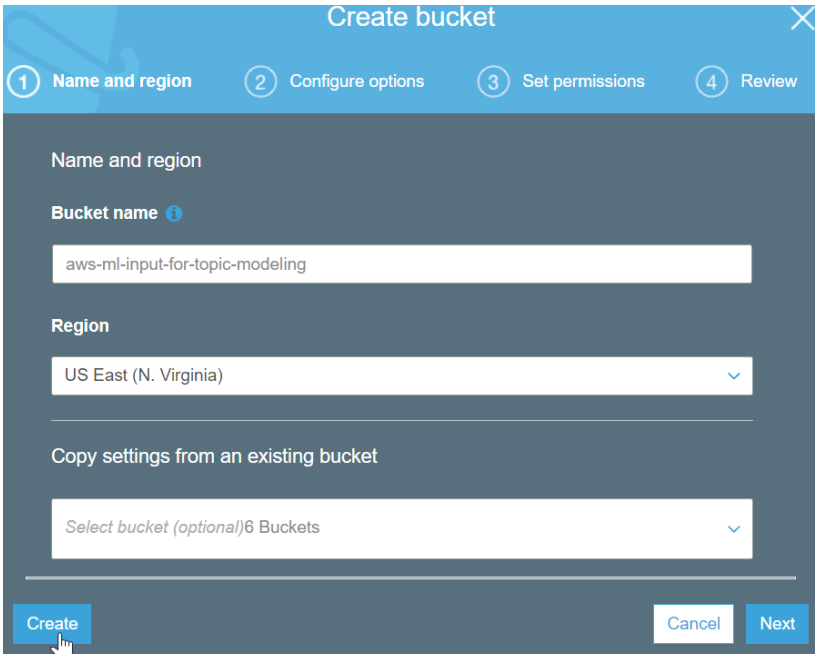


Figure 3.7: Creating bucket name input

4. Now, click on the **Create** button to create a folder:

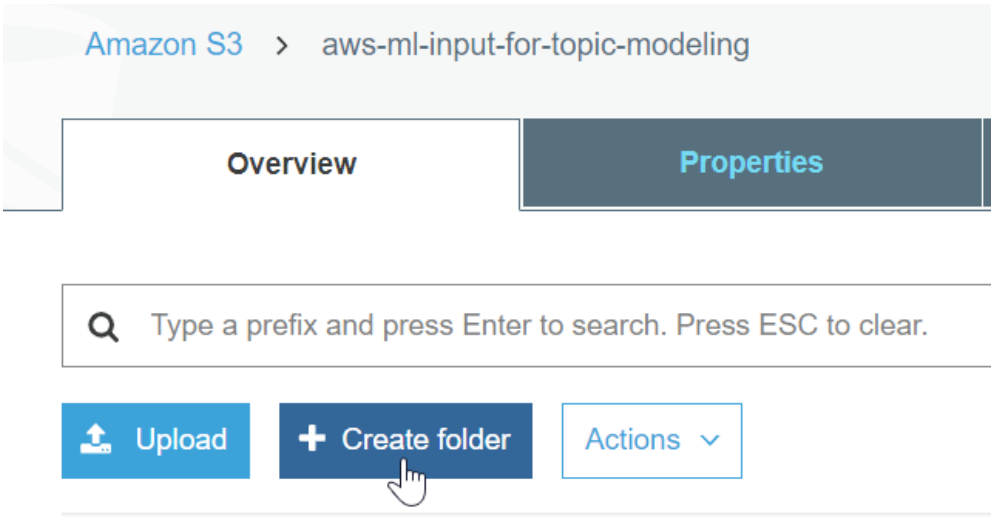


Figure 3.8: Creating a folder in S3 for Topic modeling input

5. Now, type in **known_structure**, as the folder name, and then click on the **Save** button:

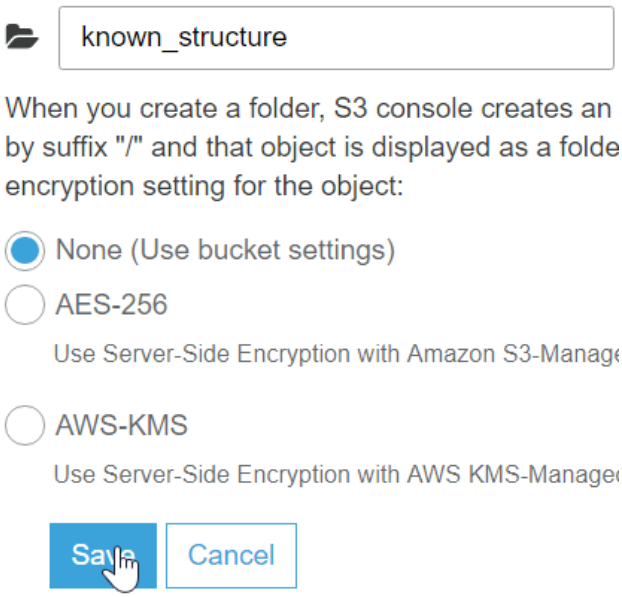


Figure 3.9: Saving the '**known_structure**' folder name

6. After clicking on the **Save** button, your folder will be generated. Now, click on the **known_structure** folder:

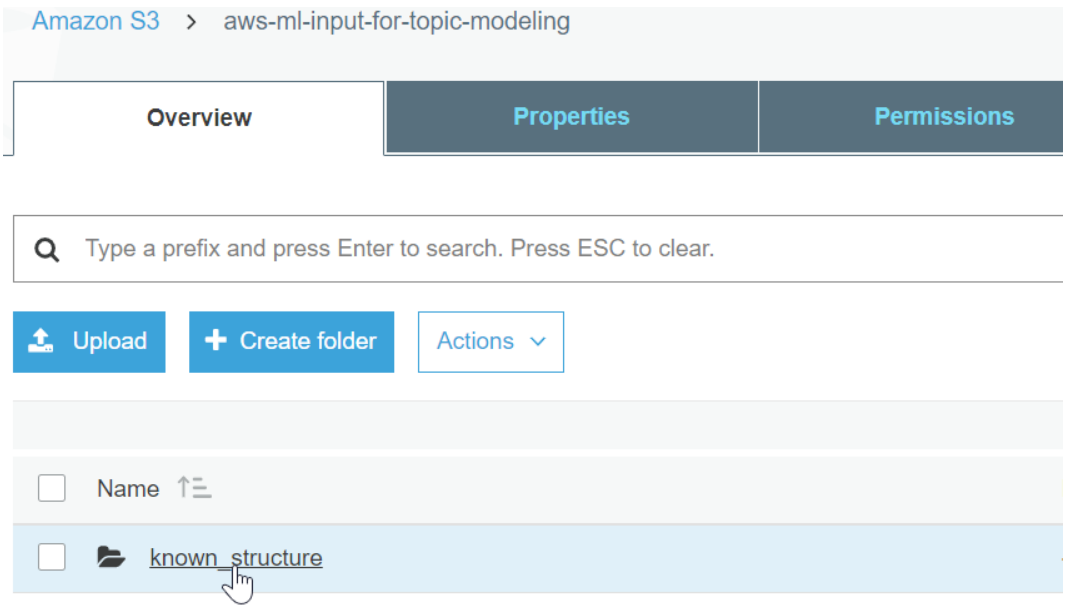


Figure 3.9: Input bucket screen

7. Now, click on the **Upload** button:

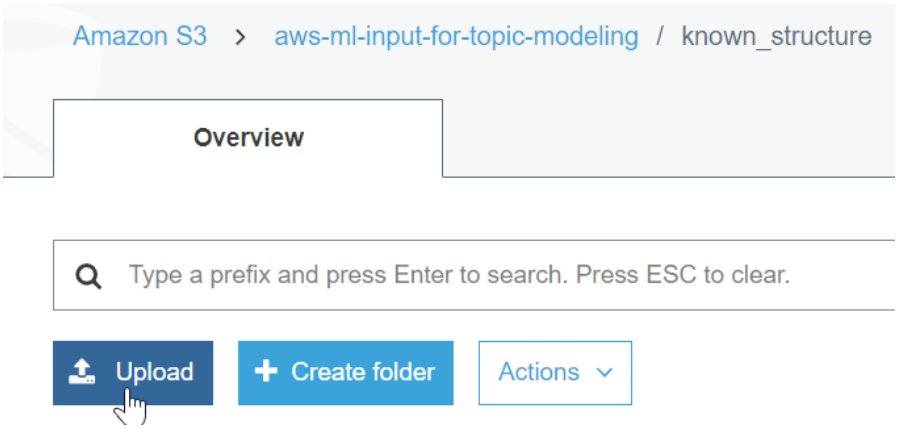


Figure 3.10: Upload screen

8. Now, you will prompted by the following for adding files. Click on **Add files** or drag the files onto the screen:

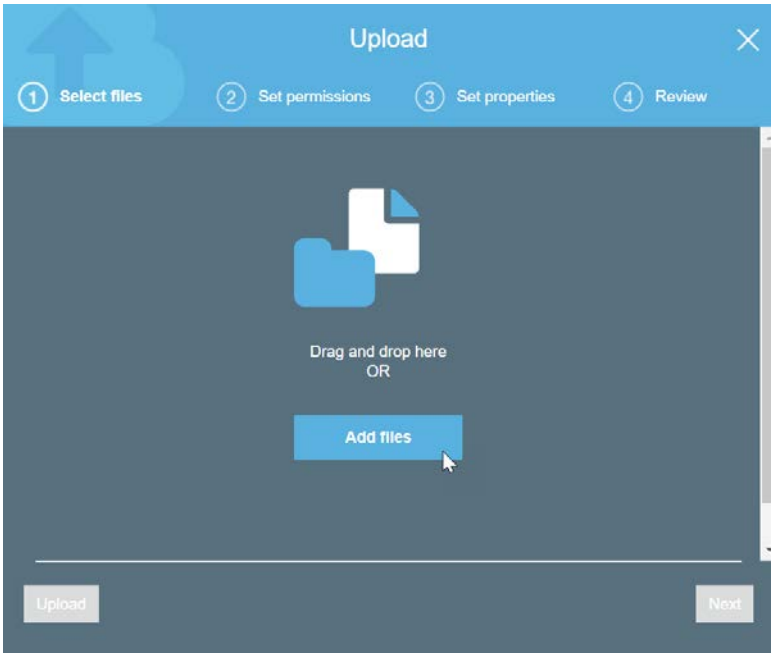


Figure 3.11 Add files screen

9. Navigate to download and upload the following two text files from the machine:

> **Note**
>
> You can download the Romeo and Juliet text file from /lesson3/topic_a/romeo_and_juliet.txt https://github.com/TrainingByPackt/Machine-Learning-with-AWS/blob/master/lesson3/topic_a/romeo_and_juliet.txt /lesson3/topic_a/the_war_of_the_worlds.txt https://github.com/TrainingByPackt/Machine-Learning-with-AWS/blob/master/lesson3/topic_a/the_war_of_the_worlds.txt

10. Once the files have been uploaded, click on the **Upload** button to upload the files:

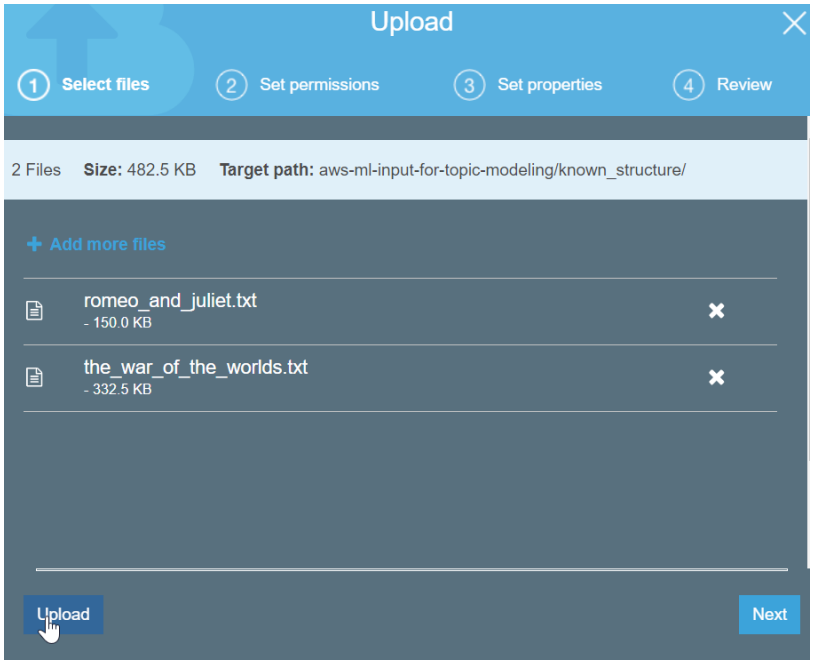


Figure 3.12: Uploading for the two `known_structure` text files

11. Navigate to the Amazon S3 homescreen:



Figure 3.13: Amazon S3

12. Next, create an output S3 bucket. Use the same S3 bucket creation process. Click on the **Create bucket** button:



Figure 3.14: Creating a bucket

13. Now, name the bucket, and then click on the **Create** button:

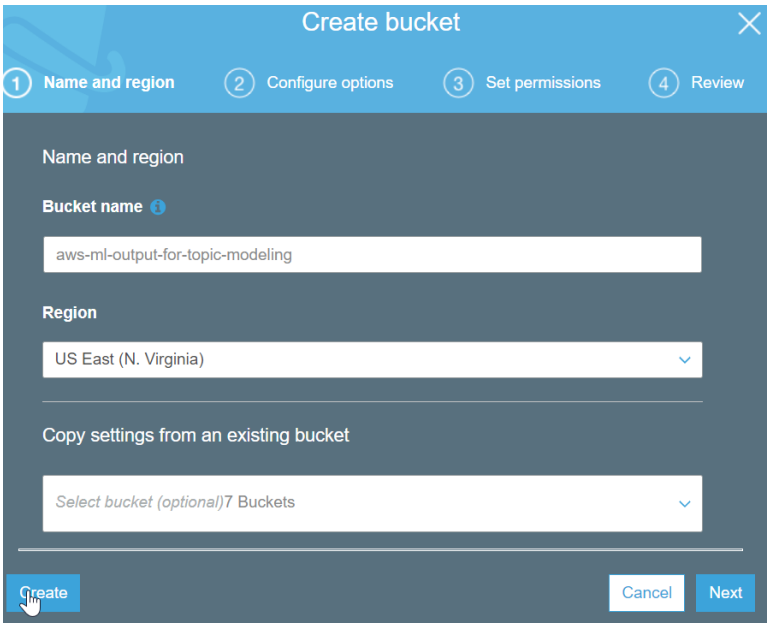


Figure 3.15: Create bucket output for Topic modeling

14. Navigate to Amazon Comprehend: https://console.aws.amazon.com/comprehend/. If you are presented with the following screen, click **Try Amazon Comprehend**:

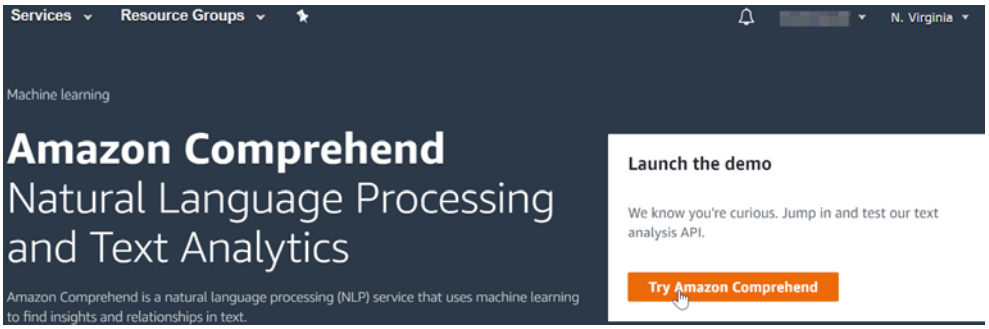


Figure 3.16: Amazon Comprehend home screen

15. Now, Click on the **Organization** in the left-hand side toolbar:

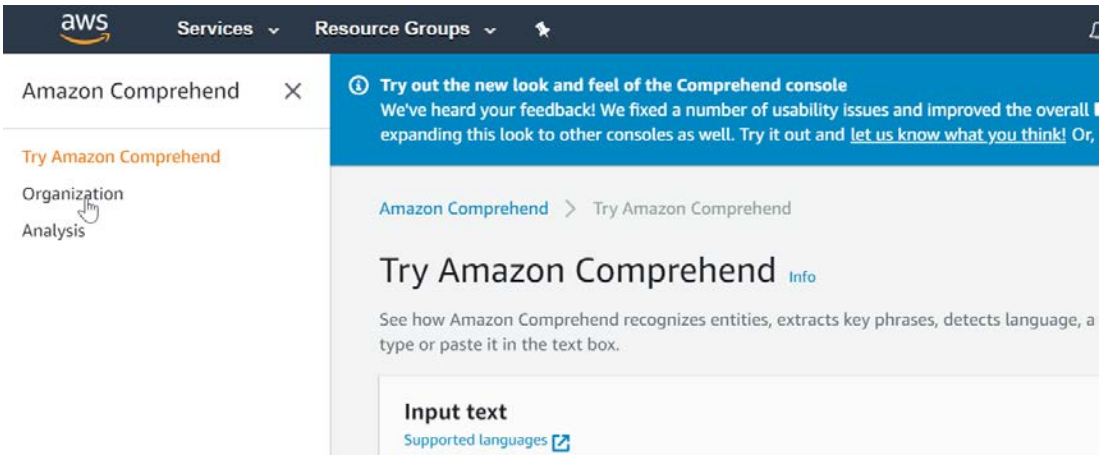


Figure 3.17: Amazon Comprehend Organization screen

16. Now, click on the **Create** button and then enter **`known_structure_topic_modeling_job`** in the **Name** field:

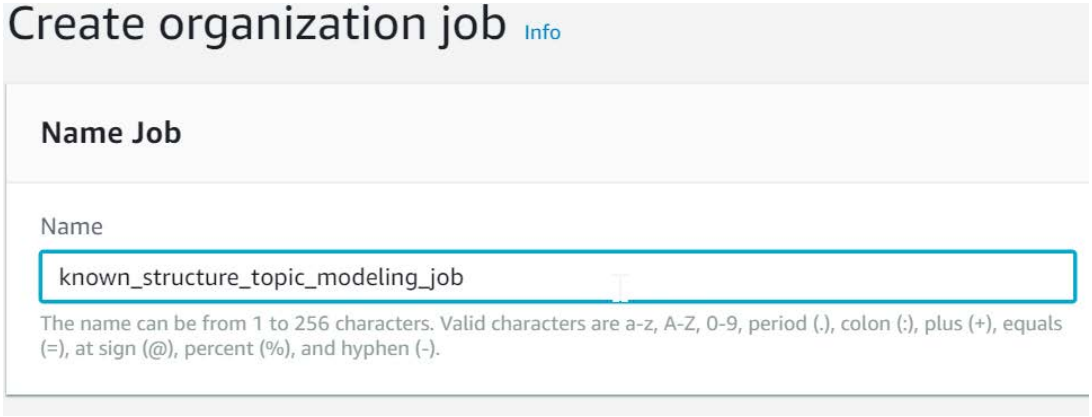


Figure 3.18: Name of the Topic modeling job

17. Now, scroll down to **Choose input data** and then click on **Search**:

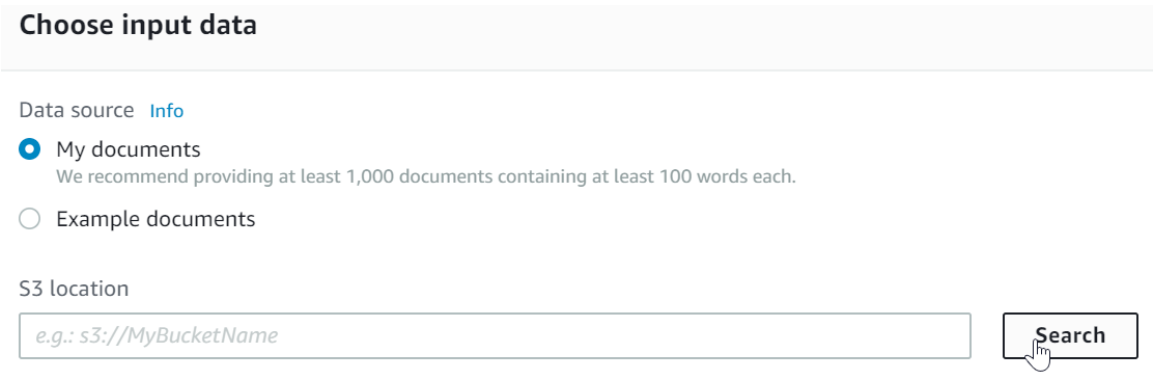


Figure 3.19: Clicking Search to locate the Topic modeling input data source

18. Navigate to the **`known_structure`** folder and then click on **Select**:

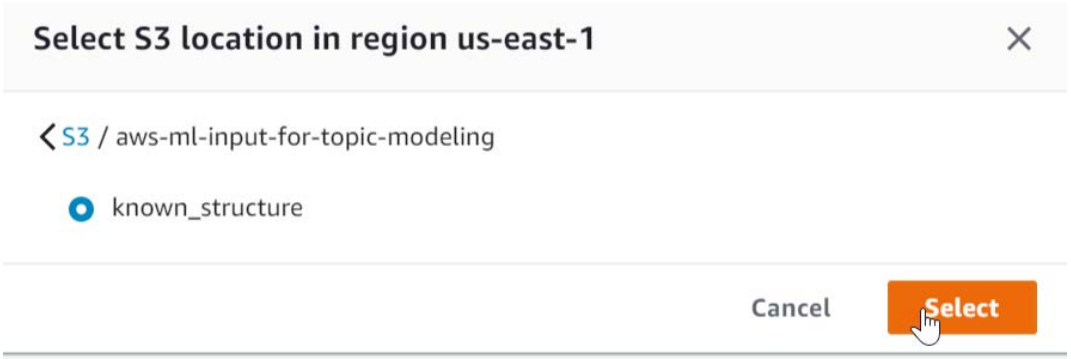


Figure 3.20: Clicking on Select for the S3 folder

19. Now, from the drop-down menu, select **One document per file**:

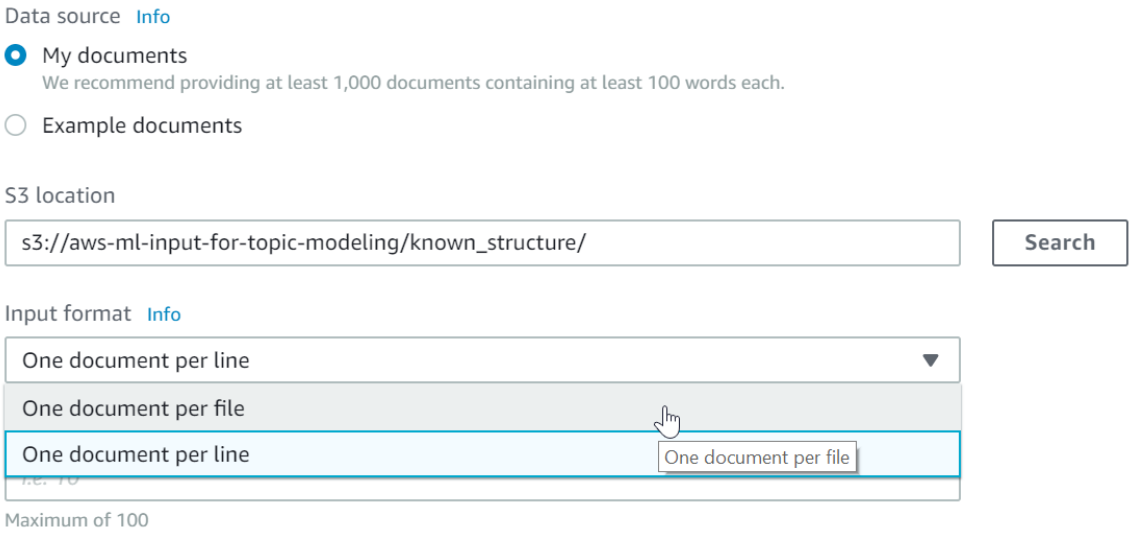


Figure 3.21: Selecting One document per file

20. Now, enter **two** for the **Number of Topics** you need to have:



Figure 3.22: Entering 2 for the number of topics to perform Topic modeling

21. Next, click on **Search** to search the bucket that was created previously:

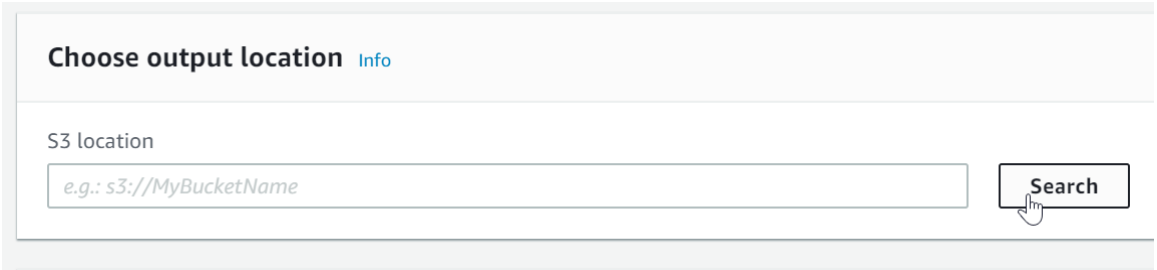


Figure 3.23: Clicking on search for the Topic modeling S3 output location

22. Once you find the bucket you created, click on the bucket you created to output Topic modeling:

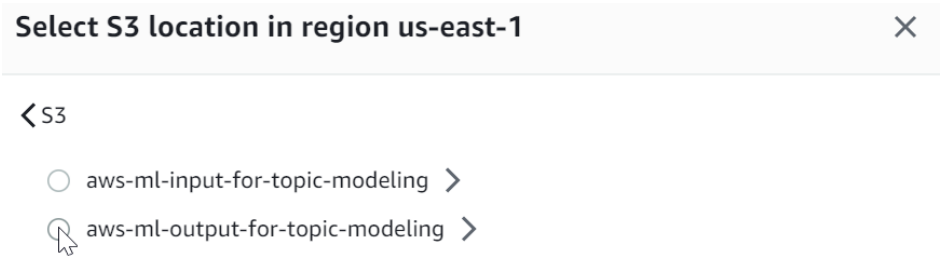


Figure 3.24: Selecting the output S3 bucket

23. Now, select the appropriate bucket and then click on **Select**:

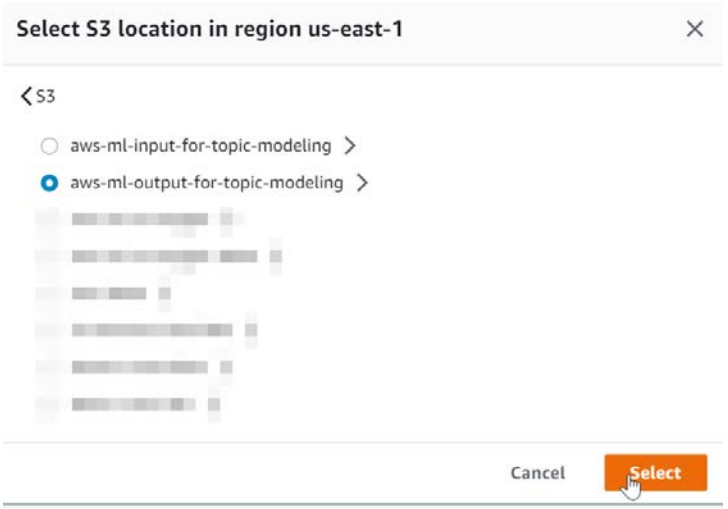


Figure 3.25: Confirming by clicking Select

24. Scroll down to choose an **IAM** role, and click the circle next to create an **IAM** role:

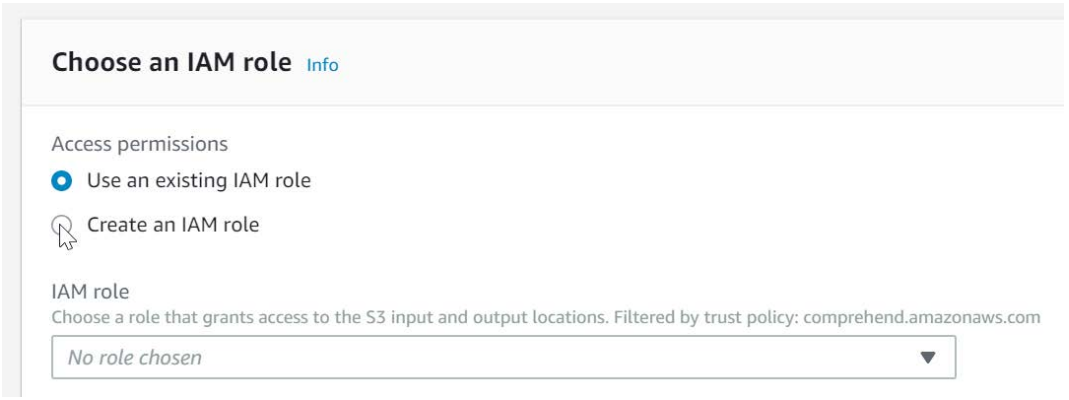


Figure 3.26: Selecting Create an IAM role

25. Now, select the **Input** and **Output S3 buckets** from the **Permissions to access**:



Figure 3.27: Providing permission to Input and Output S3 buckets

26. Enter **`myTopicModelingRole`** in the Name suffix field and then click on the **Create job** button:

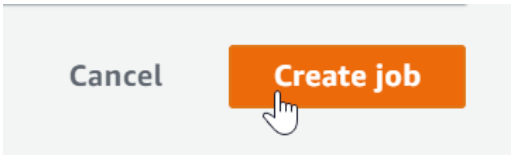


Figure 3.28: Clicking the Create job button

27. Creating the job may take a few minutes, but when completed, you will be redirected to the Comprehend home screen:

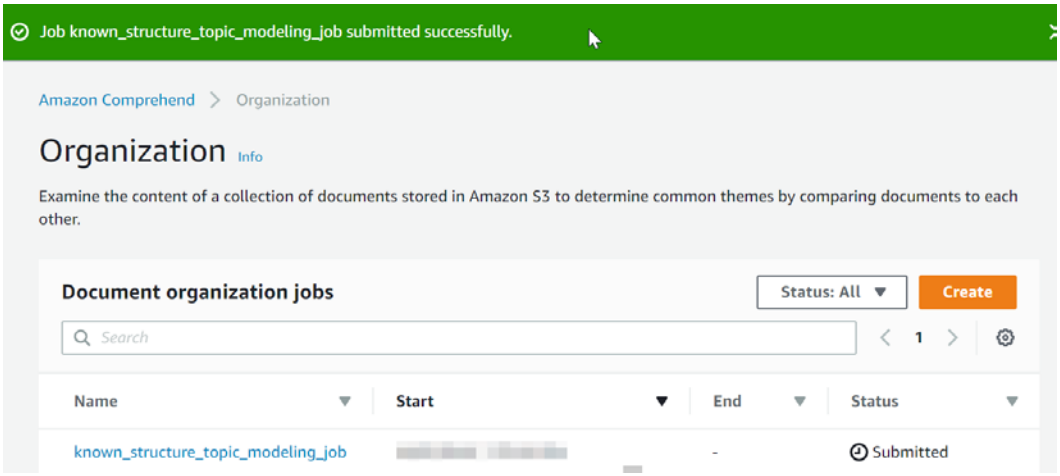


Figure 3.29: Comprehend home screen

28. While the job is being processed, the status displayed will be **In Progress**:

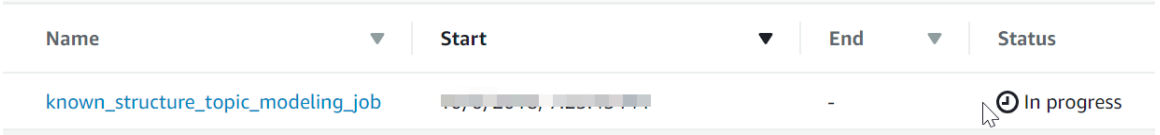


Figure 3.30: In progress status displayed

29. When the status updates to **Completed**, click on the Topic Modeling job name:

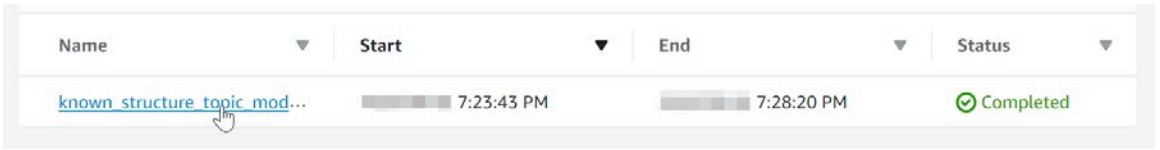


Figure3.31: Completed status displayed

30. Now, scroll down to the **Output** section:

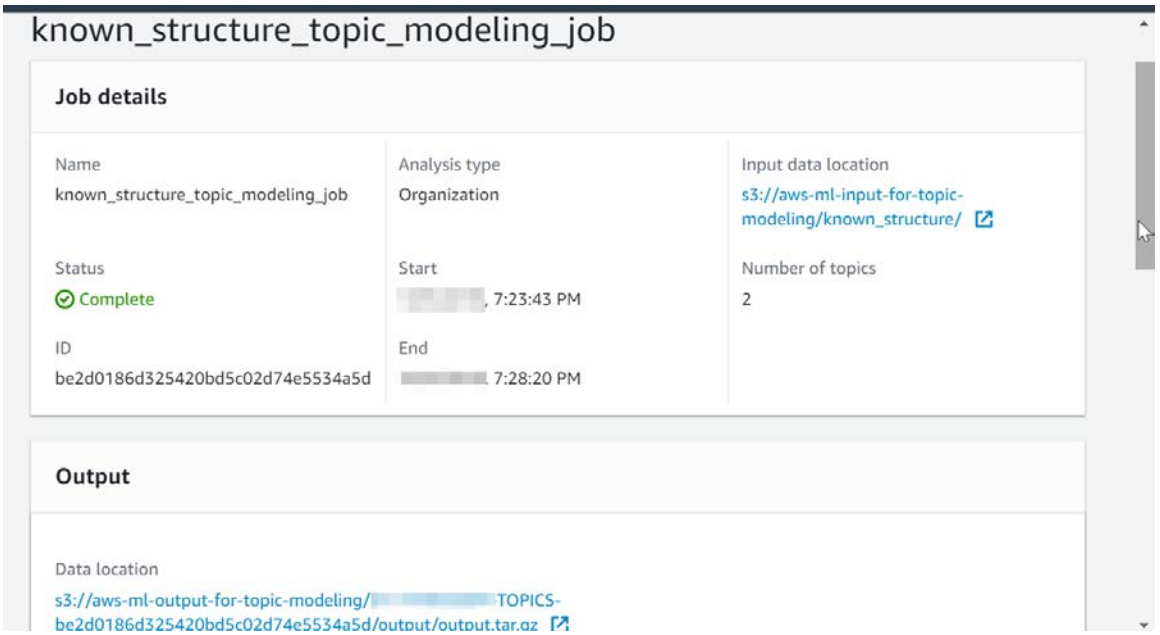


Figure 3.32: Topic modeling output display home screen

31. Click on the hyperlink under **Data location**:



Figure 3.33: Topic modeling data output hyperlinked location

32. Click on the link of the output folder:

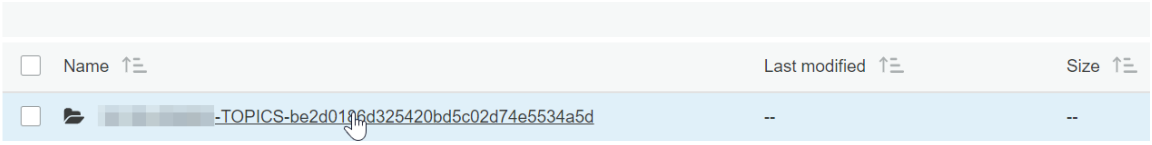


Figure 34: Topic modeling output folder

33. Click on the output folder. Then, click on **`output.tar.gz`** and download the file:



Figure 3.35: Clicking on Download

34. Click on **`output.tar.gz`** and select **Show in folder**. Click on **OK** to extract the files on your desktop:

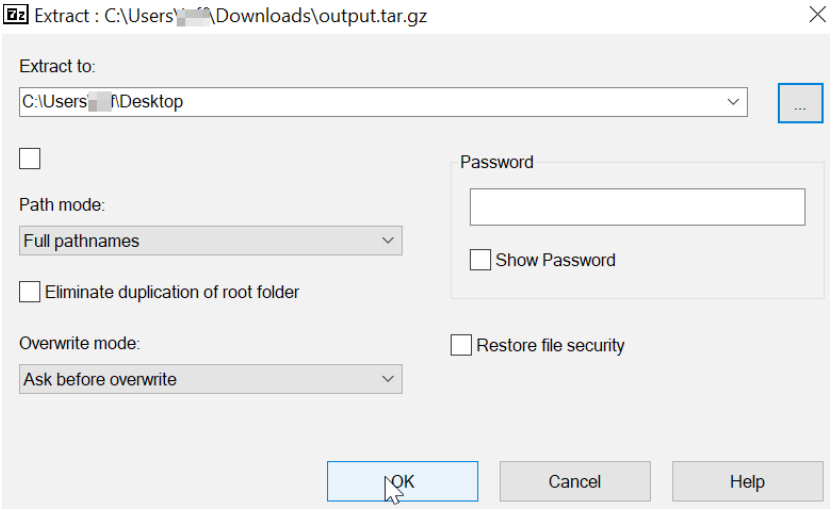


Figure 3.36: Clicking on OK

35. Navigate to your desktop. Two files will be extracted: **`doc-topics.csv`** and **`topics-terms.csv`**. There will be two files to examine: **`topic-terms.xlsx`** and **`doc-topics.xlsx`**:

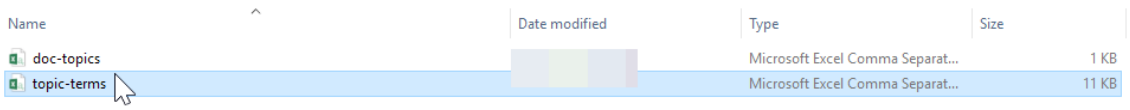


Figure3.37: Topic modeling output CSV files

> **Note**
>
> Your Topic-terms.csv and doc-topics.csv results should be the same as the following results. If your results are NOT the same, use the output files for the remainder of the chapter, which are located at Lesson3\topic_a\doc-topics.csv https://github.com/TrainingByPackt/Machine-Learning-with-AWS/blob/master/lesson3/topic_a/doc-topics.csv and lesson3\topic_a\topic-terms.csv https://github.com/TrainingByPackt/Machine-Learning-with-AWS/blob/master/lesson3/topic_a/topic-terms.csv.

The following is the output generated:

	A	B	C
1	topic	term	weight
2	0	martian	0.012308
3	0	people	0.007995
4	0	house	0.005948
5	0	black	0.005939
6	0	timar	0.005855
7	0	road	0.005638
8	0	thing	0.0054
9	0	stand	0.005232
10	0	pit	0.00488
11	0	brother	0.004484
12	1	thou	0.452083
13	1	thy	0.135417
14	1	rom	0.13217
15	1	nurse	0.090223
16	1	love	0.089694
17	1	you	0.051807
18	1	romeo	0.048605
19			
20			
21			

topic-terms

Figure 3.38: `topic-terms.csv` result

	A	B	C
1	docname	topic	proportion
2	the_war_of_the_worlds.txt	0	1
3	romeo_and_juliet.txt	1	1

Figure 3.39: doc-topics.csv results

Exercise 14: Performing Known Structure Analysis

In this exercise, we will programmatically upload the CSV (**doc-topics.csv** and **Topic-terms.csv**) to S3, merge the CSV on the Topic column, and print the output to the console. The following are the steps for performing Known Structure Analysis:

> **Note**
>
> For this step, you may either follow along with the exercise and type in the code or obtain it from the source code folder, **local_csv_to_s3_for_analysis.py**, and paste it into the editor. The source code is available on GitHub in the following repository: https://github.com/TrainingByPackt/Machine-Learning-with-AWS/blob/master/lesson3/topic_a/local_csv_to_s3_for_analysis.py.

1. First, we will import **boto3** using the following command:

```
import boto3
```

2. Next, we will import **pandas** using the following command:

```
import pandas as pd
```

3. Now, we will create the S3 client object using the following command:

```
s3 = boto3.client('s3')
```

4. Next, we will create a variable with a unique bucket name. Here, the selected bucket name is **known-tm-analysis**, but you will need to create a unique name:

```
bucket_name = '<insert a unique bucket name>' #
```

5. Next, create a new bucket:

```
s3.create_bucket(Bucket=bucket_name)
```

6. Create a list of the CSV filenames to import:

```
filenames_list = ['doc-topics.csv', 'topic-terms.csv']
```

7. Now, iterate on each file to upload to S3 using the following line of code:

```
for filename in filenames_list:
    s3.upload_file(filename, bucket_name, filename)
```

8. Next, check if the filename is **doc-topics.csv** :

```
    if filename == 'doc-topics.csv':
```

9. Now, get the **doc-topics.csv** file object and assign it to the **obj** variable using the following command:

```
obj = s3.get_object(Bucket=bucket_name, Key=filename)
```

10. Next, read the **csv** obj and assign it to the **doc_topics** variable:

```
        doc_topics = pd.read_csv(obj['Body'])
    else:
        obj = s3.get_object(Bucket=bucket_name, Key=filename)
        topic_terms = pd.read_csv(obj['Body'])
```

11. Now, merge the files on the Topic column to obtain the most common terms per document using the following command:

```
merged_df = pd.merge(doc_topics, topic_terms, on='topic')
Print the merged_df to the console
print(merged_df)
```

12. Next, navigate to the location of the **CSV** files in the Command Prompt, and execute the code with the following command:

```
python local_csv_to_s3.py
```

13. The console output is a merged **dataframe** that provides the **docnames** with their respective terms and the term's weights (see the following):

	docname	topic	proportion	term	weight
0	the_war_of_the_worlds.txt	0	1.0	martian	0.012308
1	the_war_of_the_worlds.txt	0	1.0	people	0.007995
2	the_war_of_the_worlds.txt	0	1.0	house	0.005948
3	the_war_of_the_worlds.txt	0	1.0	black	0.005939
4	the_war_of_the_worlds.txt	0	1.0	timar	0.005855
5	the_war_of_the_worlds.txt	0	1.0	road	0.005638
6	the_war_of_the_worlds.txt	0	1.0	thing	0.005400
7	the_war_of_the_worlds.txt	0	1.0	stand	0.005232
8	the_war_of_the_worlds.txt	0	1.0	pit	0.004880
9	the_war_of_the_worlds.txt	0	1.0	brother	0.004484
10	romeo_and_juliet.txt	1	1.0	thou	0.452083
11	romeo_and_juliet.txt	1	1.0	thy	0.135417
12	romeo_and_juliet.txt	1	1.0	rom	0.132170
13	romeo_and_juliet.txt	1	1.0	nurse	0.090223
14	romeo_and_juliet.txt	1	1.0	love	0.089694
15	romeo_and_juliet.txt	1	1.0	you	0.051807
16	romeo_and_juliet.txt	1	1.0	romeo	0.048605

Figure 3.40: known_strucutre Topic modeling merged results

14. To verify the CSV's, navigate to S3, (reload the page if the new bucket does not appear), and the new bucket has been created in S3. Click on the bucket to verify a successful import:

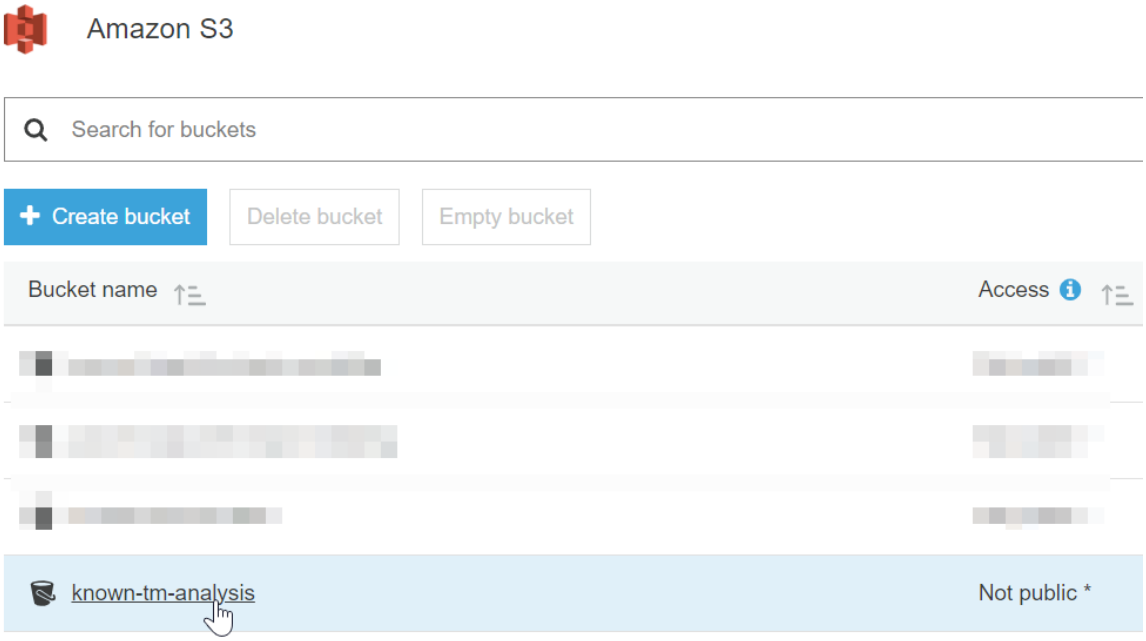


Figure 3.41: known-tm-analysis S3 bucket

15. There will be two CSV files in the bucket: **`doc-topics.csv`** and **`topic-terms.csv`**:

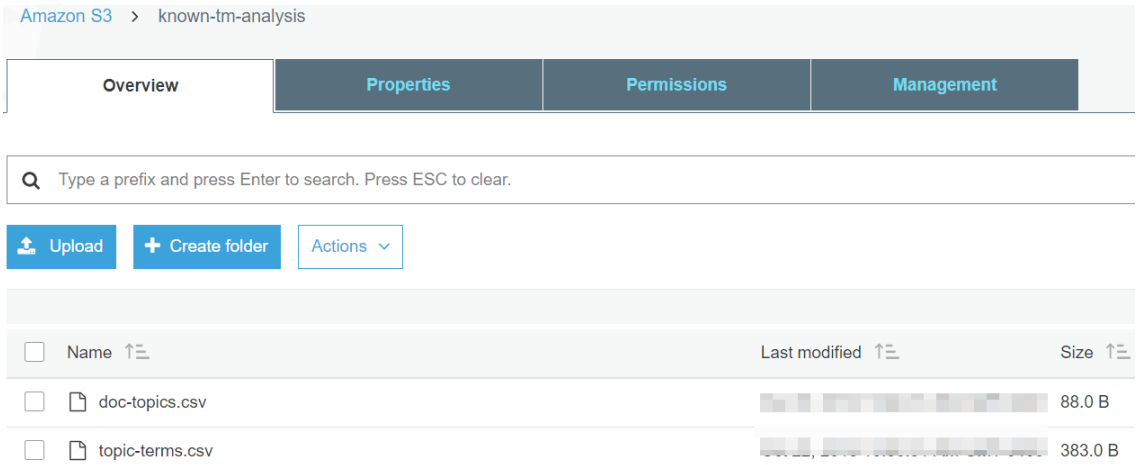


Figure 3.42: Topic modeling results uploaded to S3

Activity 4: Perform Topic Modeling on a Set of Documents with Unknown Topics

In this activity, we will perform Topic modeling on a set of documents with unknown topics. Topic modeling, we will consider an example. Suppose your employer wants you to build a data pipeline to analyze negative movie reviews that are in individual text files with a unique ID filename. Thus, you need to perform Topic modeling to determine which files represent the respective topics. Overall, negative reviews provide more monetary benefit or loss to the company, thus, they are prioritizing negative reviews versus positive reviews. The company's end goal is to incorporate the data into a feedback chatbot application. To ensure that this happens correctly, you need have a file that contains negative comments. The expected outcome for this activity will be the Topic modeling results from the negative movie review files.

Performing Topic Modeling

1. Navigate to the following link to obtain the text data file that contains negative review comments: https://github.com/TrainingByPackt/Machine-Learning-with-AWS/blob/master/lesson3/activity/localfoldernegative_movie_review_files/cv000_29416.txt.
2. Create a bucket for the Topic modelling with a unique name.
3. Create a folder for the Topic modelling.
4. Import the OS and Boto3. Mention your unique bucket name.
5. Gather all of the working directories of the local path and make them into text files.
6. Create a list for all of the text files.
7. Iterate the files and upload them to S3.
8. Create a job on Organization using Amazon Comprehend.
9. As per the requirements, choose the input data. It may be **My document** or **Example document**.
10. Choose the file from the data source.
11. Apply the input format.
12. Provide the number of topics to perform the modeling.
13. Choose an IAM role and create a job.
14. Download the output file and extract the file.
15. The generated output will include the two `.CSV` files.

Analysis of Unknown Topics

1. Import **`Boto3`** and **`pandas`**.
2. Create the S3 client.
3. Create a new bucket with a unique name.
4. Create a list of CSV filenames to import.
5. Check the filename and assign it to the **obj** variable.
6. Read the **obj** variable.
7. Merge the files on the Topic column.
8. Print the merged files to the console.

> **Note**
>
> To refer to the detailed steps, go to the *Appendix A* at the end of this book on Page no.203

Summary

In this chapter, we learned about analyzing Topic modeling results from AWS Comprehend. You are now able to incorporate S3 to store data and can use it to perform analysis. In addition, we learned how to analyze documents that we know the topics of before performing Topic modeling, and those documents where the Topic is known. The latter requires additional analysis to determine the relevant topics.

In the next chapter, we will dive into the concept of chatbots and their understanding by using Natural Processing Language.

4

Creating a Chatbot with Natural Language

Learning Objectives

By the end of this chapter, you will able to:

- Define the basics of chatbots and chatbot design
- Set up the Amazon Lex service
- Create a custom chatbot to look up different share prices

This chapter describes the designing of chatbot using Amazon Lex.

Introduction

In the last chapter, you learned how to extract and analyze common themes through topic modeling with **Amazon Comprehend**.

In this chapter, you will learn how to build a chatbot using **Amazon Lex**. First, we'll talk about how to design a chatbot. Then, we will dive into exploring the Amazon Lex service by creating a sample chatbot.

Next, we will create our own custom chatbot, which will query an for Pizza ordering. Finally, you will integrate, your chatbot with a text interface in order to interact with it.

What is a Chatbot?

A **chatbot** is an application that simulates intelligent conversations using rules and an AI inference. When interacting with the chatbot, the goal is to be able to hold a conversation with the user to the extent required in order to resolve customer queries or suggest a way to move forward from them.

As in normal conversation, the means by which we interact with the Bot can be written text or speech. Often, the chatbots are integrated with messaging platforms, such as Slack, Facebook, Kik, WeChat, and so on. This can also be integrated with a custom web or mobile interface.

It is easier, of course, to integrate within an existing messaging platform, since the user is likely to be familiar with the interface. Moreover, these platforms provide support to the chatbot developers with infrastructure and development tools.

Some examples of chatbots include systems for ordering products, reporting, internal communication, and scheduling.

The Business Case for Chatbots

Traditionally, we used to interact with computers by means of input devices, such as keyboards and mice. Today, computational power and efficiency has progressed to the point where we can have a conversation with a computer almost as naturally as interacting with other people. These conversations seem more human-like, due to the ability of the computer to add contextual information and keep track of and remember the context for a few separate interactions.

It is no surprise, then, that the conversational interface is really catching on. One study by Gartner estimates that 30% of browsing done by users will turn out to be screen less in 2020. This is a huge number, considering all of the interactions between humans and computers.

Another study done by BI Intelligence found that the number of global monthly active users for the top four messaging apps surpassed those for the top four social networks in the first quarter of 2015 and shows no signs of slowing down.

Businesses cannot afford to ignore this trend, and are looking to reach users where they are found interacting (via text) the most: the messaging platforms such as Facebook, Skype, Slack, WhatsApp, and WeChat, and the list goes on and on. Messaging platforms make it easier for users to communicate, and chatbots make it easier for users to communicate what they want and to get it more quickly than interacting with other people.

In this topic, you will learn about **Natural Language Understanding (NLU)**. Using this knowledge, you will first explore the AWS Lex service and build your first sample chatbot. As the next step, you will build a custom chatbot.

What is Natural Language Understanding?

NLP is the general term for a set of technologies that deal with natural language. NLU is a focused subset of NLP that deals with actual conversational input.

NLU is able to handle unstructured inputs and convert to a structured, machine understandable form. Words that the user enters are transformed into intents and entities, or Slots. The NLU chatbot is further able to infer intents and Slots from user input, which may be similar to – but not the same as – the examples it has been trained with.

Core Concepts in a Nutshell

Before we can get started with building chatbots, you will need to understand some concepts first. We will now take a look at the technical meaning of the term chatbot and the names of the pieces which make up a chatbot and work together to deliver a conversational experience to the user.

Chatbot

A chatbot, also known as a **`bot`**, is a piece of software that can converse using natural language with the user. The goal is for the user to believe that they can interact freely and naturally with the bot, almost as if speaking with another person.

Utterances

Things that the user says to the bot are called **utterances**. The bot regards the utterances from the user as input, and is able to parse them into machine-recognizable formats. Some examples of utterances are as follows:

- I'd like to see the dentist.
- Can you tell me what the weather is like today?

Intent

An **intent** represents an action that the user wants to perform, based on the content of their utterances. The bot infers the intent and supports it based on its internal set of business rules or application flow, with the result of either a change in its internal state or an action being performed. These also typically result in a response being provided to the user as feedback or information.

So, from the preceding utterance examples, a bot might infer intents such as the following:

- I'd like to see the dentist => SeeDentist
- Can you tell me what the weather is like today? => GetWeather

Inferring intent is a large part of what an NLU platform such as Lex does behind the scenes. A number of training examples, in the form of sentences that the user might provide,are fed to the platform, and a probabilistic model is built from these examples. This means that, in practice, the platform should be able to infer the correct intent from input which is similar to, but not necessarily a part of, the examples that the system was trained on.

Prompts

When the bot requires more information from the user or is unclear about an Intent, it can ask the user follow-up questions, in order to collect more data. These are called **prompts**. Prompts typically fill in Slot values that are required, although your application logic may attempt to fill in values which are optional as well if you desire.

Slot

A **Slot** is a piece of information, or parameter, that is associated with an intent. The information can be provided within the initial user request, and Lex will be able to parse out the information and correctly assign it to the corresponding Slot correctly. If this information is not provided as a part of the request, then the bot should be able to prompt the user for the information separately. Slots may be optional or required.

The type of information represented by a **Slot** is known as the Slot type. There are a number of built-in Slot types within Lex that represent common types of information, such as city or state. The following are a few examples of common Slot types that are built into Lex:

AMAZON.Actor	Names of actors and actresses.	tim roth, amy adams
AMAZON.Airline	Names of a variety of airlines.	air france, british airways
AMAZON.Animal	Names of many different animals.	blister beetle, opossum
AMAZON.Artist	Full names of artists.	michael jackson, paul mccartney
AMAZON.Color	Names of colors.	light brown, lemon

Figure 4.1: Table of Slot types built into Lex

Of course, this is just a very limited subset of examples. There are many more built-in types, as well as different types for different languages!

> **Note**
>
> You can refer to the following link to get a full list of built-in intents and Slots: https://docs.aws.amazon.com/lex/latest/dg/howitworks-builtins.html

Most of the built-in intents and Slots are documented as part of the Alexa Skills Kit documentation, with some differences for Lex, which are documented at the preceding link. Make sure to keep the link bookmarked and refer to the page often, since Amazon keeps updating the service, and things may change.

If the type of information that you would like your bot to handle is not represented by one of these built-in types, you can define your own, along with the actual values that the Slot is allowed to take. You will see how this works as part of our next exercise.

Fulfillment

Note that the bot will not be able to proceed to the next step until it fills in all of the required Slot values! Naturally, this does not apply to Slot values that are optional.

When all of the required Slots for an intent have been filled, Slot is then ready for fulfillment. At this stage, the bot is ready to execute the business logic required to fulfill the intent. Business logic may be any of the following actions:

- Change in internal state
- Running code internally
- Calling an internal or external service, to get information from it
- Calling an internal or external service to post information to it

The fulfillment action can be performed with or without some feedback to the user, but as a matter of best practice, it is always better to err on the side of more feedback to the user, rather than less.

Setting Up with Amazon Lex

Introduction

For this exercise, we will be creating a sample chatbot, which is an option provided with **Amazon Lex**. The goal here is to understand the various components of the sample chatbot and relate them to the previous topic, where you learned about Natural Language Understanding components. After this exercise, you should be able to navigate the Lex user interface and easily create intents, Slots, and Slot types easily for your next exercise, where you will build a custom chatbot based on your own business logic.

Exercise 15: Creating a Sample Chatbot to Order Flowers

In this exercise, we will create and test the sample chatbot as a means to gain familiarity with the **Amazon Lex** console interface. The following are the steps for completion for creating a sample chatbot:

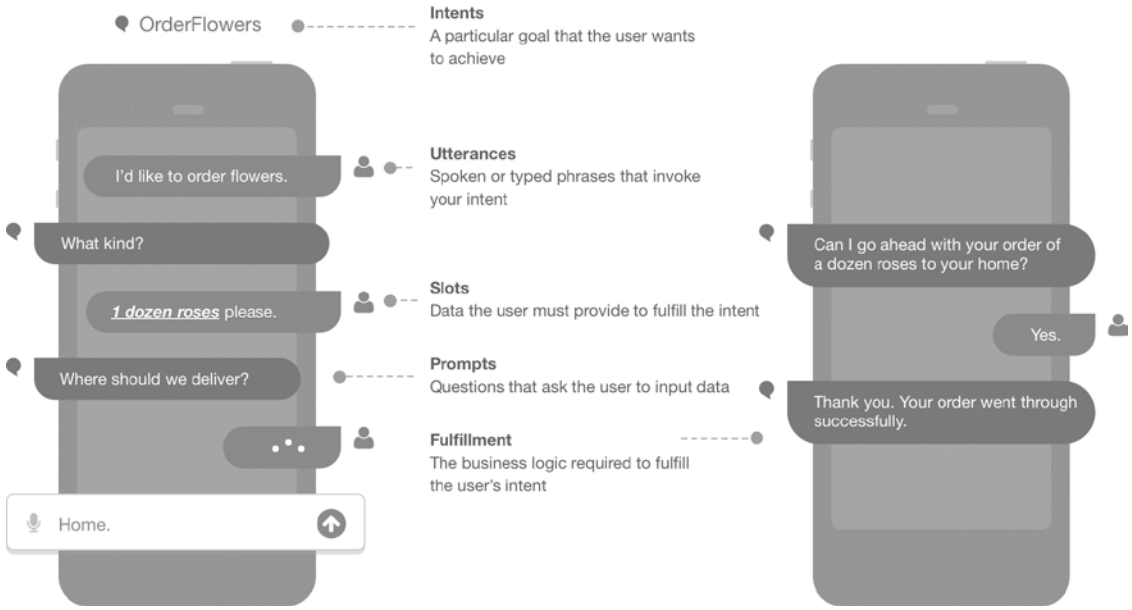


Figure 4.2: Sample chatbot interface

1. Let's first navigate to the **Amazon Lex** main screen. You can click on the Lex service link from the main AWS console screen or navigate directly to https://console.aws.amazon.com/lex.
2. If this is your first time using Amazon Lex, you should see the following screen:

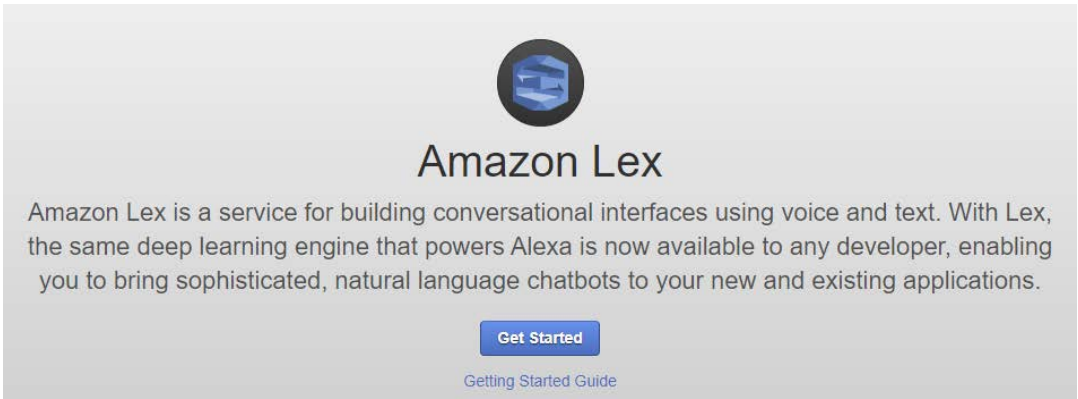


Figure 4.3: Amazon Lex Getting Started screen

3. You can click on the **Get Started** button in order to proceed.

> **Note**
>
> In case you have already built a bot previously, you will be shown a different screen, where you can click on the **Create** button, instead:

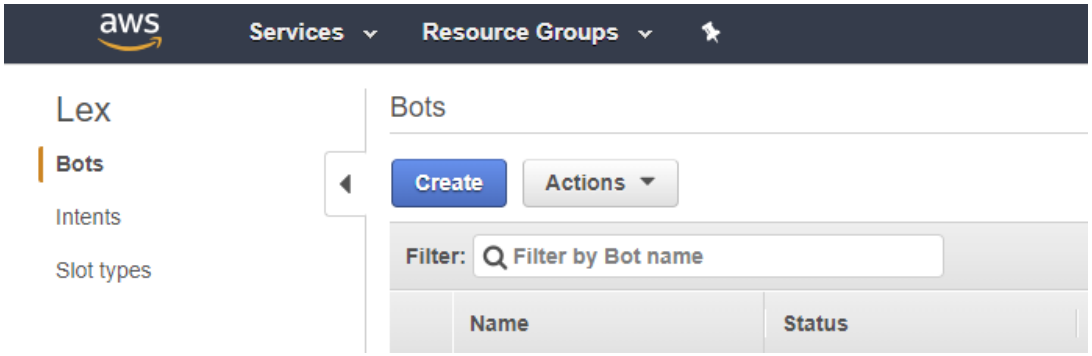


Figure 4.4: Bots create screen

4. You will then be presented a list of options for creating bots. This shows two options: **CREATE YOUR OWN** and **TRY A SAMPLE**.

5. Now, choose the **OrderFlowers** option under the **TRY A SAMPLE** section:

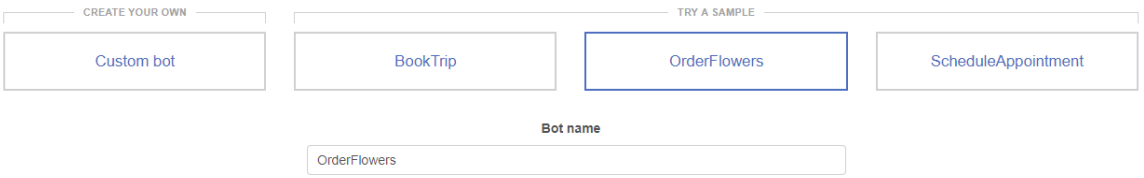


Figure 4.5: Selecting the bot

6. This will then present you with further options to be filled in. You can leave the bot name as the default, **`OrderFlowers`**, and select **No** for the **`COPPA`** option. You can leave the **`IAM`** role option as is. It will create a role automatically for the sample bot:

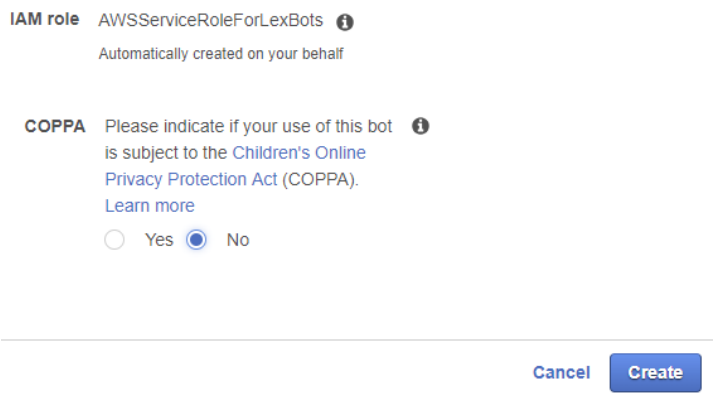


Figure 4.6: IAM role and COPPA selection

7. Click on the **Create** button to bring up the next screen and proceed to the next step.

Working with the Chatbot Editor

1. On the **Chatbot Editor** screen, you can view the predefined **`Intents`**, **`Slots`**, **`Slot Types`**, and **`Sample utterances`**. You will see that there is a single **Intent**, named **`OrderFlowers`**:

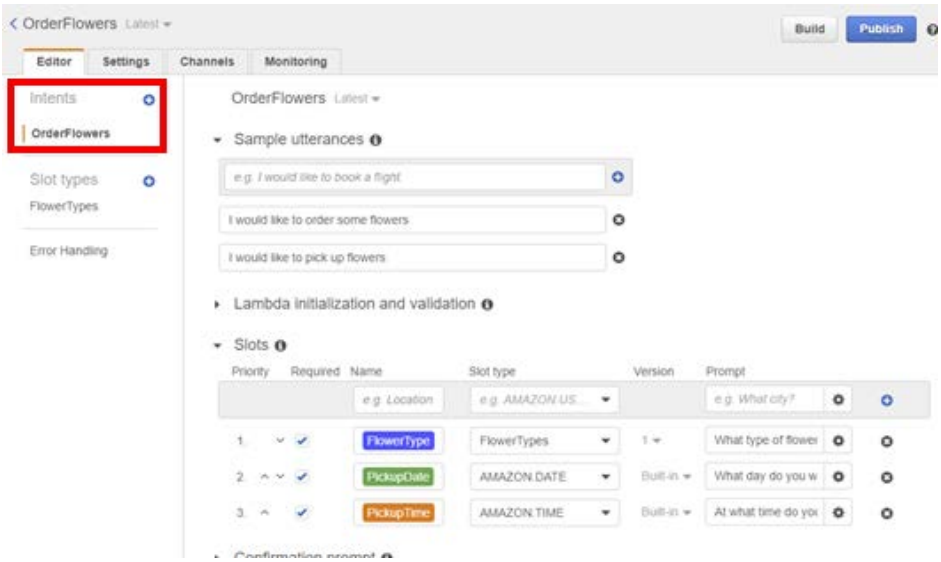


Figure 4.7: Chatbot editor screen

2. The sample utterances that are entered correspond to this Intent:

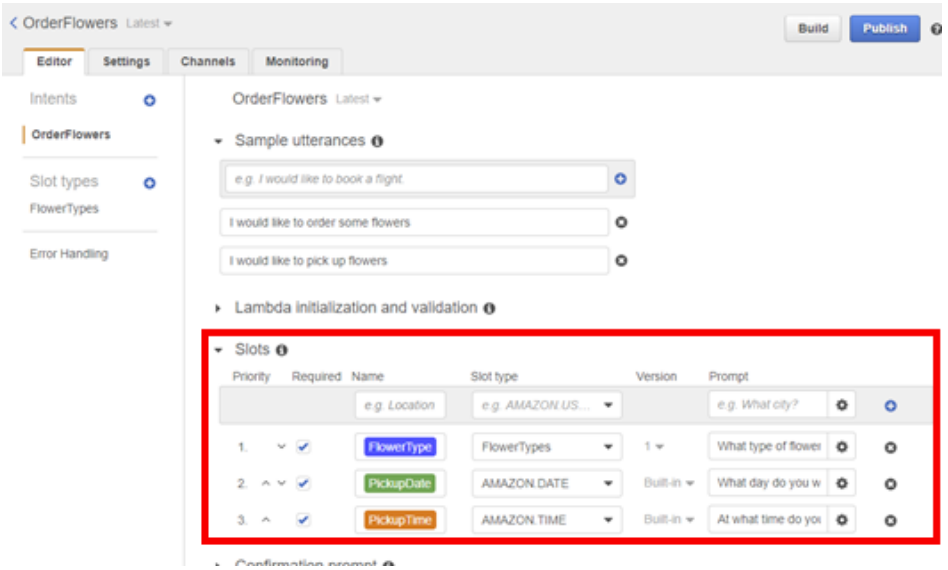


Figure 4.8: Slots types screen

3. There are also three Slots. Two of the Slots (**`PickupDate`** and **`PickupTime`**) take built-in types. The third one is a custom-defined Slot named **`FlowerType`**. Each Slot has a prompt associated with it, which is generated by the bot in order to get the information for the Slot.

4. There is also a **`Slot type`** named **`FlowerTypes`**. These values are recognized for the **`FlowerType Slot`** when the user is prompted for the type of flower to be ordered:

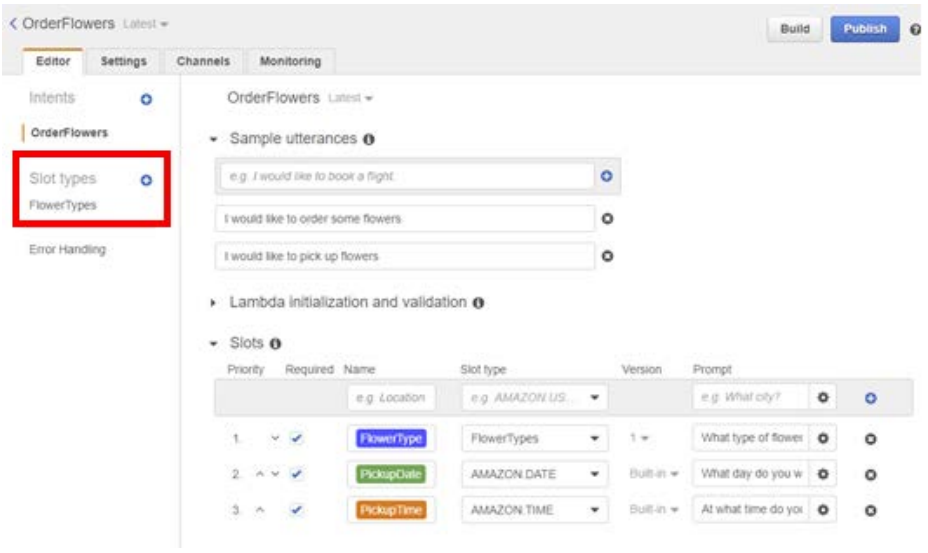


Figure 4.9: Slots types screen

5. Click on the **`FlowerTypes`** link in order to bring up a dialog with the definitions of the Slot type and sample values. The following are the steps for selecting and editing the Slot type.

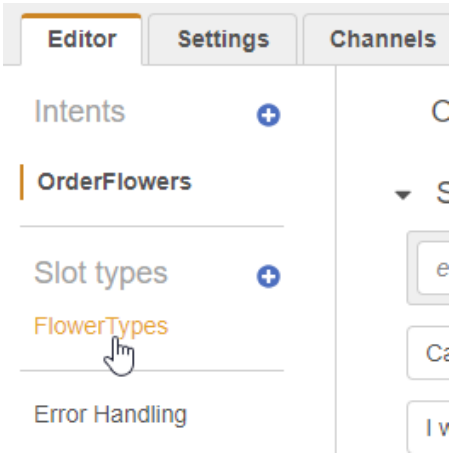


Figure 4.10: Selecting FlowerTypes

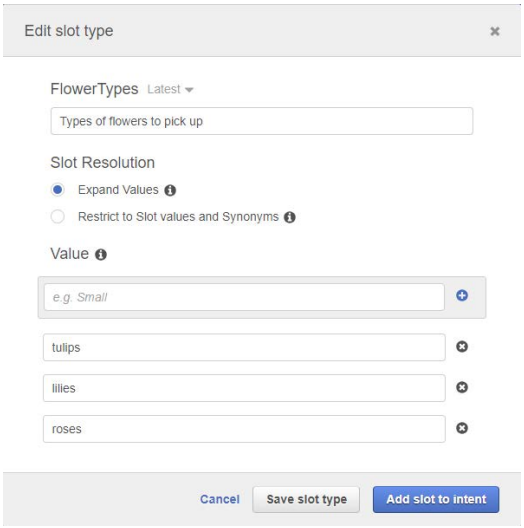


Figure 4.11: Edit Slot type

6. You can also click on the **Error Handling** link in the left area to show the prompts that the bot will show the user for clarification (**Clarification Prompts**) and the phrase used to terminate the interaction (**Hang-up phrase**) after the maximum number of retries has been attempted by the user:

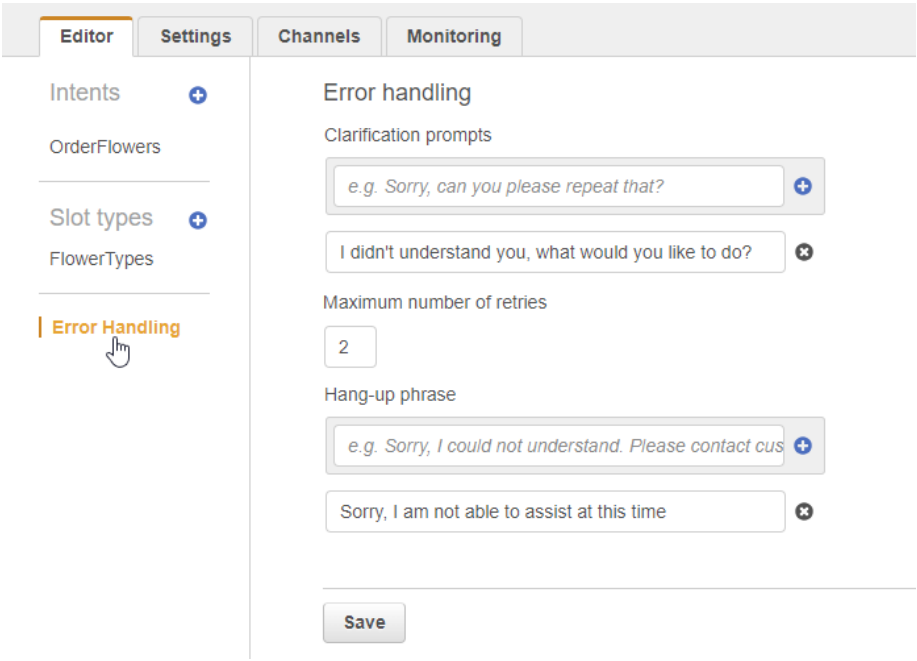


Figure 4.12: Error handling

Testing the Chatbot

1. You can test the chatbot in the Test bot area on the right-hand side of the screen. You may have to click on an arrow icon in order to expand it, if it is not already open:

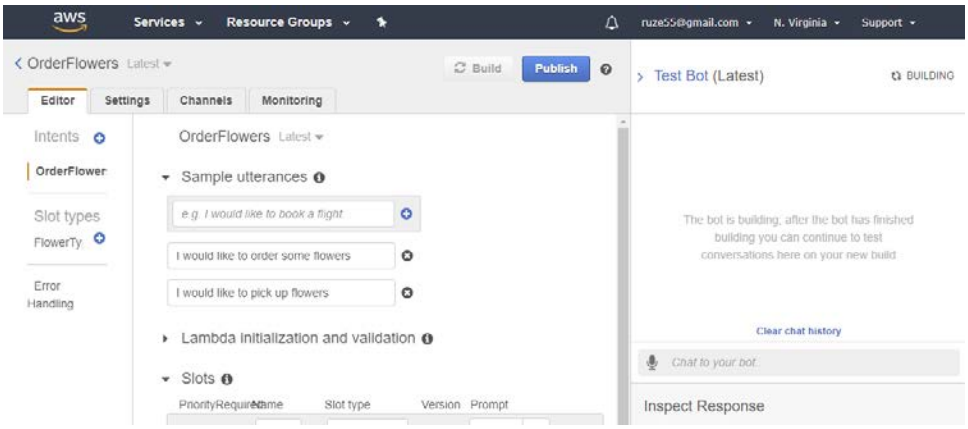


Figure 4.13: Test bot screen

2. You can have a complete interaction, as shown in the following screenshots:

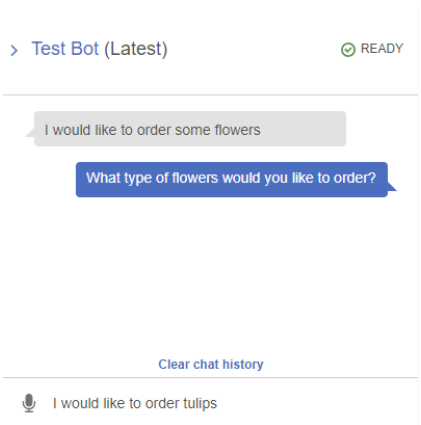


Figure 4.14: Chatbot interaction screen

3. The following screenshot shows the conversation with the bot:

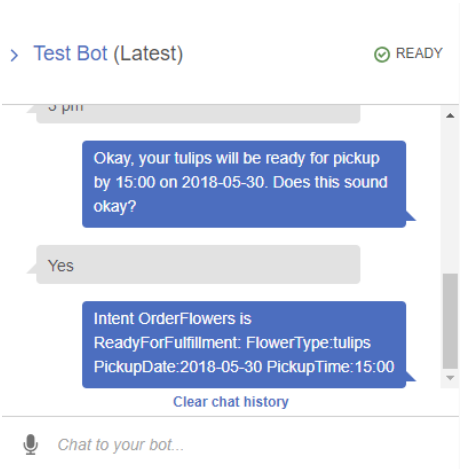


Figure 4.15: Chatbot interaction screen

When all the required **Slots** are filled, the **intent** is said to be ready for fulfillment. This means that all of the information required by the intent has been provided, and the application logic behind the chatbot can now handle the intent appropriately.

4. Intent can be returned to the calling application in the form of a **JSON** object or a **Lambda function** that has been implemented in the **AWS Lambda service.** This can be called at this point, with the **JSON** information as a parameter passed to it. You will see how this works later:

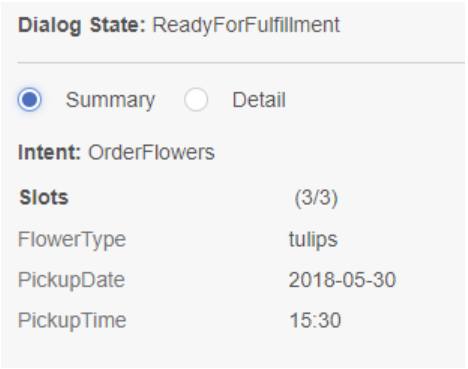


Figure 4.16: Summary view screen

5. As for the results displayed in the Dialog State pane, the first one shows a text representation of the state of the chatbot. This is the summary view which displays information such as the recognized intent, the Slot types, and the values that are now available from the Lex NLU system.

6. The Dialog State is **ReadyForFulfillment**, which signifies that the intent is ready for fulfillment, as shown in the preceding screenshot:



Figure 4.17: DialogState

In the detail view, the **JSON** object that would be returned to an application, which is interacting with the chatbot, is displayed. Using **JSON** is a structured way in which an application can retrieve information from and send information to the chatbot system in Lex. You can see that the information being provided here is the same as in the Summary view, except in the **JSON** format.

> **Note**
>
> The Lex JSON structure for intents, as well as formats for other resources, such as Slots and Slot types, are documented here: https://docs.aws.amazon.com/lex/latest/dg/import-export-format.html.

Creating a Custom Chatbot

In this topic, we will create a custom chatbot to get stock market quotes, using **Amazon Lex**. The bot will listen to our utterances for a valid intent: **GetQuote**. This signals to the bot that we had, for example to get a stock market quote for a given stock ticker symbol, will reside in a Slot named **ticker**. The bot will then look up the quote for that ticker symbol from a freely available financial API named **IEX**, and will return the information to the user via a conversational response:

> **Note**
>
> A stock ticker symbol is the standard way in which stocks that are traded on an exchange, such as the New York Stock Exchange or NASDAQ are represented. A sequence of alphabetical letters represents the company's stock which is being traded.

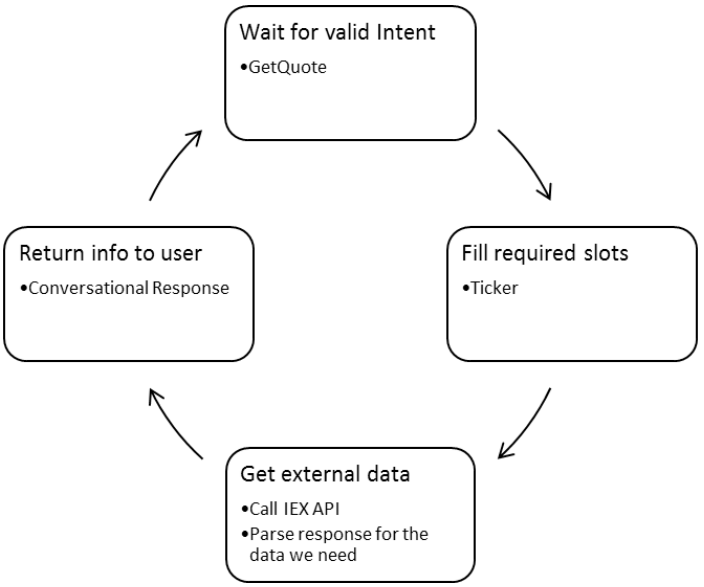


Figure 4.18: The chatbot's workflow

We can create a flowchart for this process, as shown in the following diagram. Let's go over it in some more detail:

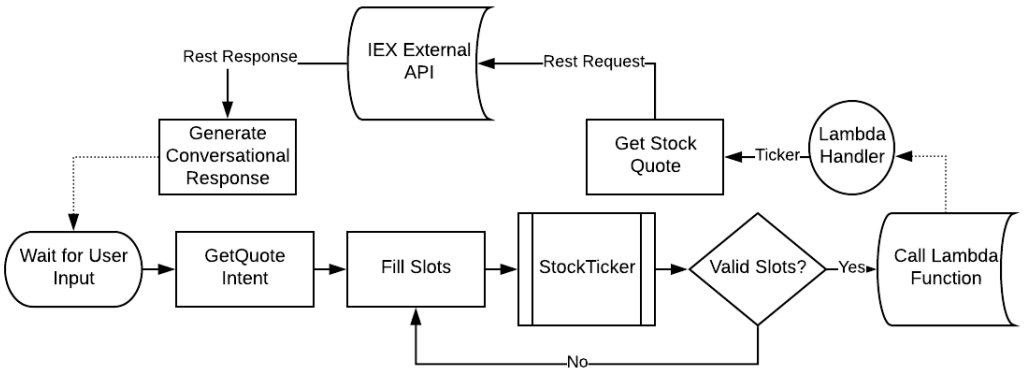


Figure 4.19: Flowchart of the chatbot's workflow

Recognizing the Intent and Filling the Slot Value

As the first step, the bot waits for the user's input, in order to recognize a valid intent. When it extracts the **`GetQuote`** intent as the intent from an utterance posted by the user, it will then try to fill the required Slots. In our case, we have only one Slot of the **`StockTicker`** type (which is a custom Slot type). The bot will issue a prompt, asking the user to provide the value of the Slot and parse the utterance in response, in order to fill the Slot value.

Valid Slots are those that the system recognizes. If the Slot value is not part of the list of allowed values, or if the system does not recognize what is entered for the Slot value, it is said to be **invalid**, or **not valid**.

If the Slot value is not valid, it will go back to trying to fill the Slot (at least up to the number of times we have specified it should try before giving up and going back to the beginning). Once the bot has a Slot filled with a valid value, it then proceeds to fulfill the intent.

Fulfilling the Intent with a Lambda Function

While the default fulfilment action is to return the intent and Slot value to the user so that he/she can proceed to work with it within his/her own application, we will instead choose to set up a Lambda function on AWS, which can handle the intent and run the business logic required to fulfil it.

At this point, the bot process running within Lex proceeds to call the Lambda function, which we have written and specified for fulfilment.

Lambda_function.Lambda_handler

When Lex calls out to the function for fulfilment, it sends a **JSON** payload containing various pieces of information about the sender, as well as the intent and Slot value. The **Lambda_handler()** method parses the intent and Slot parameter value from the **JSON**, and then dispatches another function call to the method, which gets the market quote value that we're looking for from the external API.

Finally, the **Lambda function** also packages the response as another **JSON** string and returns it back to Lex. Lex parses the **JSON** response behind the scenes and presents the response message to the user.

We will go through all of these elements in a lot more depth in the next two activities. In the first activity, we will set up the new chatbot, and in the second one, we will implement our Lambda handler function to return the actual value of the market price of the ticker symbol that the user asks the bot for back to him or her.

A Bot Recognizing an Intent and Filling a Slot

In the next exercise, you will create a custom chatbot which recognizes the intent, named **GetQuote,** to get a market price quote for a given ticker symbol. The bot will prompt the user for the value of the ticker symbol which the user is interested in, until the Slot is filled. You will also learn how to state the intent and fill the Slot in the same utterance. The chatbot can be tested via a conversational interface.

Exercise 16: Creating a Bot that will Recognize an Intent and Fill a Slot

In this exercise, we will create and test an Amazon Lex-based bot with a custom **`intent`** and **`Slot`**. The following are the steps for creating a bot with a custom intent and Slot:

1. The first step is to navigate to the **`Amazon Lex service`** within the **`AWS console`**, by clicking on the appropriate links within the AWS console, or navigating to https://console.aws.amazon.com/lex.
2. The next step is to click on the **Get Started** button, in order to get to the **bot** creation screen:

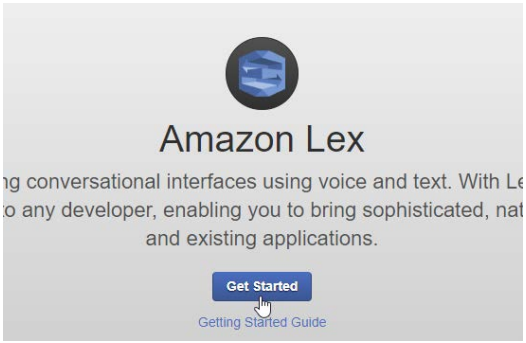


Figure 4.20: Getting Started screen

3. At this point, you can create a Custom bot by clicking on the **Custom bot** option button. This reveals the bot's details, which can be filled out, as shown in the following screenshot:

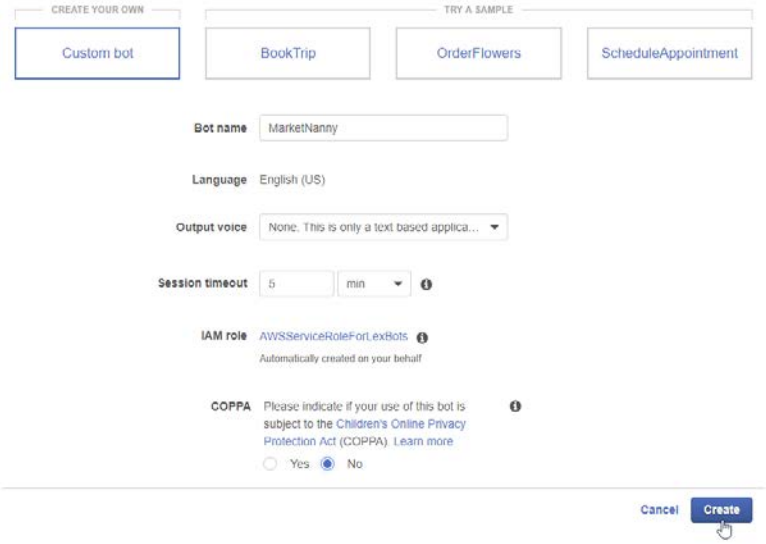


Figure 4.21: Custom bot option

4. The **bot** name field can be set to `MarketNanny`. The Output voice field is set to `None. This is only a text based application`. This is because we will be interacting with the bot only with text in this section, and not with voice just yet.
5. The session timeout can be set to the default of 5 min. The IAM role field displays the name of the IAM role, which is automatically created by Lex for use by bot applications.
6. Finally, the `COPPA` field pertains to the **Children's Online Privacy Protection Act**, to which online applications must conform. Assuming that no children under 13 are present in the class, you can click on **No**. If, however, you are a student under 13 or intend to have someone under 13 use your chatbot, then you should click on the **Yes** option, instead.

> **Note**
>
> A law was passed in 1998 to protect the privacy of children under 13. It states that online sites may not collect personal information from users younger than 13 years old without parental consent, among other provisions. You can learn more about the COPPA act at https://www.ftc.gov/enforcement/rules/rulemaking-regulatory-reform-proceedings/childrens-online-privacy-protection-rule.

7. Finally, clicking on the **Create** button will create the chatbot and bring you to the bot Editor screen. This screen will allow you to create and define an Intent for the bot, as well as a `Slot` with a custom `Slot type`.
8. Click on the **Create Intent** button to bring up an **Add** Intent pop-up dialog window:

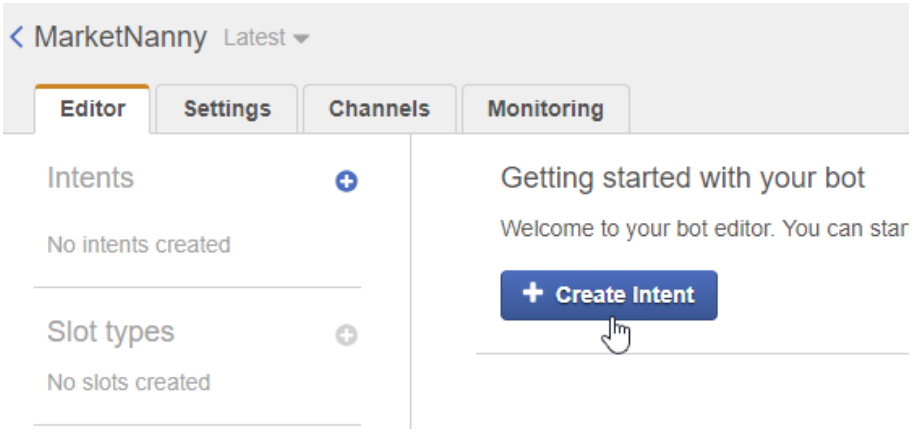


Figure 4.22: MarketNanny bot Editor

9. Conversely, if you already have an intent defined, you can create a new one by clicking on the **+** sign next to the Intents heading in the left-hand side column on the screen.
10. The Create **Intent window** offers a few options to add an intent to the bot. The Import intent link allows for importing an intent from a **`ZIP`** file containing one or more **`JSON`** files with intents in the Lex format.
11. The search for existing intents allows you to reuse the intents that you may have defined or imported previously, as well as the built-in intents defined by **Amazon Lex**.
12. You should just click on the **Create** intent link, however, to get to the following dialog box.
13. In the **Create** intent dialog box, name your new intent **`GetQuote`**. The bot will recognize this intent when you let it know that you are interested in a market quote. Click on the **Add** button to complete this step:

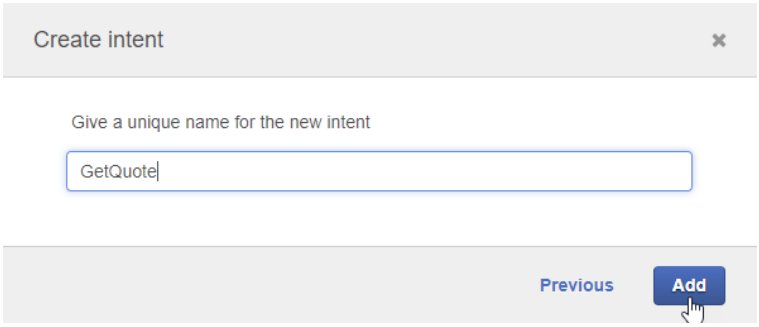


Figure 4.23: Create intent screen

14. You should be back at the Editor screen at this point, and you should see the **`GetQuote`** intent in the left toolbar portion of the screen. The Editor screen also contains a number of fields that are used to define and customize the new intent.

15. The first thing to do is fill in some Sample utterances to train the **NLU** system behind Lex to recognize the utterances you will provide to the bot as signaling the **`GetQuote`** intent:

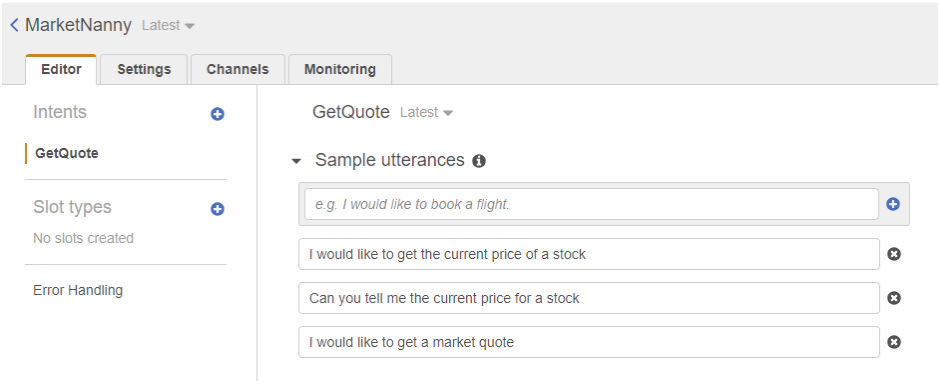


Figure 4.24: Creation of the Intent

16. After entering some sample utterances, you click on the **Build** button near the top of the page, in order to kick off the training process for the bot:



Figure 4.25: Building the bot

17. There will be a follow-up dialog box with another **Build** button, which you should also click:

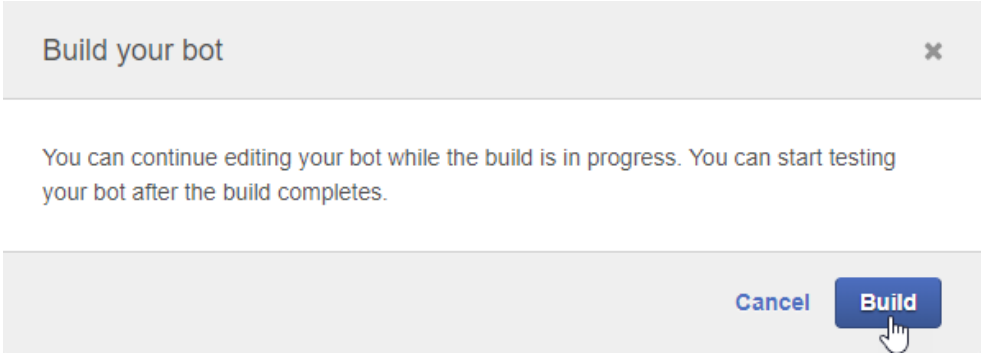


Figure 4.26: Build confirmation

18. After this, you should wait until you see the successful build dialog, which might take a few seconds to a couple of minutes:

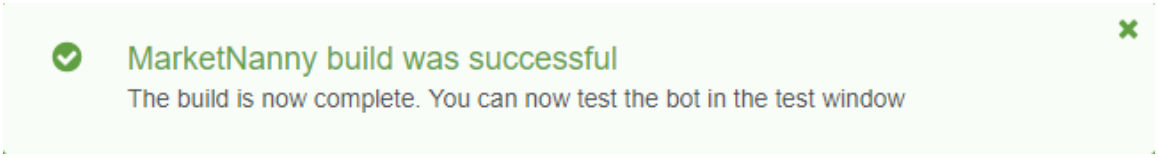


Figure 4.27: Bot build is successful

19. You can test your new intent within the bot in the **`Test bot`** pane, in the upper right-hand corner of the screen.

> **Note**
>
> If the Test bot pane is not visible, you may have to click on an arrow button in order to expand it and make it visible.

20. Type utterances into the pane to verify that the bot is able to recognize the correct intent from the utterances:

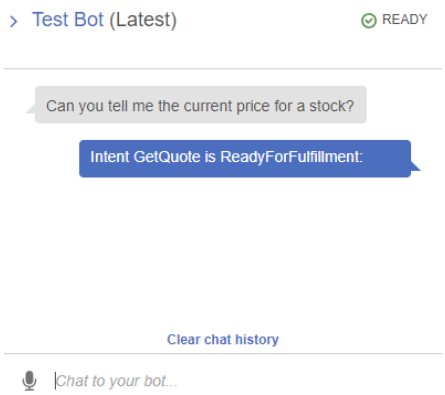


Figure 4.28: Test bot

21. You know that it has recognized the intent correctly when it returns the response: Intent **`GetQuote`** is **`ReadyForFulfillment`**. Feel free to experiment with different utterances, based on your sample utterances, in order to verify that the **NLU** engine is working correctly.

At this point, your bot does not do much, other than try to recognize the **`GetQuote`** intent and flag that it is ready for fulfillment. This is because we have not added any **`Slots`** to the **`intent`**.

Slot Addition

1. Your next step will be to add a Slot, along with a custom Slot type for the Slot:

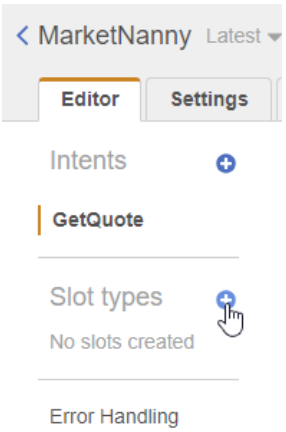


Figure 4.29: Adding a Slot

2. Add a **`Slot type`**. This can be done by pressing the **`+`** button next to Slot types in the left toolbar section of the Editor screen. This brings up an **Add Slot type** dialog box, where we can also choose to Import the Slot type as before with intents by using the Lex **`JSON`** structure. However, before, we will click on the **Create** Slot type link to create a **`new Slot`** type:



Figure 4.30: Creating Slot type

3. In the **Add Slot type** dialog box that pops up, enter the Slot type name as **StockTicker**. This is the name of the Slot type that we are defining. Optionally, you can enter a description in the **`Description`** field and leave the **Slot Resolution** option as **`Expand Values`**.

4. Under the **Value** field, enter a few **stock ticker** symbols, as shown in the following screenshot, to provide sample values for the **`StockTicker`** Slot type. You can add some of your own, as well, if you wish:

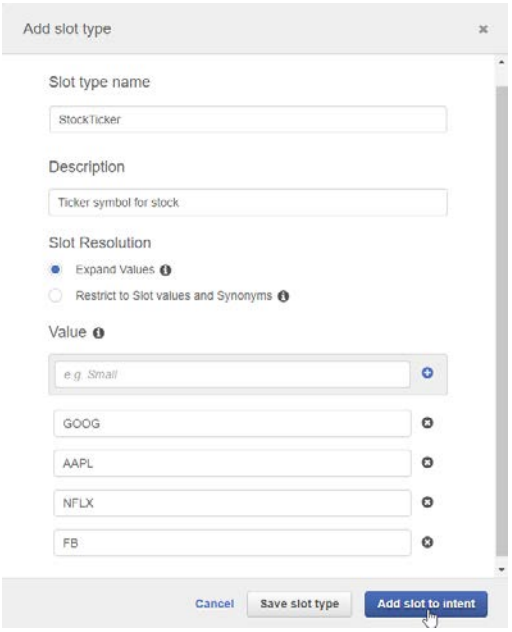


Figure 4.31: Adding a Slot type

5. Finally, click on the **Add** Slot to intent button, in order to add the Slot type to the intent, and close the dialog.
6. We could have also clicked on the **Save** Slot type button and added the Slot to the intent in a separate step, but using the button is the shortcut to accomplishing both actions in a single step.
7. When you close the dialog box, you will find that Lex has added a new Slot entry, as well, under the Slots section. It is helpfully prefilled with the **`StockTicker`** Slot type, and you should change the name of the Slot to ticker, under the **Name** field for the entry.
8. Click on the wheel under the **Prompt** field to expand it to a new dialog box:

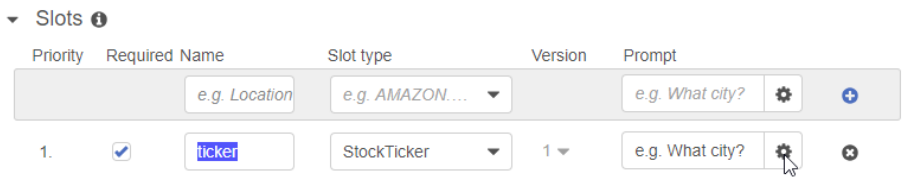


Figure 4.32: Editor dialog box

9. The prompts editor dialog box (named ticker Prompts) allows entry of prompts for the **`Slot`**, which the bot will use to store the user **`inputs`** and corresponding sample **`utterances`** that the user would typically provide to the bot while the bot is trying to elicit the information from the user with the prompts.

10. The placement of the Slot value within the corresponding utterances for the Slots is denoted by curly braces **`{}`** and the name of the Slot within the braces. In this case, since the Slot is named ticker, it is denoted by **`{ticker}`** within the sample utterances.

11. Fill in the prompts (a single prompt is fine – if you add more prompts, the bot will use them randomly, for variety) in the Prompts section.

12. Then, add some **`utterances`** to the Corresponding utterances section, denoting the placement of the Slot value, using the placeholder token **`{ticker}`** in each sample statement.

13. Leave the Maximum number of retries field as the default value of two. This means that it will try to get the value for the Slot twice before signaling an **`error`**:

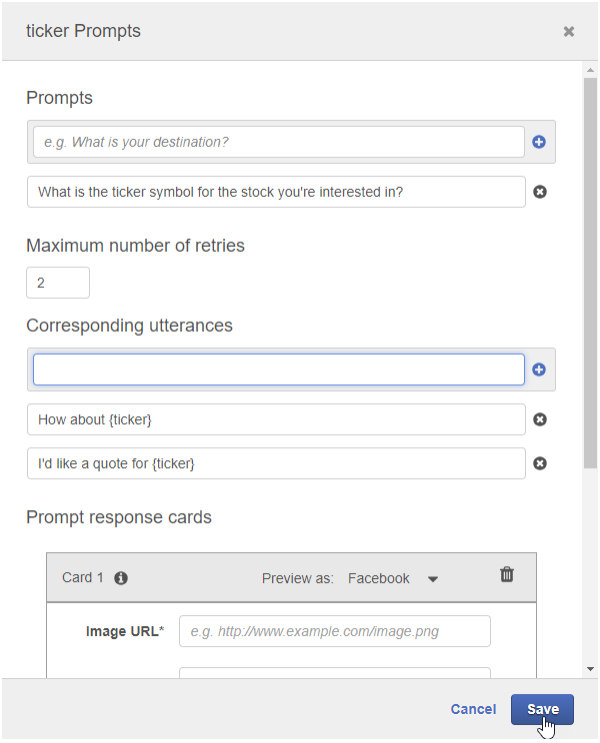


Figure 4.33: Ticker prompt screen

14. Finally, click on the **Save** button to save the Slot prompts and the corresponding utterances definitions.

15. Finally, click on the **Save** Intent button at the bottom of the screen, then the **Build** button at the top of the screen, in order to kick off the training process with the new Slot and Slot type that we have defined, and wait for the completion dialog to display when the training is done:

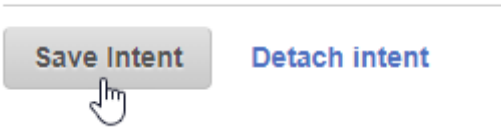


Figure 4.34: Saving the Intent

16. Your updated intent is now ready to test in the **Test bot** pane:

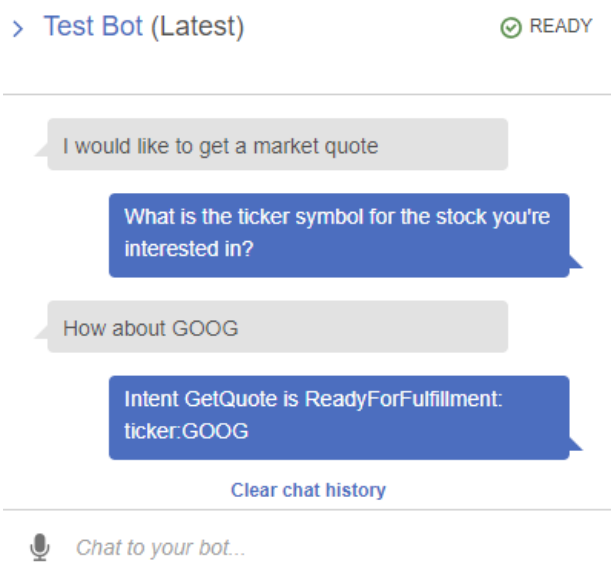


Figure 4.35: Updated intent test bot

17. At the end of the bot flow, when all of the information it requires is filled, it returns the intent in the same format as before. However, it follows this response line with another line, containing the name and value of the Slot parameter:

```
ticker:GOOG
```

18. This indicates that the ticker Slot has been filled with the value **`GOOG`**. So, that's great; our intent with Slot is working!

19. While you're having fun playing around with the bot to verify that the intent and Slot are working as they should, why not try something a little bit different: enter some utterances that are not a part of the sample utterances that you previously entered to train the bot.

20. Type **Can I get a market quote?** as your initial utterance, and see if the bot can recognize the intent. Bear in mind that the sentence, though similar to the sample utterances, is not one of those utterances:

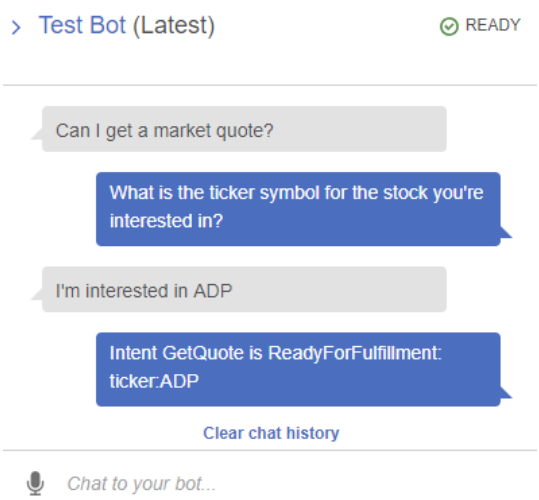


Figure 4.36: Test bot screen

21. As you can see from the testing shown in the preceding screenshot, not only does Lex recognize the correct intent from an utterance it has not been trained on, but it also recognizes a new symbol which it has not seen before (ADP) correctly as the value for the ticker Slot.

22. Now, let's try a conversational form of a corresponding utterance for the Slot prompt by inserting a random interjection as a part of the sentence, again using a new ticker symbol (**VZ**) that the bot has not previously trained on. Again, it is correctly processed and recognized:

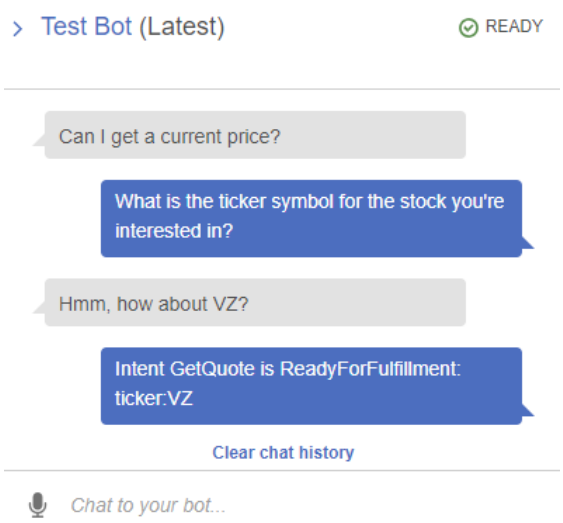


Figure 4.37: Test bot screen

23. Clearly, there is quite a bit of flexibility possible between training and real-world examples of conversational input with an NLU engine.

Natural Language Understanding Engine

NLU demonstrates the advantage of using an NLU engine that has been trained on a huge set of conversational sentences and has formed a large inference model.

It is able to connect sentences that are not the same as the ones it has specifically been trained on. In fact, they can be significantly different, but the model is large enough to infer that the semantic meanings are similar.

There is one more trick that you can use to make it easier for the user to interact with your bot. You can fill the Slot value in the same utterance as the one which establishes intent. This can be accomplished by simply including the Slot placeholder token (`{ticker},` in this case) in your sample utterances:

1. Add a new sample utterance to your **`GetQuote`** intent, as follows:

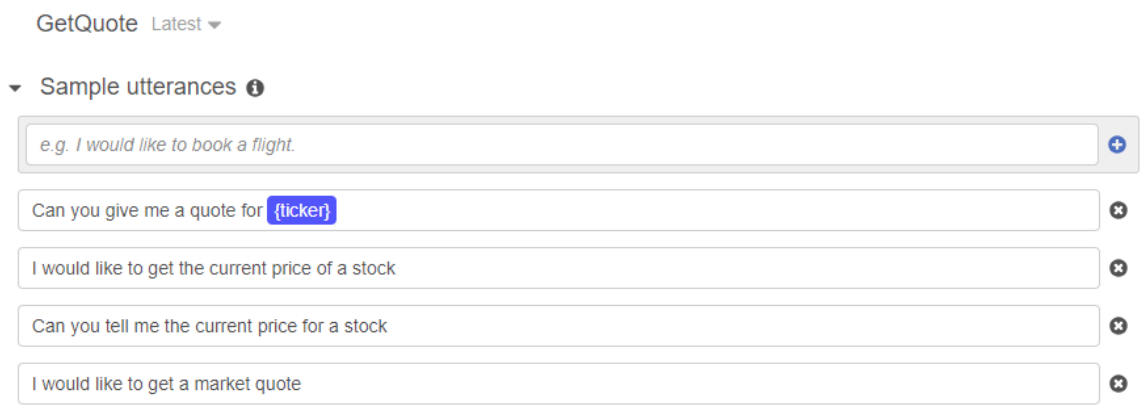


Figure 4.38: GetQuote screen

2. The **ticker** placeholder token denotes that that Slot may be filled directly within the initial utterance, and, in that case, a prompt doesn't need to be generated:

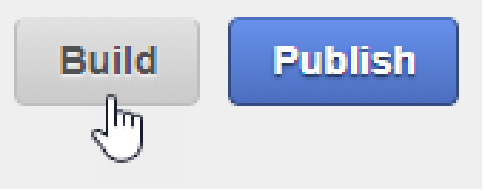


Figure 4.39: Build screen for the bot

3. Click on the **Build** buttons to train your updated intent as before, and then test it in the Test bot pane, as follows:

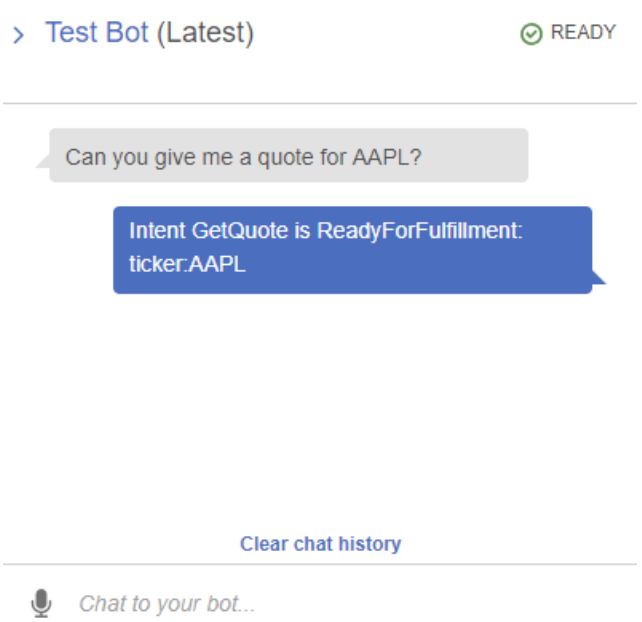


Figure 4.40: Test bot screen

4. You can see that the intent is ready for fulfillment and that the Slot value is filled appropriately, in a single step.

We have now gone through the process of defining a custom chatbot, complete with a custom intent, Slot type, and Slot, within Amazon Lex. Furthermore, we have trained and tested the bot to verify that it is able to classify the correct intent and correctly infer the Slot values from conversational input, to a high degree of accuracy. Finally, we added a shortcut method to fill in the Slot value directly in the initial utterance, by inserting the placeholder token for the Slot value in the sample utterance to train the NLU engine behind Lex.

Lambda Function – Implementation of Business Logic

You can create AWS Lambda functions, so that you can run your code for your Amazon Lex bot. You can recongnize Lambda functions to boot up and validating the fulfillment, in your intent configuration. Without a Lambda function, your bot returns the intent information to the client application for fulfillment.

In the next exercise, you will learn how to implement the business logic behind the bot as a Lambda function in AWS and call a real-world REST API to get information to return to the user from an external service.

Exercise 17: Creating a Lambda Function to Handle Chatbot Fulfillment

In this exercise, we will handle chatbot fulfillment business logic with a **`Lambda function`** that is created and deployed on AWS. In the last exercise, we created a chatbot with a **`GetQuote`** intent and ticker Slot. In this exercise, we will implement the fulfillment business logic. The following are the steps for implementing business logic:

1. Navigate to the AWS Lambda screen via the AWS Console, or by navigating directly to: https://console.aws.amazon.com/Lambda.
2. If you have never used Lambda before, you should be presented with a Welcome screen:

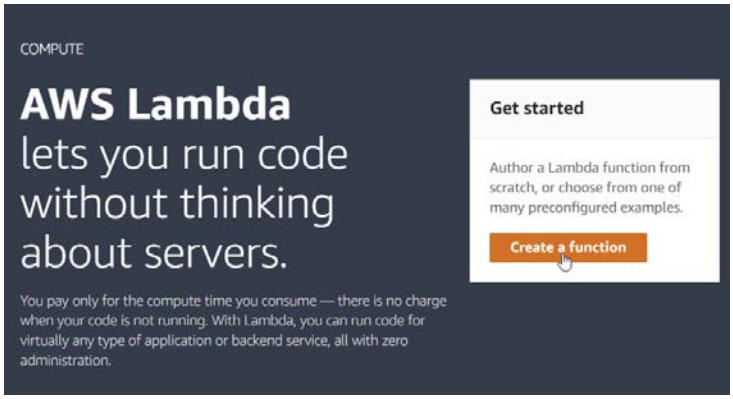


Figure 4.41: AWS Lambda start up screen

3. Click on the **Create** a function button, in order to get started.
4. Select the **Author from scratch** option on the next page:

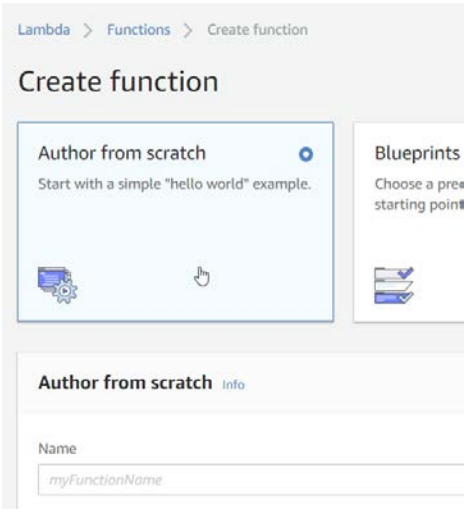


Figure 4.42: Selecting an author

5. For the runtime, choose **`Python 3.6`** from the drop-down menu as you will be implementing the handler in the Python language for this exercise. In the Name field, fill in **`marketNannyHandler`**:

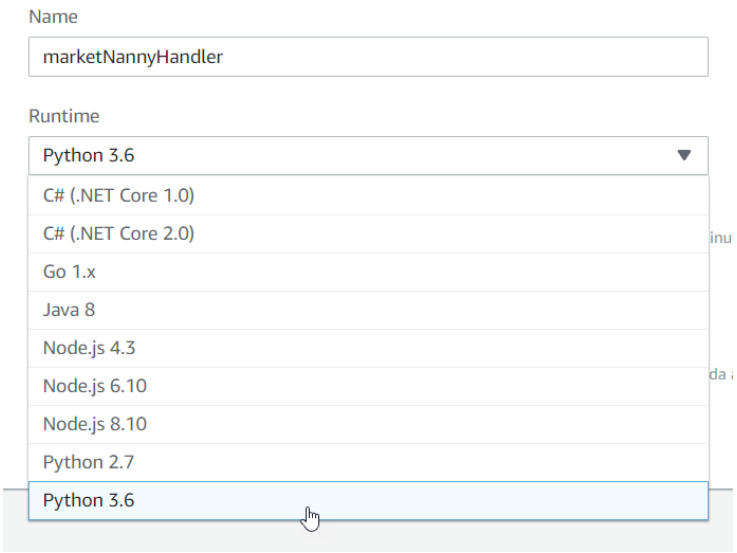


Figure 4.43: Filling in the values

6. For the Role field, choose **Create** new role from template(s) from the drop-down menu:

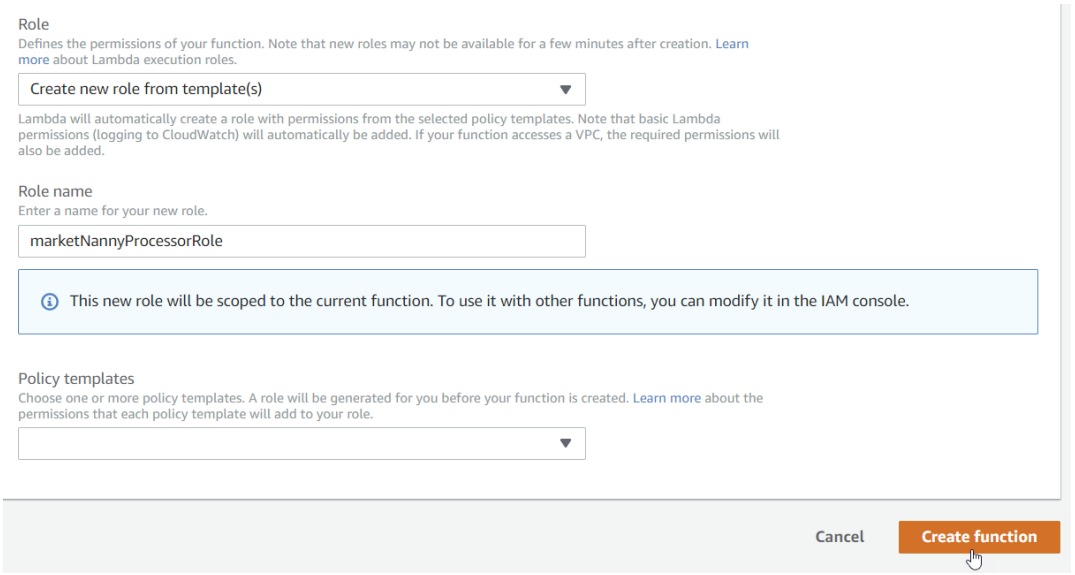


Figure 4.44: Role selection screen

7. Enter the name **`marketNannyProcessorRole`** in the Role name field. Then, click on the **Create function** button to create the Lambda function in AWS. You should see a confirmation screen, as follows:

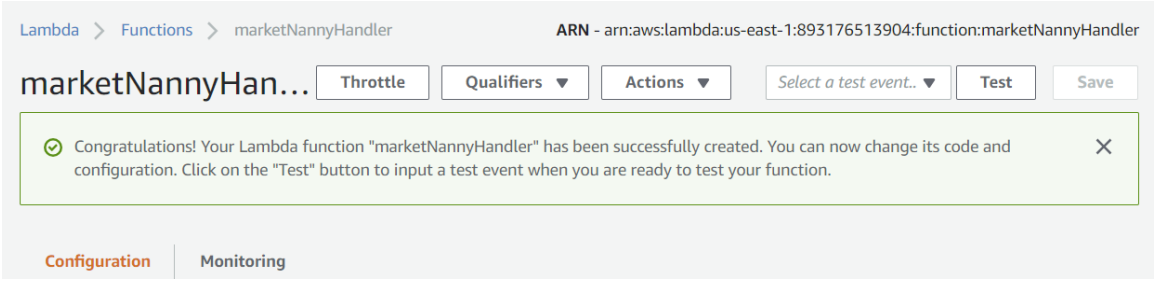


Figure 4.45: Confirmation screen

Implementing the Lambda Function

Here, you will use the Lambda Function editor entirely in-line, which means that you can enter and modify the code directly, without having to upload any files to AWS. The code that you enter will be executed when the Lambda function is invoked:

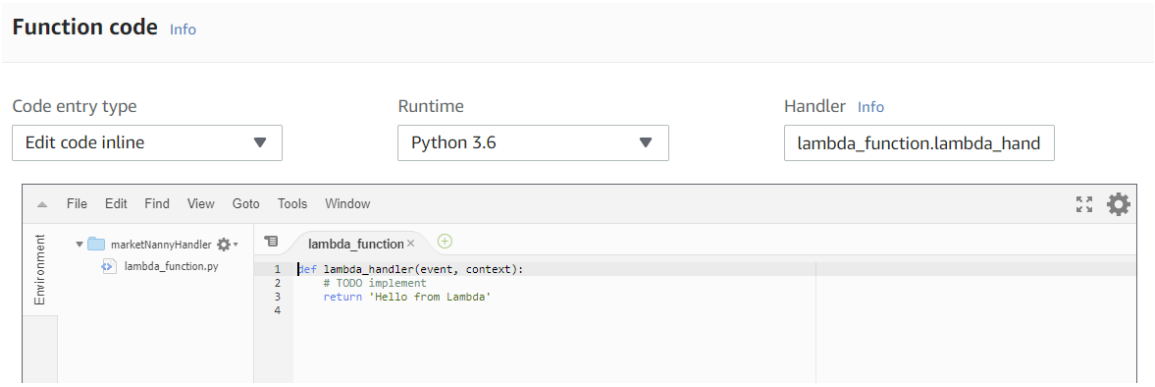


Figure 4.46: Function code screen

First, let's look at the structure of the Lambda function.

When you created the function named **marketNannyHandler**, AWS created a folder with the same name, with a Python file named **Lambda_function.py** within the folder. This file contains a stub for the **Lambda_handler** function, which is the entry point of our Lambda function. The entry point takes two parameters as arguments:

- The event argument provides the value of the payload that is sent to the function from the calling process. It typically takes the form of a Python **dict** type, although it could also be one of **list**, **str**, **int**, **float**, or **NoneType**.
- The context argument is of the type **LambdaContext** and contains runtime information. You will not be using this parameter for this exercise.

The return value of the function can be of any type that is serializable by **JSON**. This value gets returned to the calling application after serializing.

Input Parameter Structure

Now, let's take a closer look at the structure of the event argument, which gets passed to the **Lambda_handler function**. If we are asking for a market quote with the ticker value **GOOG**, the **JSON** value of the intent section within the parameter will look as follows:

```
{
     …
    "currentIntent":
    {
        "name": "GetQuote",
        "Slots":
        {
            "ticker": "GOOG"
        },
     …
}
```

The relevant values which we are interested in for processing are **name** and the single **ticker** value within the **Slots** section under **currentIntent**.

Since our **JSON** input gets converted into a Python dictionary, we can obtain these values within the Lambda function simply, as follows:

```
event['currentIntent']['name']
event['currentIntent']['Slots']['ticker']
```

Implementing the High-Level Handler Function

The first step in implementing our handler is identifying the intent name and calling the corresponding function that implements it. This code looks as follows:

```
def get_quote(request):
    return "Quote handling logic goes here."


def Lambda_handler(event, context):
    intent = event['currentIntent']['name']
    if intent == 'GetQuote':
        return get_quote(event)
    return "Sorry, I'm not sure what you have in mind.  Please try again."
```

This is complete enough to actually be tested against your chatbot at this point, if you so desire, but let's press on with the implementation.

Implementing the Function to Retrieve the Market Quote

The next step will be to implement the **get_quote** function, which does the work of actually getting the market quote information and returning it to the calling handler function:

```
def get_quote(request):
    Slots = request['currentIntent']['Slots']
    ticker = Slots['ticker']
    price = call_quote_api(ticker)
```

Note that we have named the parameter as request, so the event **object** that we send the function to is referred to as a request within this function. It contains the same value and structure, just renamed. Therefore, we can get the value of the ticker Slot, as mentioned previously, by getting the value of the item with the **ticker** key under by using the following code:

```
request['currentIntent']['Slots']
```

We then call the **call_quote_api()** function to retrieve the value of the market quote for the value of the ticker item. We haven't implemented **call_quote_api()** yet, so let's do this next.

We will implement the **call_quote_api** function, as follows:

```
import json
from urllib.request import urlopen
def call_quote_api(ticker):
    response = urlopen('https://api.iextrading.com/1.0/stock/{}/delayed-
quote'.format(ticker))
    response = json.load(response)
    return response['delayedPrice']
```

Where ticker is the value of the ticker parameter (in this specific example, it would be **GOOG**). We use the IEX API, which provides a static endpoint on the internet at https://api.iextrading.com, to retrieve a delayed quote, using the 1.0 version of their REST API.

Since it is implemented as a simple **GET** request, with the ticker parameter embedded within the **URL**, and no other special header values such as the API key or user information, we can simply use the built-in **urlopen** method in the **urllib.request** module (which we will have to remember to import) to receive a response from the URL with the ticker embedded within it.

Since the response is also in the **JSON** format, we need to import the **json module** and load the response using the **json.load function**. The only field we are interested in within the response is **delayedPrice**, so we return that as the return value from our function.

Returning the Information to the Calling App (The Chatbot)

Now that we have the market quote value, we can return it to our calling application, which is the chatbot that we implemented. We have to do a couple of small things, however, to return this value. First, we need to format it as a conversational response, as in the following string:

```
message = 'The last price (delayed) of ticker {} was {}'.format(ticker,
price)
```

This should let the chatbot display the following message:

```
The last price (delayed) of ticker GOOG was 1107.32
```

There is one final step, which is to construct an **Amazon Lex** `JSON` return format, containing our message and a couple of other items of information. We will use the helper function close to do this:

```
return close(message)
```

Our close function takes a single parameter, which is the string that we wish to return to the chatbot (in this case, this is the value of the message variable). It generates a `JSON` wrapper around the content, which conforms to the structure, which our Lex-based bot is expecting and from which it can extract the content and deliver it to the user. The structure of the wrapper is not important at this stage, but if you are curious, you can look at the implementation of the close function.

Connecting to the Chatbot

At this point, the only task remaining is to connect the Lambda function to the chatbot and test it:

1. To do so navigate back to the Amazon Lex dashboard and select the `MarketNanny` bot:

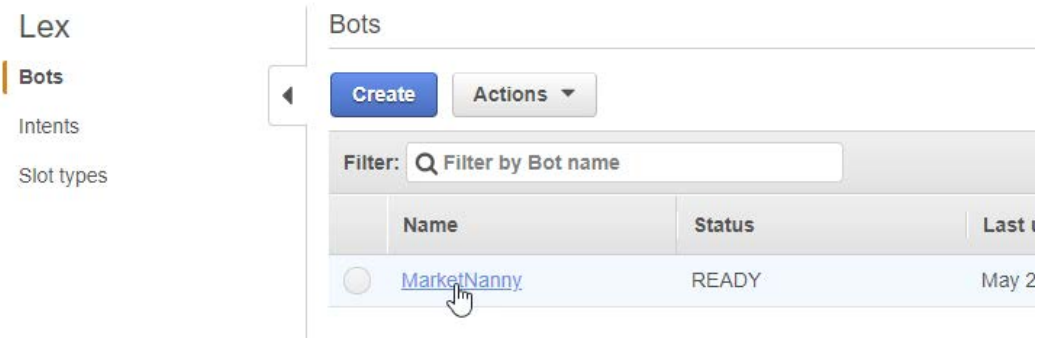


Figure 4.47: Connecting to the bot

2. Then, scroll down to the Fulfillment section and select the AWS Lambda function option. Next, select the `marketNannyHandler` function from the Lambda function drop-down menu and leave Version or alias as the default value of **Latest**:

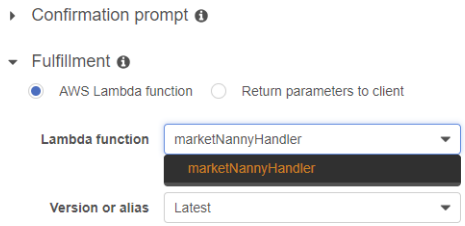


Figure 4.48: Confirmation prompt

3. Rebuild the intent by clicking on the **Build** buttons, and test the chatbot together with the Lambda handler in the Test Chatbot pane:

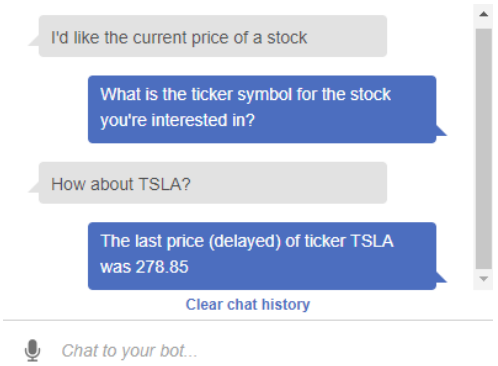


Figure 4.49: Chatbot updated

The following screenshot shows the interaction with bot for knowing the current price of AAPL:

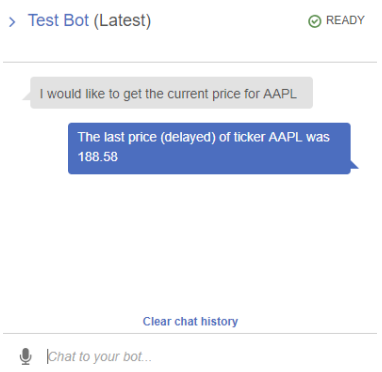


Figure 4.50: Chatbot updated

Activity 5: Creating a Custom Bot and Configuring the Bot

In this activity, we will create a custom bot for **`PizzaOrdering`**. A bot performs the automated task of ordering a pizza. An Amazon Lex bot is powered by **Automatic Speech Recognition** (**ASR**) and **Natural Language Understanding** (**NLU**) capabilities, the same technology that powers **Amazon Alexa**. For this, we will consider the following example: suppose that a user wants to order a pizza; a user might say, **Can I order a pizza, please?** or, **I want to order a pizza**. You can configure the intent so that Amazon Lex simply returns the information back to the client application, to perform the necessary fulfillment. To ensure that it happens correctly, you will need to navigate to the Amazon Lex services to create the custom bot:

1. Create a custom bot for **`Pizza Ordering`**.
2. Create an Intent.
3. Create a Slot type.
4. Configure the Intent.
5. Configure the bot.

> **Note**
>
> To refer to the detailed steps, go to the *Appendix A* at the end of this book on Page no. 217

Summary

In this chapter, you learned about the basic chatbots and chatbot design. You also learned how to set up the Amazon Lex services, and how to create a sample chatbot. Next, we looked at the Chatbot workflow. We then looked upon the flowchart of a Chatbot's workflow. Finally, we dove into creating a custom chatbot that shows share prices.

In the next chapter, we will look at how to use speech with chatbots.

5

Using Speech with the Chatbot

Learning Objectives

By the end of this chapter, you will able to:

- Set up Amazon Connect as a personal call center
- Integrate the MarketNanny chatbot with Amazon Connect
- Interact with the chatbot using voice and speech
- Connect the custom bot with Amazon Connect

This chapter describes Amazon Connect services for interacting with chatbots using speech and voice.

Introduction

In the previous chapter, you learned how to create the `MarketNanny` chatbot using the **Amazon Lex** service. You are able to interact with the chatbot using natural language speech via a textbox, and the chatbot is able to query an external service API to get the information to send back to you.

In this chapter, you will interact with the `MarketNanny` chatbot using speech, rather than text. First, you will set up a call center within Amazon Connect, with a local phone number. Then, you will connect `MarketNanny` to Amazon Connect.

Finally, you will test the chatbot by speaking to it; this will be done by calling it. By the end of this chapter, you will be able to talk to your `MarketNanny` bot via a telephone interface, using the Amazon Connect interface.

Amazon Connect Basics

Amazon Connect is a service from AWS that allows for the creation of cloud-based contact centers. This means that people can use a telephone number to contact the service and have conversations with bots or human representatives. In this case, we are primarily interested in automating interaction, using the chatbot that we built previously.

Some key features of Amazon Connect are as follows:

- It is easy to set up and use Connect with workflows that are defined using a graphical editor.
- There is no infrastructure to deploy or manage, so contact centers can be scaled up and down quickly.
- It is a **pay-as-you-go service**, so there is no setup or monthly fees. We will be using the free tier, so we should not have to pay anything to use the service. For commercial usage, charges are based on per minute rates.
- Connect is deployed in 42 availability zones, within 16 geographic regions.

Free Tier Information

In this book, you will only be using the free tier of the services presented. However, it is important to be aware of the limits of free usage and pricing otherwise.

You should check the official web-page https://aws.amazon.com/connect/. A single contact center may include a set amount of Amazon Connect service usage and a direct dial number including 60 minutes of inbound and outbound calls. This can change depending on your regional settings.

> **Note**
>
> You should not need to go beyond the services provided by the free tier for this book. If you go beyond the limits of the free tier, you will get charged by Amazon, at the rates published at: https://aws.amazon.com/connect/pricing/.

Interacting with the Chatbot

Using Connect to interact with your chatbot by voice, requires that you first set up a call center within Connect. The call center receives a free local phone number (or a US toll-free number, in the United States). This number can be connected to an **Amazon Lex** chatbot; in our case, this will be the chatbot that we have already created (`MarketNanny`). The chatbot uses the Lambda **serverless service** to get data from an external API and relay it back to the user on the phone:

Reference for image: https://aws.amazon.com/connect/connect-lexchatbot/

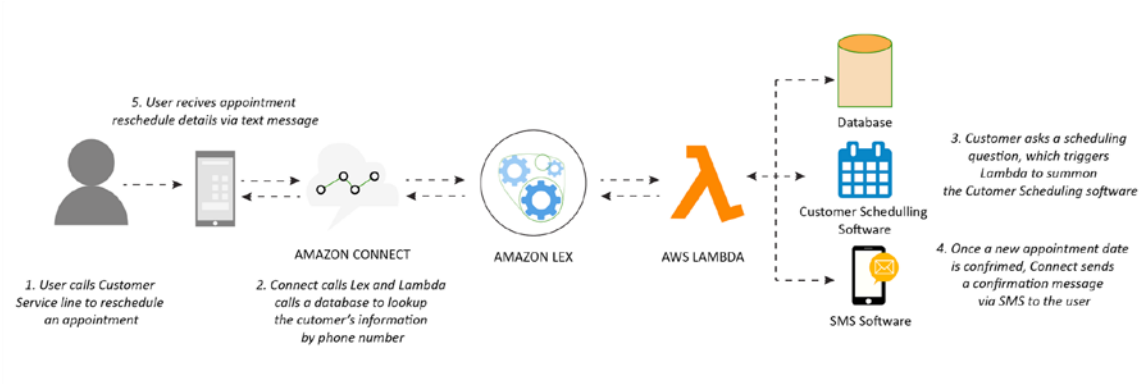


Figure 5.1: A sample voice-based chatbot application using Amazon Connect, Lex, and Lambda

Talking to Your Chatbot through a Call Center using Amazon Connect

Using Amazon Connect, we can create a call center application that will allow us to connect to the chatbot which we built in the preceded chapter (**`MarketNanny`**), using voice commands, rather than a textbox;

To create an Amazon Connect call center, you make an Amazon Connect occurrence. Each example contains the majority of the assets and settings identified with your call focus. You can oversee settings for your scenario from the Amazon Connect support. You can oversee settings for your contact focus from inside your contact focus.

You can make various occurrences; however, each example capacities just inside the AWS area in which you make it. Settings, clients, measurements, and details are not shared between Amazon Connect occasions.

> **Note**
>
> It is mandatory to have an AWS Account. Amazon Lex services can be accessed through Console page.

Exercise 18: Creating a Personal Call Center

In this exercise, we will create a personal call center using Amazon Connect, and we will connect your **`MarketNanny`** chatbot to it. We have already discussed the Amazon Connect service in this chapter:

1. For the first step, navigate to the Amazon Connect service page in the AWS Console. You can select the link to navigate to, or go directly to this URL: https://console.aws.amazon.com/connect:

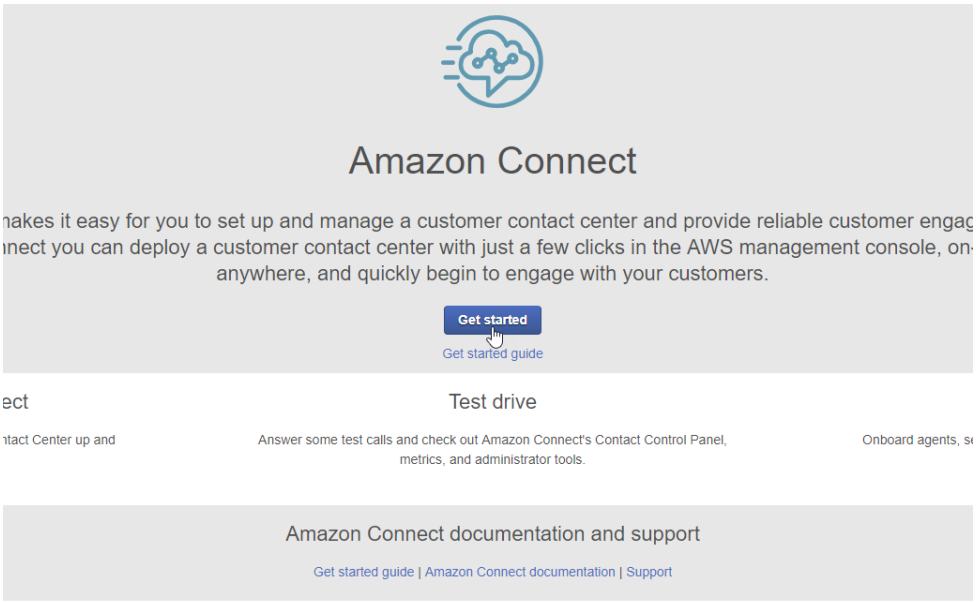


Figure 5.2: The Amazon Connect welcome screen

2. Click on the **Get started** button to create a new **Call Center**. In the first screen that follows, keep the default option of Store users within Amazon Connect, and enter a name for your application. The name is unique across AWS, and duplicates are not allowed, so you may have to experiment a bit or add numbers to the name until you get a name, which is unique.

3. Click on the **Next step** button when you are done:

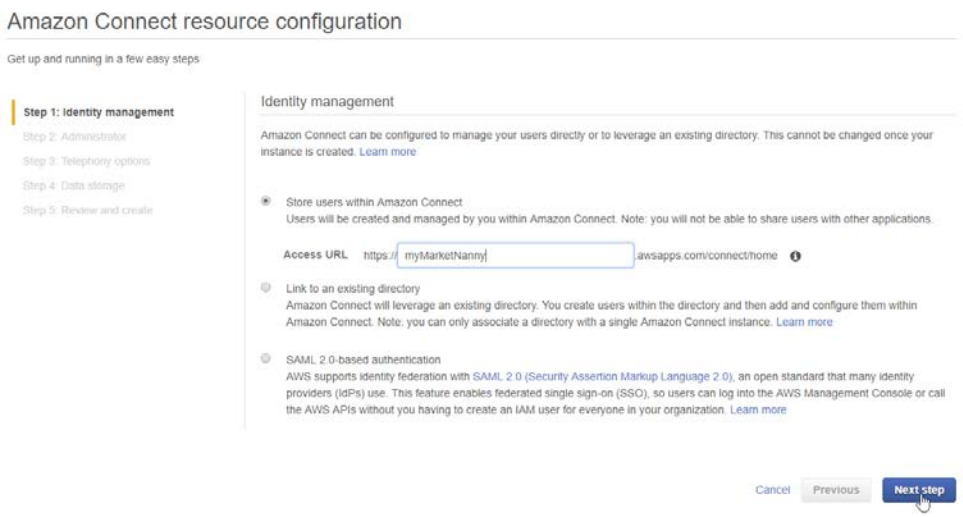


Figure 5.3: Step 1 of the Amazon Connect resource configuration screen

4. In the next step, add the information for a new **`admin`**, **`setting`** the **`email address`** to your own email address. Again, click on the **Next step** button when you are done:

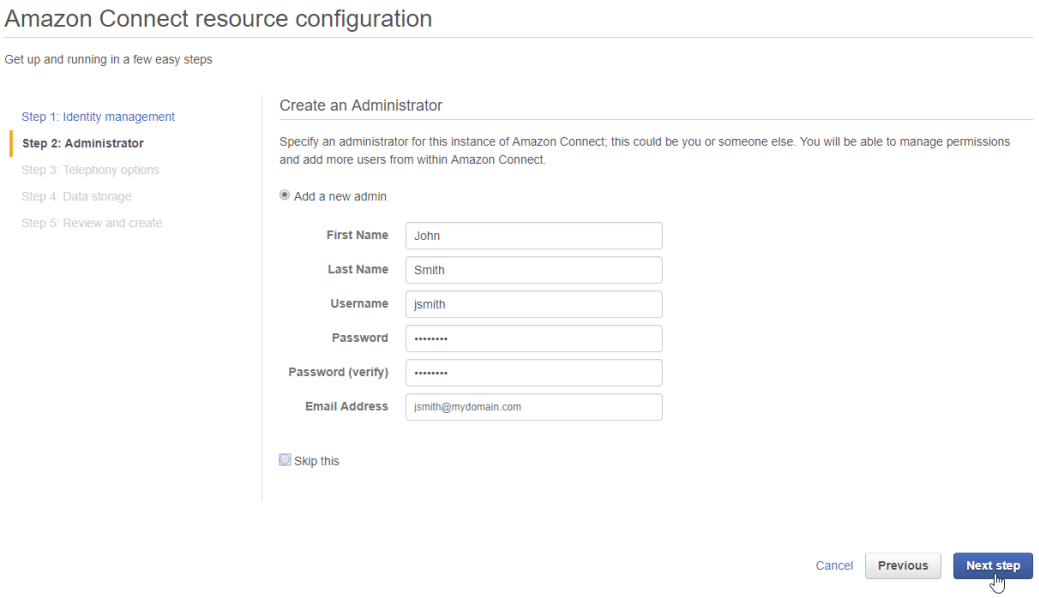


Figure 5.4: Step 2 of the Amazon Connect resource configuration screen

5. In the next step, leave the **Incoming calls** checkbox checked, and uncheck the **Outbound calls** checkbox:

> **Note**
>
> We have no requirement for our bot to initiate a call to anyone, in this case, but we do want it to be able to receive calls.

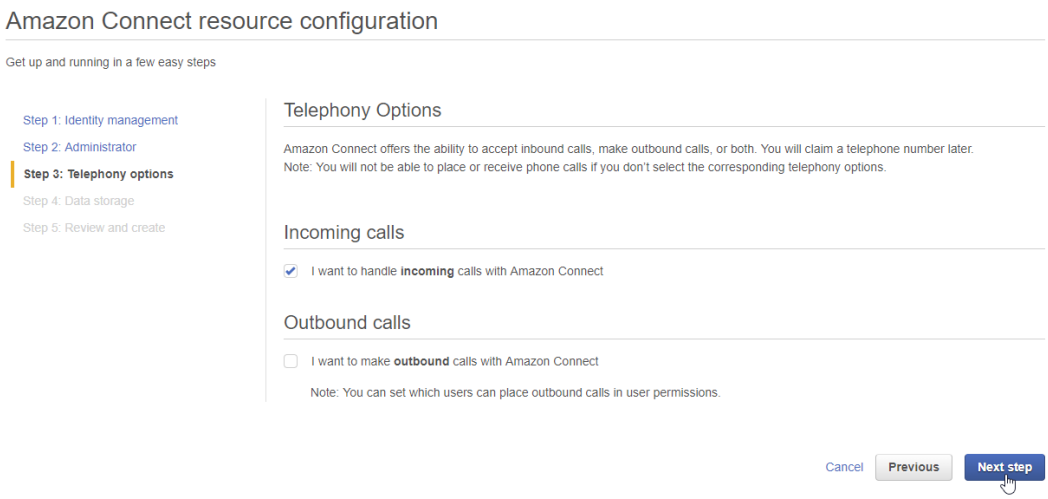


Figure 5.5: Step 3 of the Amazon Connect resource configuration screen

6. Click the **Next step** button when you are done.
7. Next you will be presented with an informational screen, letting you know that you are granting the call center application permissions to read and write data to or from your S3 bucket, encrypt or decrypt data, and read and write to and from **`CloudWatch`** logs.
8. It will also show you the location of the **`S3 buckets`** where your **`data`** and **`Contact flow`** logs will be stored:

> **Note**
>
> Note that there is a **Customize** settings link to further customize these locations, but we will not use this.

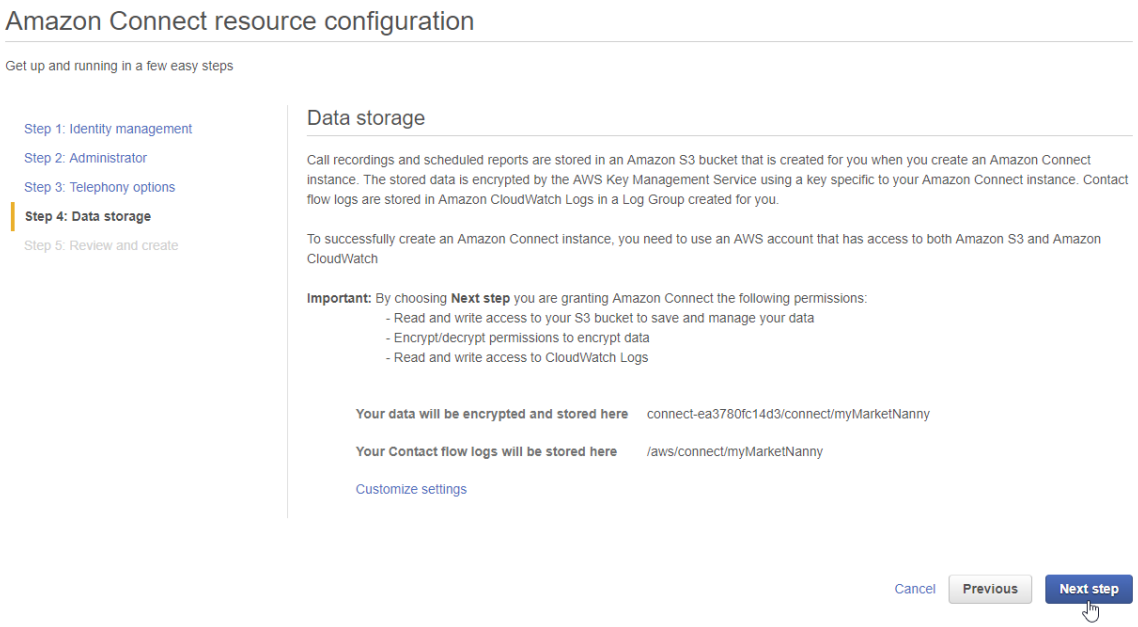


Figure 5.6: Step 4 of the Amazon Connect resource configuration screen

9. This will bring you to the final screen, which will provide you with a review of all of the settings for your **Contact center** application:

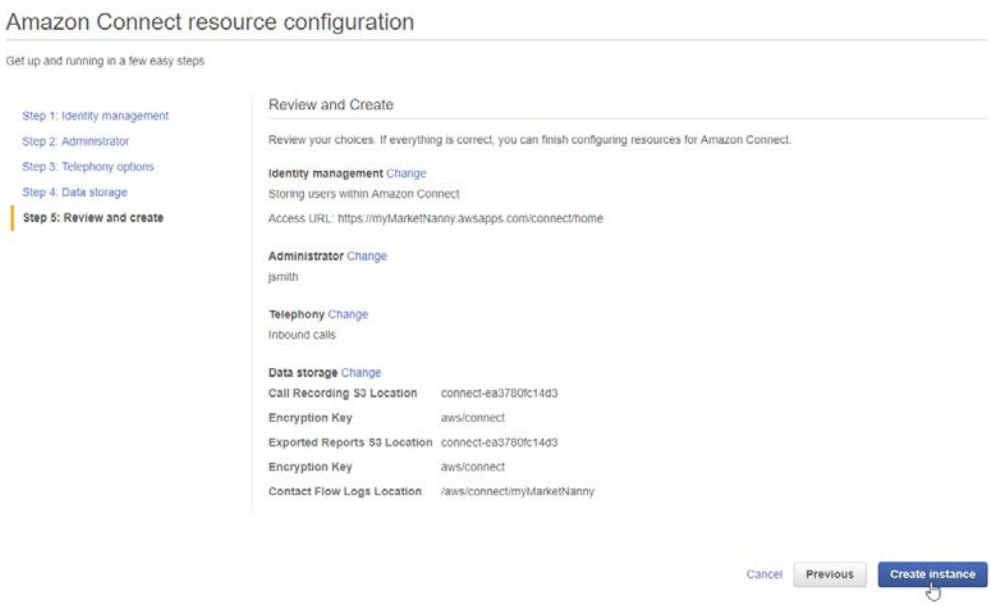


Figure 5.7: Step 5 of the Amazon Connect resource configuration screen

10. Now, click on the **Create Instance** button to create your application. You should see a dialog window:

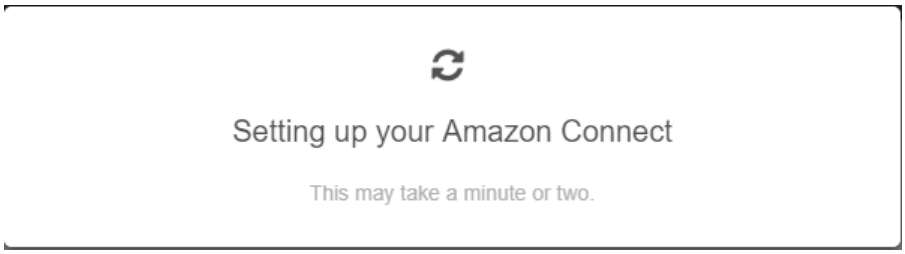


Figure 5.8: Setting up Amazon Connect

11. In a minute or two, it will complete the setup process and take you to a **Success** screen, where you can click on the **Get started** button to go directly to your application:

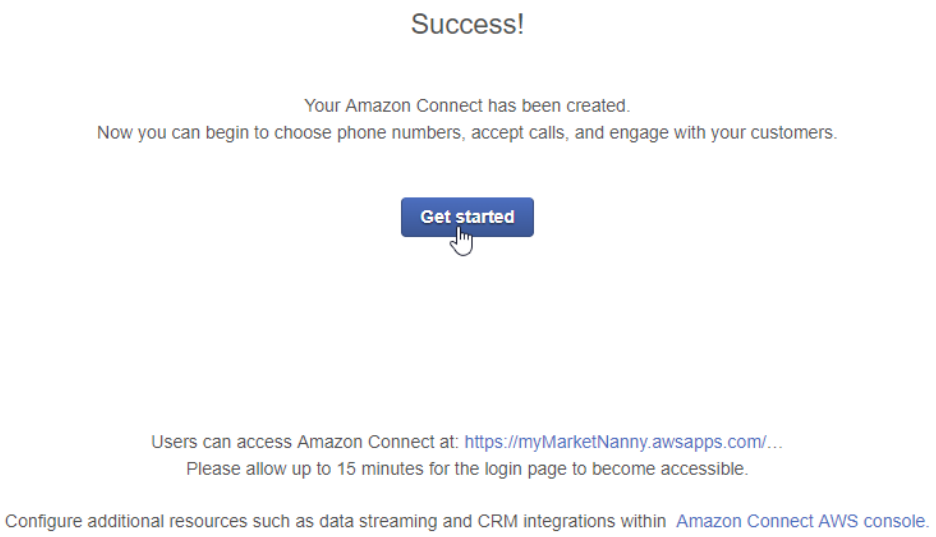


Figure 5.9: The Success screen

> **Note**
>
> It may take up to 15 minutes for the application to get set up, so if you do not succeed in getting to the application page initially, have some patience and try again every once in a while until it does succeed.

12. Typically, it should not take very long, and generally, the application page will be accessible right away.

Exercise 19: Obtaining a Free Phone Number for your Call Center

In this exercise, we will acquire a free number for our custom call center. The free telephone number can be acquired by the free tier services provided by Amazon Connect. The following are the steps for completion for obtaining the free number:

1. The welcome screen for your personal call center should be visible, and, if it is not, you can access it easily at the **URL** displayed within the previous screen, right before you click on the **Get Started** button:

> **Note**
>
> In my case, it was https://mymarketnanny.awsapps.com/connect; in yours, it will be based on your own application's name.

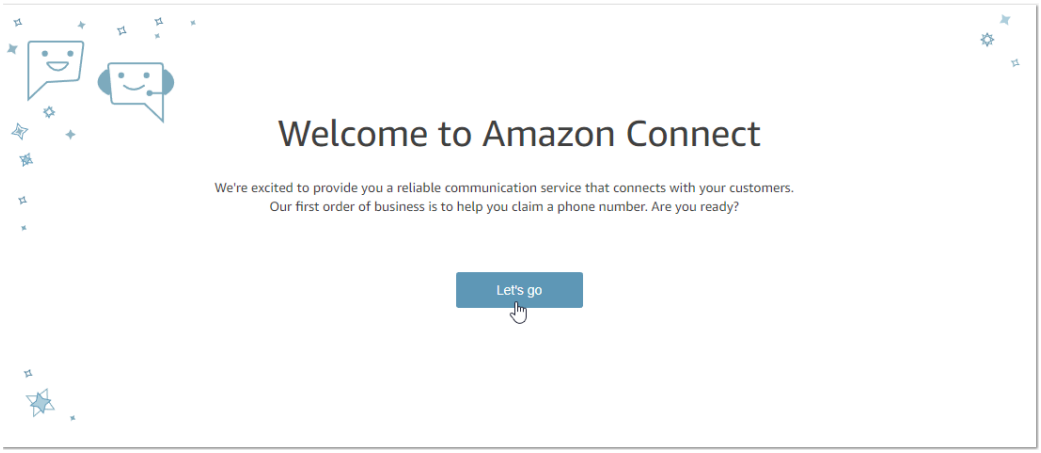


Figure 5.10: The Amazon Connect welcome screen

2. At this point, you can click on the **Let's go** button, in order to initiate the first step, which is to claim a **local** phone number for your call center application. On this screen, you can select your country:

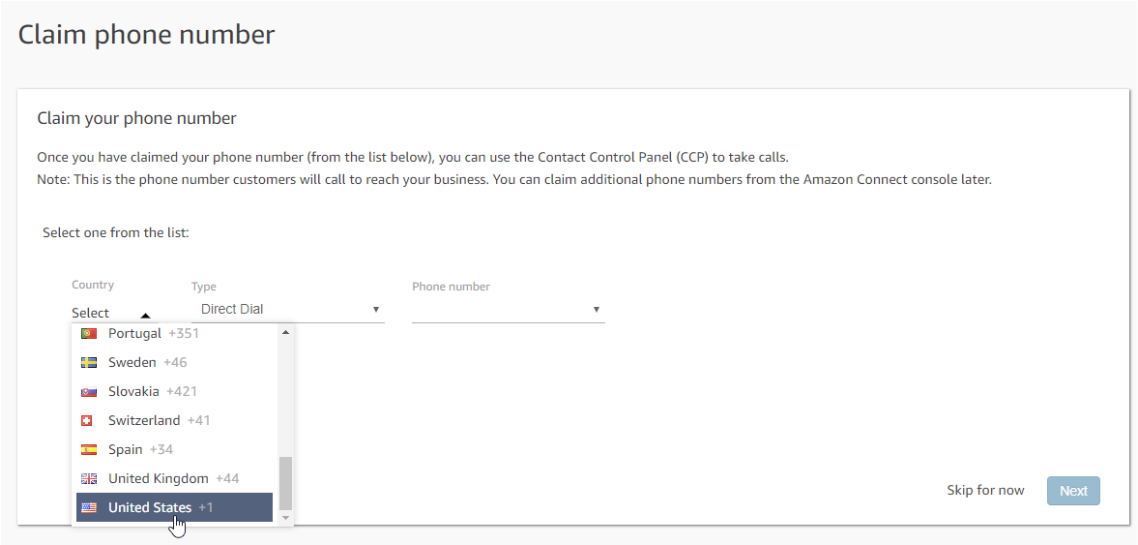


Figure 5.11: The Claim phone number screen

3. Leave the **Type** as the **default** (Direct Dial) option, and, from the Phone number drop-down menu, select a phone number. This should be, in most cases, a number that is local to you. If you choose one that is not, you may be charged, based on the cost to dial the number:

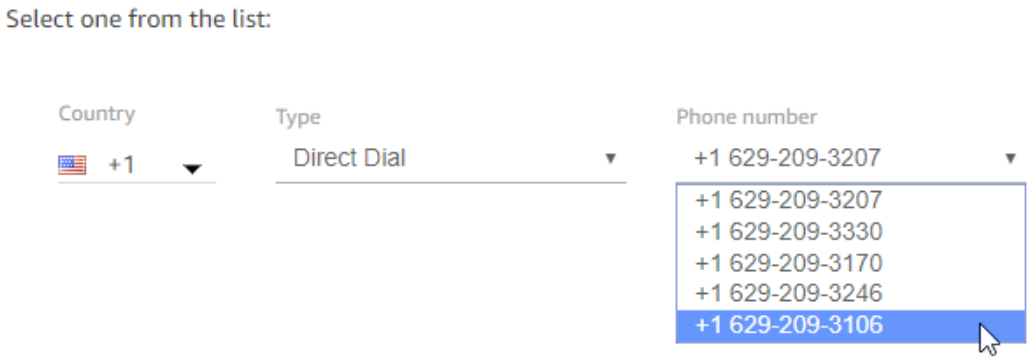


Figure 5.12: Phone number selection

4. Click on the **Next** button when you are done. This will allow you to test your new phone number by dialing it. You should hear a message that starts with **Hello**, thanks for calling. These are some examples of what the Amazon Connect virtual contact center can enable you to do
5. This will be followed by a list of options; you can try them for yourself to see what they do when you have some time. For now, click on the **Continue** button, which will bring you to the **Dashboard** screen:

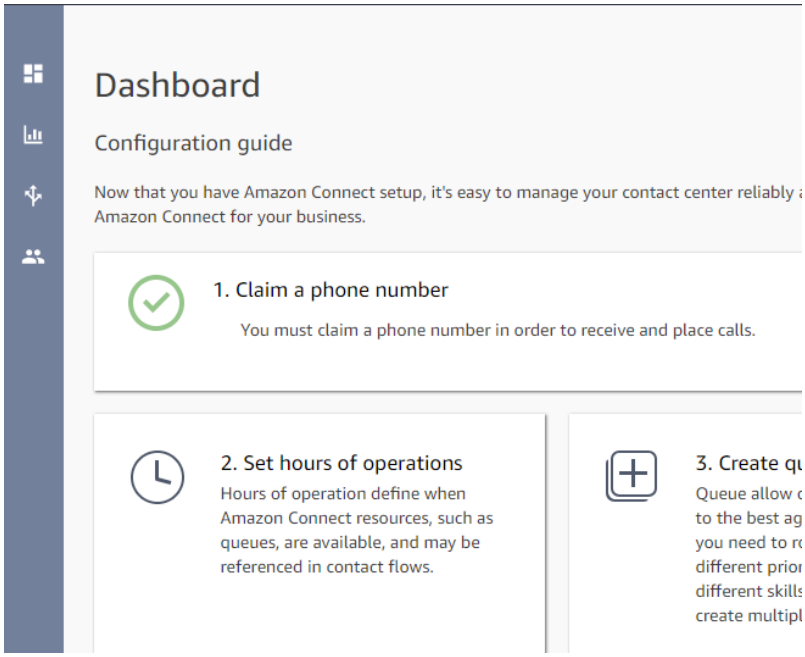


Figure 5.13: The Amazon Connect Dashboard screen

6. You now have a working call center application on Amazon Connect.

Using Amazon Lex Chatbots with Amazon Connect

With Amazon Lex, you can create intelligent chatbots. This can be done by converting contact flows into regular conversations. The Amazon Connect service compliments the forementioned feature by enabling the creation of a contact center (cloud base) along with the generation of dynamic flows. This delivers a personalized experience to your callers. Additionally, **Dual-tone multi-frequency** (**DTMF**) digits entered by callers during a call can be easily recognized by these chatbots.

With both Amazon Lex and Amazon Connect working in sync, the chatbot can gauge the callers' intent. Furthermore, you can use the AWS Lambda function along with your chatbot for processing caller requests, providing information to callers, and querying applications. These chatbots are capable of maintaining context by intelligently managing dialogue and review replies based on callers' responses on the fly.

In the next section, we will connect your *MarketNanny* chatbot to the call center, so that the user can speak into the phone and have his or her speech automatically converted to a text utterance that will get sent to the Lex chatbot.

Understanding Contact Flows

A **contact flow** defines each step of the experience that customers have when they interact with your call center. You can create a contact flow by using the contact flow templates provided. You can also create your own contact flow from scratch, by using the **Create** contact flow editor.

The **Contact** flow designer screen provides a way to visually create contact flows in a drag and drop environment. Contact flows are made of nodes that can be accessed in the left panel, within the designer area (under the Show additional flow information link).

Nodes are categorized under the `Interact`, `Set`, `Branch`, `Integrate`, and `Terminate/Transfer` categories. Nodes in each category can be hidden or revealed by clicking on the drop-down icon next to the section name. You will explore some of the nodes that are available by creating the contact flow for your chatbot.

Contact flows are a way for users (typically supervisors) to progressively refresh the settings for each call entering the framework and ensure that the users hear customized and pertinent choices. While there are a number of predesigned contact flows, you will create a new one for your `MarketNanny` chatbot.

Contact Flow Templates

The following are the templates that are available:

- **Customer queue flow**: Manages what the customer experiences while in that queue, before being joined to an agent.
- **Customer hold flow**: Manages what the customer experiences while the customer is on hold.
- **Customer whisper flow**: Manages what the customer experiences as a part of an inbound call,immediately before being joined with an agent.

- **Outbound whisper flow**: Manages what the customer experiences as a part of an outbound call, before being connected with an agent.
- **Agent hold flow**: Oversees what the specialist encounters when on hold with a client. With this stream, at least one sound prompt can be played to a specialist utilizing the Loop prompts square while the client is on hold.
- **Agent whisper flow**: Manages what the agent experiences as a part of an inbound call, immediately before being joined with a customer.
- **Transfer to agent flow**: Manages what the agent experiences when transferring to another agent.
- **Transfer to queue flow**: Manages what the agent experiences when transferring to another queue.

In the next section, we will set up your call center so that it can accept incoming calls, connect your `MarketNanny` chatbot to the call center so that the user can speak into the phone, and have the user's speech automatically converted to a text utterance that will be sent to the Lex chatbot.

The chatbot then continues with processing the utterance, finding the intent, and slots and fulfilling it through the Lambda function you had set up in the preceded section.

In order to connect your call center Instance with your chatbot, you will need to go back to the main screen (not the Dashboard) briefly. Select the link for the `Contact` flows.

Exercise 20: Connect the Call Center to Your Lex Chatbot

In this exercise, you will set up your call center so that it can accept incoming calls, and so that you can connect your `MarketNanny` chatbot to the call center. The following are the steps for completion:

> **Note**
>
> This is not the previous screen with the Dashboard. You may find a Contact flows entry there, as well, but it is not the one we're interested in.

1. Here, you will find a section titled Amazon Lex, with a drop-down under Lex bots. Click on the dropdown to find the **`MarketNanny`** bot as an entry, and select it, and then select the **Save** Lex Bots link to save it:

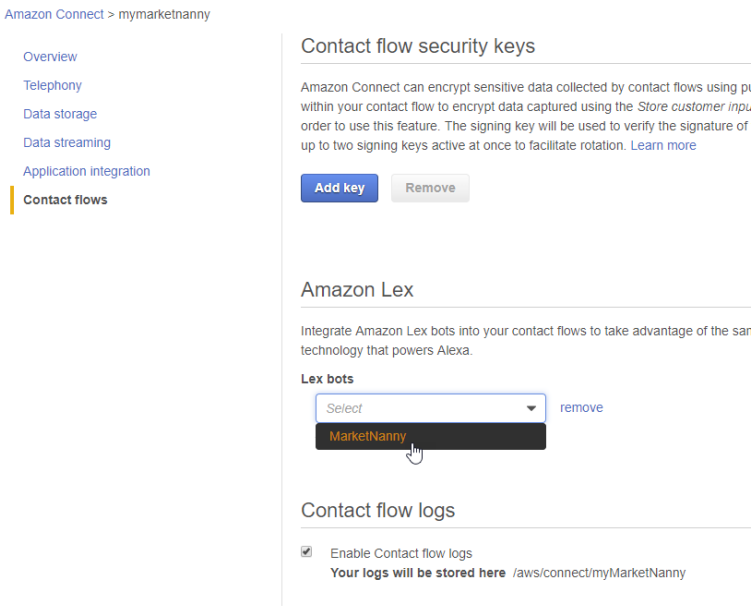


Figure 5.14: The Contact flows screen with the Lex bots selection dropdown

2. Now, you can navigate back to the **Dashboard** screen by selecting the **Overview** link and clicking on the Login as **administrator** button. The **Dashboard** screen will open in a new browser tab:

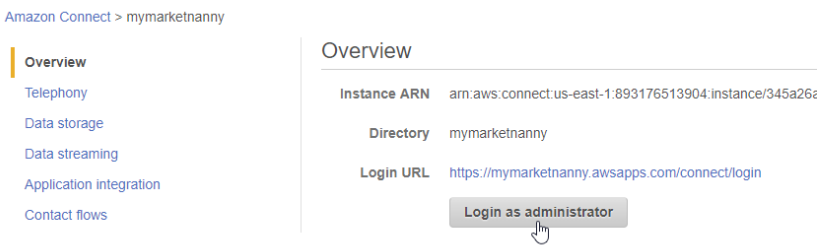


Figure 5.15: The Login as administrator screen

3. On the Dashboard screen, select the **Routing** icon, and when it opens, select **Contact flows**:

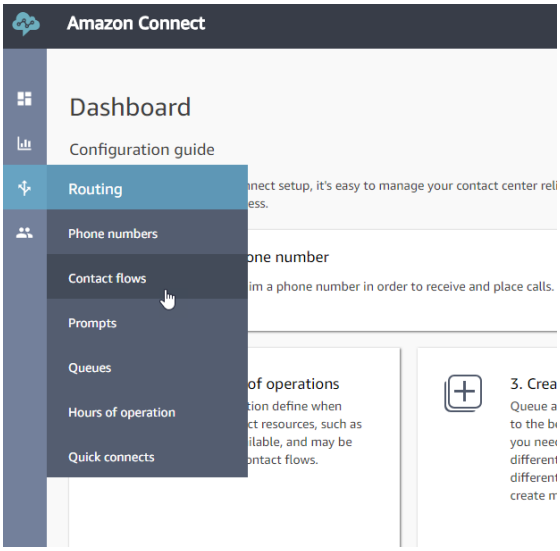


Figure 5.16: Selecting Routing and Contact flows from the Dashboard

4. This will bring you to a **Contact flows** screen than the one we were just on previously. This screen is the **Contact flow designer**, where you can design and create **New** contact flows:

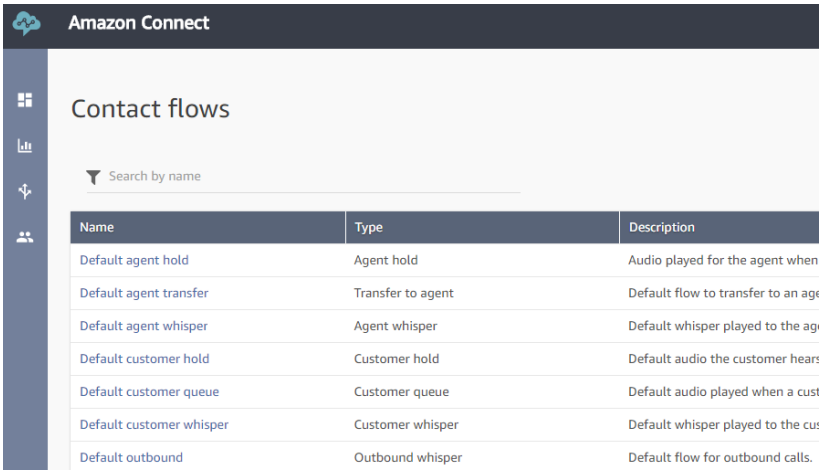


Figure 5.17: Contact flows screen

5. Click on the **Create** contact flow button in the upper right of the page to create a new contact flow. Name it **`MarketNannyFlow`**:

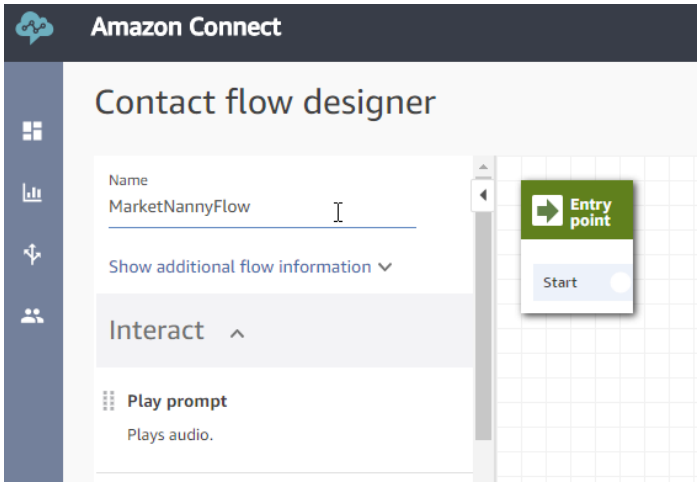


Figure 5.18: The Contact flow designer screen

6. The first node that you will use for your chatbot's contact flow is the **Set voice** node. You can find it under the **Set** category and drag it to the graphical view to the right of the **Entry point** node, which is already in the view.

7. Connect the output of the Entry point node with the input of the **Set voice** node by clicking on the output circle and dragging with your mouse to the input bump on the left of the **Set voice** node:

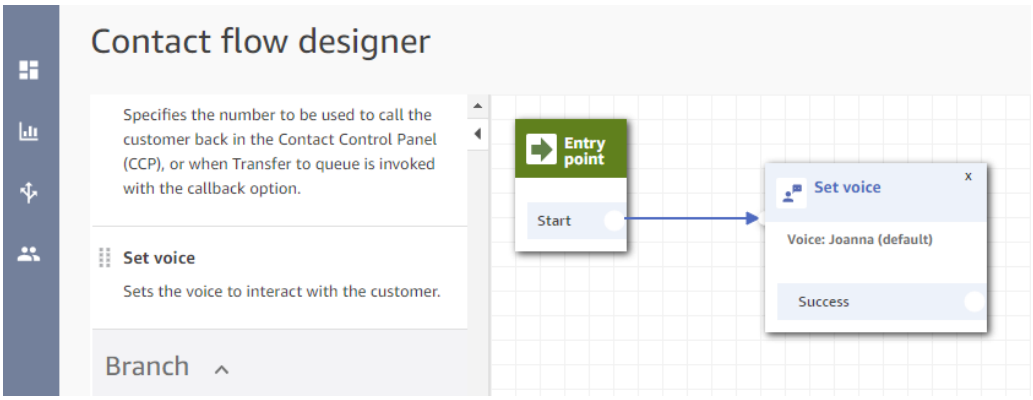


Figure 5.19: The Contact flow designer screen with Set voice node

8. Next, you can set the properties of the node by clicking on the top of the node. This will bring up a panel on the right-hand side of the screen, with the relevant properties that you can set.

9. For this component, it provides **Language** and **Voice** properties, both of which can be set by selecting a value from the dropdown under each property label. You can set the values appropriate to your scenario, and then click on the **Save** button:

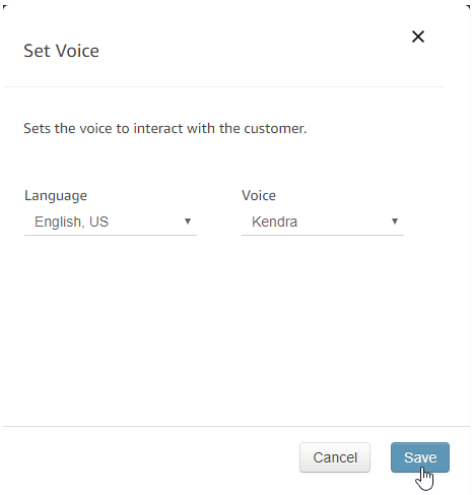


Figure 5.20: Properties for Set Voice node

10. The next node to add is the **Get customer input** node, which you will find under the Interact tab. You can connect the input of this node to the output of the **Set voice** node, just like when you connected **Set voice** to the Entry point node:

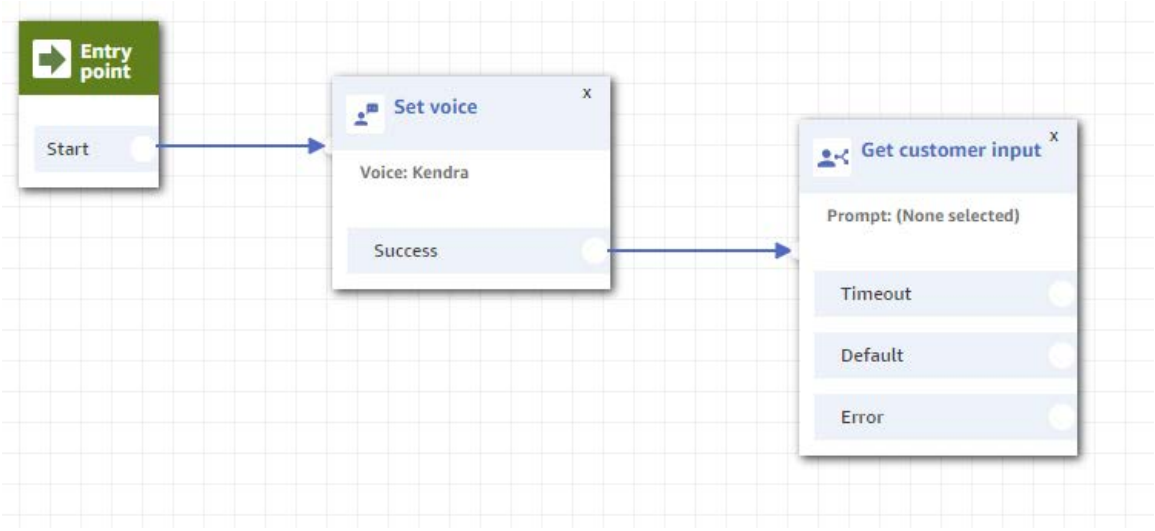


Figure 5.21: Adding the Get customer input node

11. First, select the **Text to speech** (ad hoc) option and click on **Enter** text under it. In the input textbox, enter a greeting such as the following:

```
Hi this is the Market Nanny chatbot. How may I help you today?
```

> **Note**
>
> Feel free to replace this with some other text that may be more interesting or appropriate for you.

12. This is the text that will be converted to a voice greeting when the **Contact center** application first answers the phone:

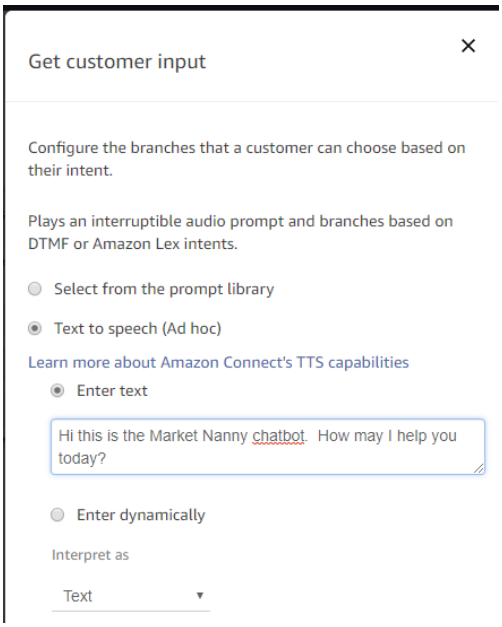


Figure 5.22: Get customer input properties

13. Next, scroll down the setting pane and select the Amazon Lex subsection, and not the **DTMF** section. Here, you will be able to configure the node to work with your chatbot.

> **Note**
>
> The DTMF section provides a way to specify interactions with the caller via button presses on his/her phone, but we will not cover that today. We are much more interested in having our chatbot interact with the caller!

14. In this section, under Lex bot, you can enter the name and alias for your chatbot. Under Name, enter **`MarketNanny`**, and under **Alias**, enter **`$LATEST`**.

15. **`$LATEST`** is a system-defined alias which specifies that the contact center will always access the most recently published version of the **`MarketNanny`** chatbot:

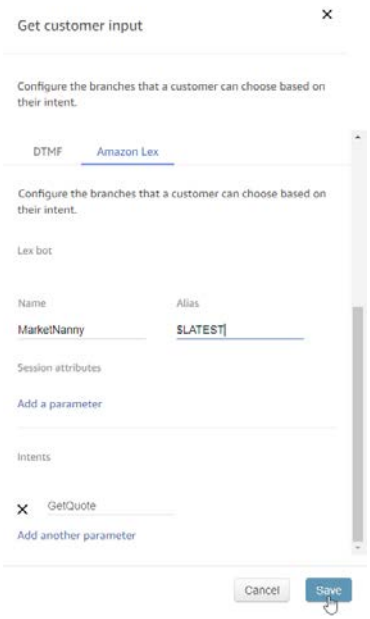


Figure 5.23: Amazon Lex properties in the Get customer input properties

16. Finally, specify the **`GetQuote`** intent under the Intents section. Since that is the only intent we are currently working with, click on the **Save** button:

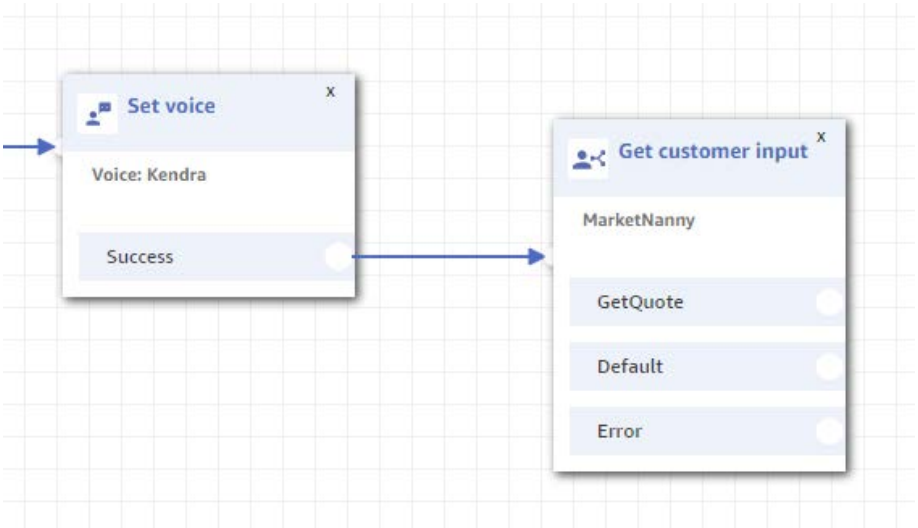


Figure 5.24: The GetQuote intent in Get customer input node

17. You can now see that the **Get customer input** node has changed to display the intent that we specified in its properties pane:

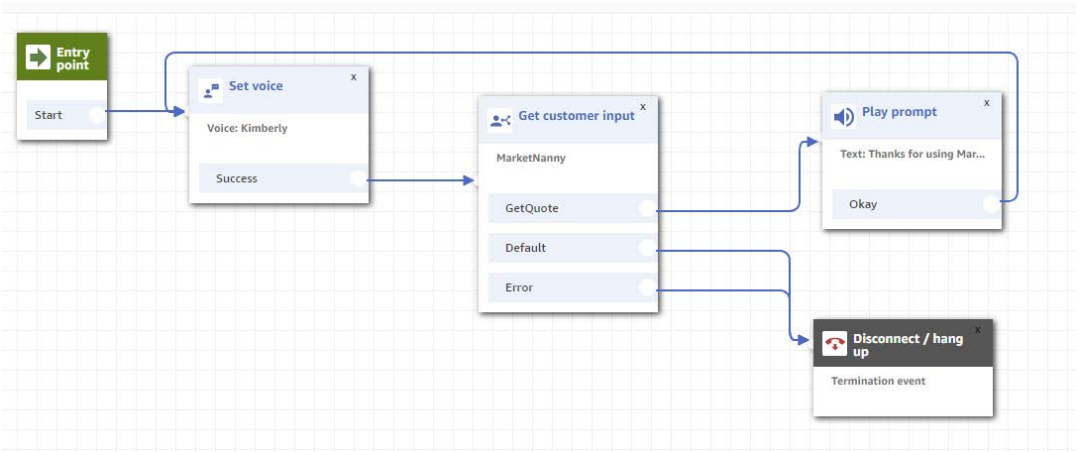


Figure 5.25: Full contact center flow

18. You can finish the flow by adding two more nodes:

- **`Play`** prompt from the **`Interac`**t section. Set the Prompt property to Text to speech (ad hoc) in the property pane, and enter text in the textbox under Enter text as follows:

```
"Thanks for using MarketNanny" Feel free to ask me something else.
```

- **`Disconnect/hang up`** from the **`Terminate/Transfer`** section. Do not set any properties.

19. Connect the nodes, as shown previously.

20. Connect the outputs of the Default and Error states in **Get customer input** to the input of the **Disconnect/Hang up** node.

21. Connect the **`GetQuote`** output to the Play prompt node.

22. Connect the output from **`Play`** prompt to the input of **Set** voice. This ensures that you can interact with your chatbot for as long as you keep asking the right questions.

23. Finally, click on the dropdown icon next to the **Save** button in the upper right of the screen, and select **Save and Publish** to publish the contact flow:

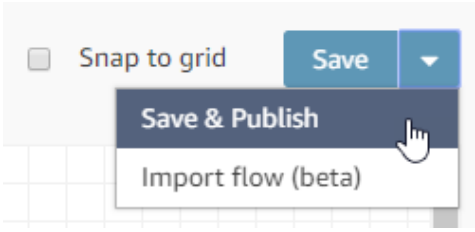


Figure 5.26: Save and Publish section

24. As the final step, you will connect your call center phone number to the new contact flow you have just created.

25. From the menu panel on the left-hand side of the screen, select the **Routing** menu, and from that, select the **Phone numbers** item:

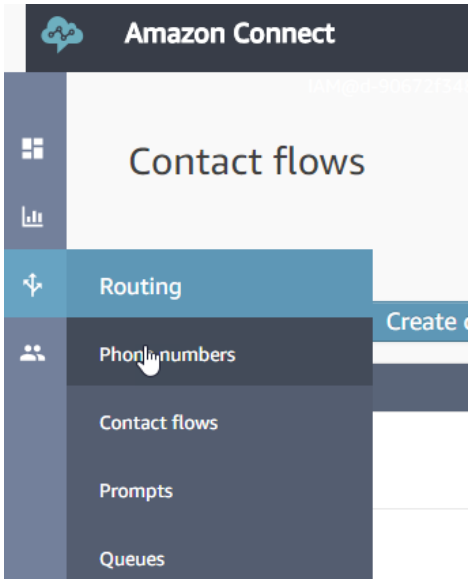


Figure 5.27: Selecting Routing and Phone numbers from the Dashboard

26. This will bring up the **Manage Phone** numbers screen, where you can select the phone number that you wish to use for `MarketNanny`.

27. Selecting the phone number will bring you to the Edit Phone number page:

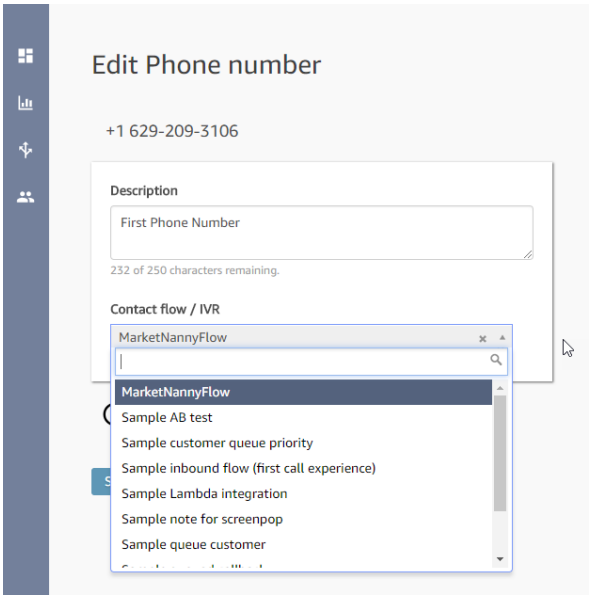


Figure 5.28: The Edit Phone number screen

28. Here, click on the dropdown under **Contact flow/IVR** and select or type the name of your new contact flow: `MarketNannyFlow`.

29. Call your contact center phone number to verify that you can interact with your `MarketNanny` chatbot by speaking to it!

Activity 1: Creating a Custom Bot and Connecting the Bot with Amazon Connect

In this activity, we will create an **Amazon Lex** bot for checking the **Account Balance**. The bot will check the account balance. An **Amazon Lex** bot is controlled via **Automatic Speech Recognition** (**ASR**) and **Natural Language Understanding** (**NLU**) capacities, a similar innovation that forces Amazon Alexa. For this, we will consider the following example: suppose a user wants to check their account balance; a user might use **I want to check my account balance**. You can configure the intent to of the Amazon Lex bot to an Amazon Connect Instance. You can create a contact flow and add your Amazon Lex bot. To ensure that this happens correctly, you will need to navigate to the Amazon Lex services and Amazon Connect services for connecting the bot:

1. Create an Amazon Lex bot.
2. Configure the bot by creating intents and adding slots.

3. Configure the bot and configure the bot error handling such as when the caller's utterance is not understood.
4. Build and test the bot.
5. Publish the bot and create an alias.
6. Add the bot to your Amazon Connect.
7. Create a contact stream and include the bot.
8. Assign the contact flow to a telephone number.

> **Note**
>
> To refer to the detailed steps, go to the *Appendix A* at the end of this book on Page no. 225

Summary

In the first part of this chapter, you learned about the basics of Amazon Connect, and even looked in the free tier version of Amazon Connect. Later, we explored the sample voice-based Chatbot application using Amazon Connect, Lex, and Lambda. Next, we created a personal call center.

You have also learned how to obtain a free phone number for a personal call center. We have also learned about using Amazon Lex Chatbots with Amazon Connect with contact flows and different templates. Finally, you learned how to connect the call center to your Lex Chatbot.

In next chapter, you will learn how to analyze images with computer vision, using Amazon Rekognition services. Amazon Rekognition services are used to detect the objects, scenes, and text in. It also used to match the faces in different images and compare them how closely both images are interlinked with each other.

6

Analyzing Images with Computer Vision

Learning Objectives

By the end of this chapter, you will able to:

- Use the Rekognition service for image analysis using computer vision.
- Detect objects and scenes in images.
- Detect the need for content moderation in images.
- Analyze faces and recognize celebrities in images.
- Compare faces in different images to see how closely they match with each other.
- Extract text from images.

This chapter describes the Amazon Rekongnition service for analyzing the content of the images using various techniques.

Introduction

In the preceding chapters, you created a chatbot using the Amazon Lex service, named **`MarketNanny`**. This chatbot receives input from you about a stock ticker symbol and then, relays that information to a Lambda function, which then goes out to an external **`API`** and retrieves that information. The Lambda function then sends this information back to the chatbot, which in turn provides it to you by means of conversational text and speech interfaces.

You have already used a text interface before, and in the following chapter you learned how to set up a voice-based call center using the Amazon Connect service. You created a contact flow and populated it with the appropriate nodes and properties for those nodes to connect to your **`MarketNanny`** chatbot and allow anyone to converse with the chatbot through a voice interface that's available by dialing a local phone number!

In this chapter, you will use the **Amazon Rekognition service** to perform various image processing tasks. First, you will identify objects and scenes within images. Then, you will test whether images may be flagged as needing content moderation. Next, you will analyze faces using Rekognition. You will also recognize celebrities and well-known people in images. You will compare faces that appear in different images and settings (for example, in groups or in isolation) and recognize the same people in different images. Finally, you will extract text from images that might have some text displayed in them.

Amazon Rekognition Basics

Amazon Rekognition is a service from AWS that allows you to perform image analysis using machine learning. It is built on the same scalable infrastructure as AWS itself, and uses deep learning technology to be able to analyze billions of images daily if required. It is also being updated constantly by Amazon and learning new labels and features.

Some of the use cases for Amazon Rekognition are as follows:

- Searching across your library of image content with text keywords.
- Confirm user identities by comparing live images with reference ones.
- Analyze trends based on public images, including the sentiments and emotions of the people in the images.

- Detect explicit and suggestive content and automatically filter it for your purposes
- Retrieve text from images.

> **Note**
>
> Amazon Rekognition is also an **HIPAA** Eligible Service for healthcare applications. If you wish to protect your data under HIPAA, you will need to contact Amazon Customer Support and fill out a Business Associate Addendum (BAA). For more information about HIPAA, go to the following link: https://aws.amazon.com/compliance/hipaa-compliance/.

Free Tier Information on Amazon Rekognition

In this book, you will only be using the free tier of the services presented. However, it is important to be aware of the limits of free usage and pricing. For image processing, you can receive the following services for free:

- Upon joining, new Amazon Rekognition clients can break down up to 5,000 pictures initially a year.
- You can utilize all Amazon Rekognition APIs with this complementary plan, and furthermore put away up to 1,000 faces free of charge.

> **Note**
>
> You should not need to use more than, if you go beyond the limits of the free tier, you will get charged by Amazon at rates published at this link: https://aws.amazon.com/rekognition/pricing/.

Rekognition and Deep Learning

Deep learning is a significant sub-field of machine learning and a branch of artificial intelligence. The way in which deep learning works is by inferring high-level abstractions from raw data by using a deep graph with many layers of processing.

Deep learning structures, for example, **Convolutional deep Neural Networks** (**CNNs**), and **Recurrent Neural Networks** (**RNNs**) have been connected to computer vision, speech recognition, natural language processing, and audio recognition to deliver results.

The Rekognition service employs deep learning in order to provide its various features behind the scenes. It uses pre-trained models so that users do not have to train the system. The exact details are proprietary and confidential to Amazon, but we can see how it works and how to use Rekognition in this chapter.

Detect Objects and Scenes in Images

Amazon Rekognition provides a feature that can detect objects and scenes in an image and label them. The label may be an object, scene or concept such as **Person**, **Water**, **Sand**, **Beach** (scene) and **Outdoors** (concept).

Each label comes with a confidence score, which measures, on a scale from 0 to 100, the probability that the service got the correct answer for that label. This allows you or your application to judge for itself the threshold against which to allow or discard results.

Rekognition supports thousands of labels from categories such as those shown in the following table:

Category	Example Labels
People and Events	Wedding, Bride, Baby, Birthday Cake, Guitarist
Food and Drink	Apple, Sandwich, Wine, Cake, Pizza
Nature and Outdoors	Beach, Mountains, Lake, Sunset, Rainbow
Animals and Pets	Dog, Cat, Horse, Tiger, Turtle
Home and Garden	Bed, Table, Backyard, Chandelier, Bedroom
Sports and Leisure	Golf, Basketball, Hockey, Tennis, Hiking
Plants and Flowers	Rose, Tulip, Palm Tree, Forest, Bamboo
Art and Entertainment	Sculpture, Painting, Guitar, Ballet, Mosaic
Transportation and Vehicles	Airplane, Car, Bicycle, Motorcycle, Truck
Electronics	Computer, Mobile Phone, Video Camera, TV, Headphones

Figure 6.1: Labels supported by Amazon Rekognition

Additionally, Amazon is continuously training the system to recognize new ones, and you can request labels that you might wish to use which are not in the system through **Amazon Customer Support**.

To create an Amazon Rekognition of a sample image, you can do the following:

1. Navigate to the Amazon Rekognition service web page in the AWS Console:

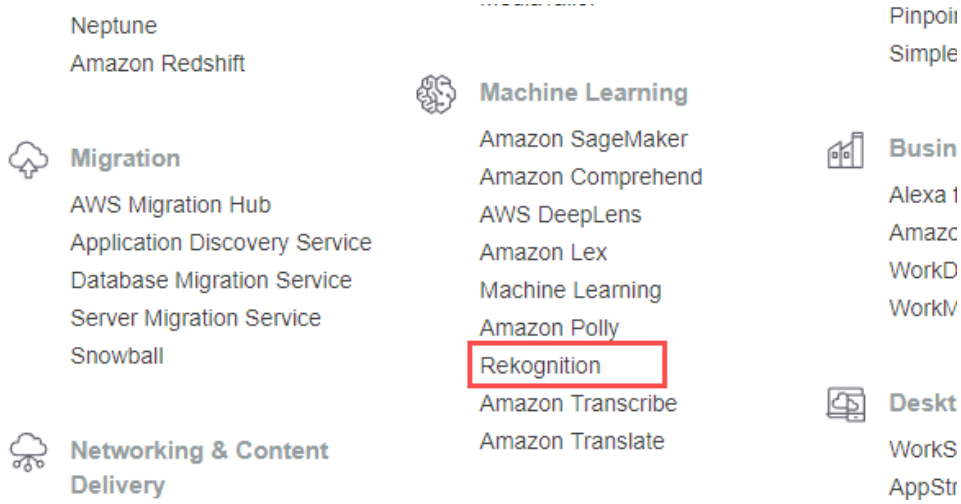


Figure 6.2: Selecting the Rekognition service in the AWS Console.

2. You can find Rekognition under the **Machine Learning** section. When you are on the Rekognition page, click on the Object and scene detection link in the left hand side toolbar to navigate to the **Object and scene detection** page.
3. Click on the **Object and Scene detection** link in the toolbar to navigate to the **Object** and **Scene detection** page.
4. Next, choose the **Text box** under the **Use your own image** panel. https://images.unsplash.com/photo-1540253208288-6a6c32573092?w=800



Figure 6.3: Use image URL input text box.

The result for the image is as follows:

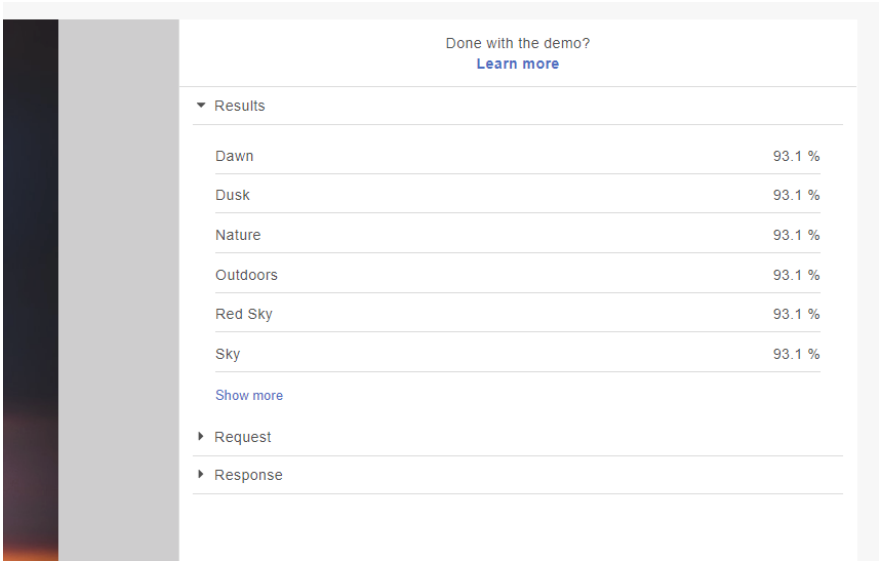


Figure 6.4: Results for the sample image.

5. You can see that within the image, the objects that have been detected with a 93.1% confidence are as follows:

- Dawn
- Dusk
- Nature
- Outdoors
- Red Sky
- Sky

6. Click on the **Show more** link to show more results with lower confidence levels. This will show more objects have been detected:



Figure 6.5: Full set of labels for sample image.

7. You can choose the threshold amount of the confidence level at which you would like to cut off the results for your own application.

Exercise 21: Detecting Objects and Scenes using your own images

In this exercise, we will detect objects and scenes of custom images using Amazon Rekognition. The custom images can be referred from site, or you can upload them from your local machine. The following are the steps for detecting objects and scenes:

1. Navigate to the Amazon Rekognition service page in the AWS Console or go directly to the URL https://console.aws.amazon.com/rekognition.
2. Click on the **Object and Scene detection** link in the toolbar to navigate to the **Object and Scene detection** page.
3. Next, choose the **Text box** under the **Use your own Image panel**.
4. Enter the following URL so that you have an image to analyze: https://images.unsplash.com/photo-1522874160750-0faa95a48073?w=800.

5. The following is the image:



Figure 6.6: The first image that's provided.

> **Note**
>
> We have collected images from a stock photo site called https://unsplash.com/ , which has photos that are available for free to download and use without restrictions for the purpose of this book. Always obey copyright laws and be mindful of any restrictions or licensing fees that might apply in the jurisdiction where you reside (if applicable). You may view the license for images from unsplash.com here: https://unsplash.com/license .

6. You may view the results of the object detection under the **Results** panel on the right-hand side of the image. In this case, it is an image of a camera, and the results should look as follows:

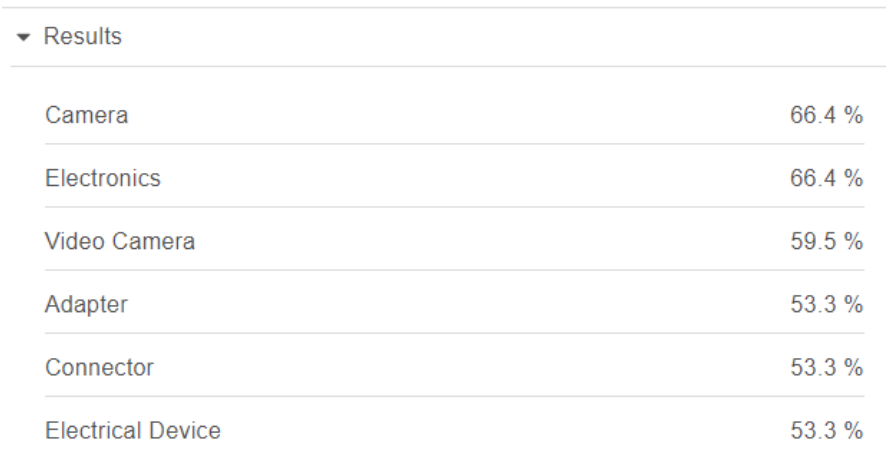


Figure 6.7: Results for the first image provided.

7. As you can see, the results are quite accurate. Next, you can try the following images and verify that the results are as shown in the tables that immediately follow each image.https://images.unsplash.com/photo-1517941875027-6321f98198ed?w=800 and https://images.unsplash.com/photo-1500111709600-7761aa8216c7?w=800

8. The following is the image:



Figure 6.8: The second image provided.

Results

Jewelry	83.7 %
Ornament	83.7 %
Ring	83.7 %
Human	60.7 %
Person	60.7 %
Accessories	51.2 %

Figure 6.9: Results for the second image provided.



Figure 6.10: The third image provided.

Results

Arch	98.3 %
Arch Bridge	98.3 %
Arched	98.3 %
Architecture	98.3 %
Bridge	98.3 %
Building	98.3 %

Figure 6.11: Results for the third image provided.

Image Moderation

In addition to object and scene detection, Rekognition also provides the ability to filter out objectionable content. You can use moderation labels to give point by point sub-classifications, enabling you to tweak the channels that you use to figure out what sorts of pictures you consider satisfactory or shocking. **Amazon Rekognition Image** provides the `DetectModerationLabels` operation to detect unsafe content in images.

You can utilize this component to enhance photograph sharing destinations, gatherings, dating applications, content stages for youngsters, online business stages and commercial centers, and more. In this book, we will not use any adult or nude images, but we can show the use of this feature with content which may be considered racy or suggestive in some locales featuring women in revealing clothing such as swimsuits or club wear.

The images are blurred by default in this section so you do not have to view them unless you press the View Content button.

> **Note**
>
> If you find any racy or suggestive images offensive, please skip this section based on your own personal, moral, religious, or cultural norms.

Amazon Rekognition uses a hierarchical taxonomy to label categories of explicit and suggestive content. The two top-level categories are **Explicit Nudity** and **Suggestive**. Each top-level category has a number of second-level categories. The types of content which are detected and flagged using this feature are as follows:

Top-Level Category	Second-Level Category
Explicit Nudity	Nudity
	Graphic Male Nudity
	Graphic Female Nudity
	Sexual Activity
	Partial Nudity
Suggestive	Female Swimwear Or Underwear
	Male Swimwear Or Underwear
	Revealing Clothes

Figure 6.12: Content type categories

To create an **Image Moderation** of a sample image, you can do the following:

1. Navigate to the Amazon Rekognition service page in the AWS Console or go directly to the URL https://console.aws.amazon.com/rekognition.
2. Click on the **Image Moderation** link in the toolbar to navigate to the **Image Moderation** page.
3. Next, choose the **Text box** under the **Use your own image panel**.
4. Enter the following URL of an image to analyze: https://images.unsplash.com/photo-1512101176959-c557f3516787?w=800.
5. You will notice right away that the content is blurred. The image being analyzed is the first image to the left, which is already selected. You will see that the service has correctly identified Suggestive content in the image of type **Female Swimwear or Underwear** with a confidence of **98.4%**.

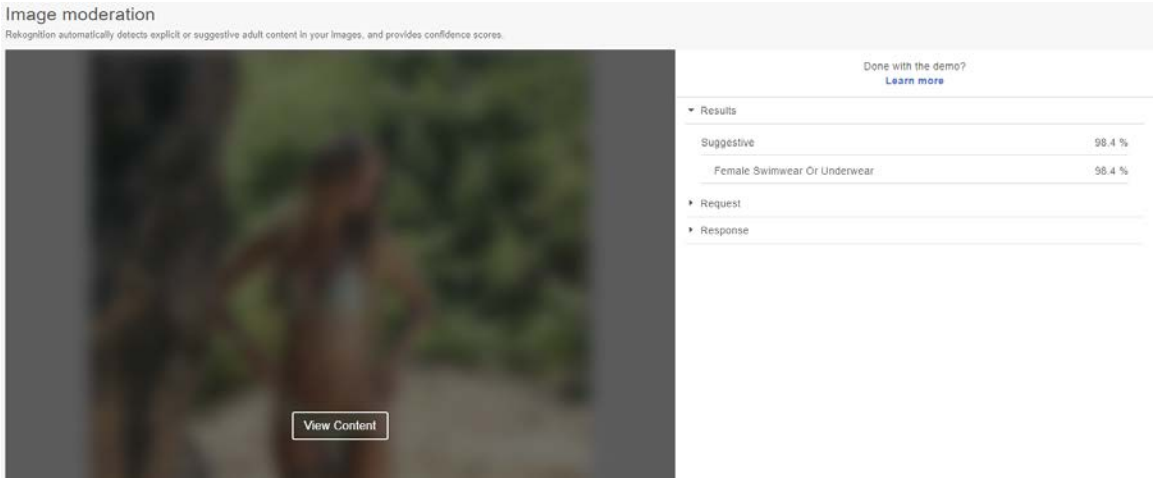


Figure 6.13: Results for Image moderation.

Exercise 22: Detecting objectionable content in images

In this exercise, we will detect objectionable content in images. Now, you can try this service on your own images. We have selected three images that we will try out with this feature:

1. Copy and paste or type the following `URL` into the **Use image URL** text box under the **Use your own image** section at the bottom of the page, and press the **Go** button to receive results from the service: https://images.unsplash.com/photo-1525287957446-e64af7954611?w=800.



Figure 6.14: Use image URL upload text box.

2. You should receive a result stating that the service has found Female Swimwear or Underwear content with a **98.4%** degree of confidence:

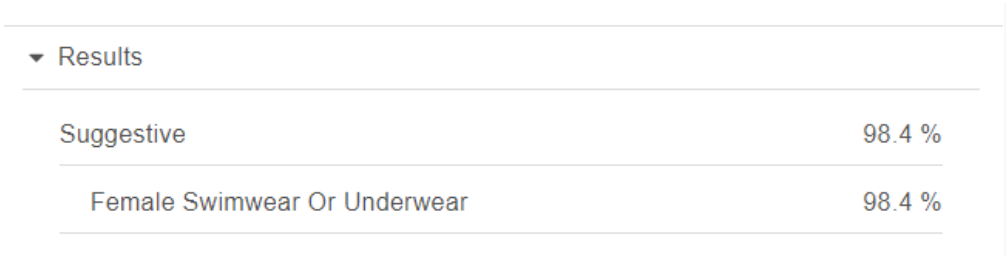


Figure 6.15: Result that the service has found

3. The following image `URL` to provide to the service is: https://images.unsplash.com/photo-1509670811615-bb8b07cb3caf?w=800.

4. Just as before, enter it in the **Use image URL** text box and press the **Go** button. This image has no objectionable content:



Figure 6.16: First provided image for Image moderation.

5. You should see that **Rekognition** correctly returns no results:

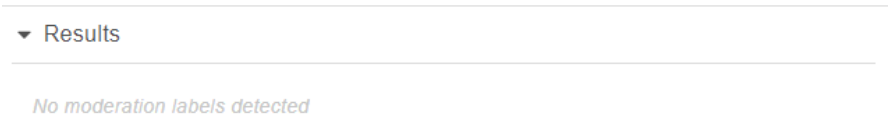


Figure 6.17: Results of the first provided image for Image moderation.

6. Finally, we will use an image that should, again, return some results: https://images.unsplash.com/photo-1518489913881-199b7c7a081d?w=800.

   This one should have again correctly been identified as containing content with Female Swimwear or Underwear with a **97.1%** degree of confidence:

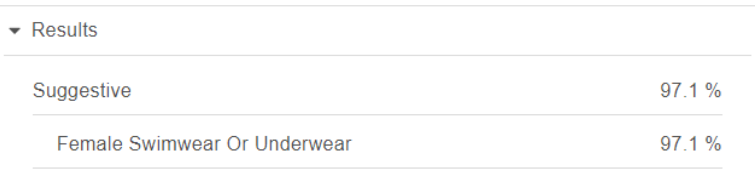


Figure 6.18: Results of the second provided image for Image moderation.

Facial Analysis

Rekognition can perform more detailed analysis on faces as well. Given an image with a detectable face, it can tell whether the face is male or female, the age range of the face, whether or not the person is smiling and appears to be happy, and whether they are wearing glasses or not.

It can also detect more detailed information such as whether the eyes and mouth are open or closed, and whether or not the person has a mustache or a beard.

To create a facial analysis of a sample image, you can do the following:

> **Note**
>
> Click on the Facial analysis link in the left toolbar to navigate to the Facial analysis page.

1. Choose the **Text box** under the **Use your own image** panel.
2. Enter the following URL of an image to analyze: https://images.unsplash.com/photo-1525943837837-af668e09139d?w=800.
3. For this image, you will see that the main image box displays a bounding rectangle which shows the region in which the face was detected. Within the bounding box, there are also three dots to identify the locations of key facial images: the mouth and nose detection. .

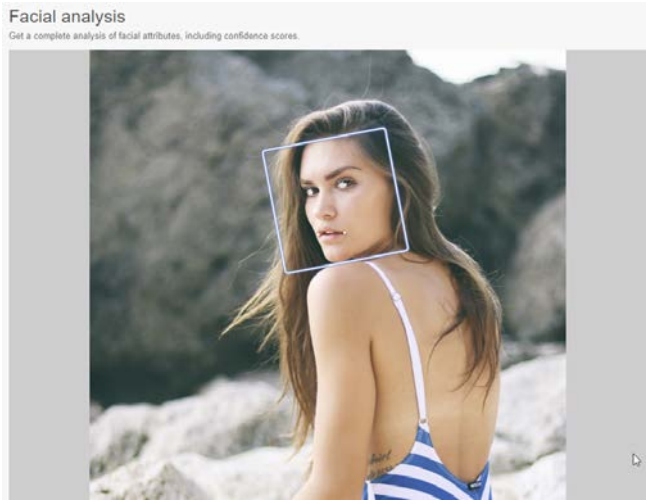


Figure 6.19: First sample image for Facial analysis.

4. Under the Results section to the right of the image, you can see that Rekognition has detected the following attributes of the face in the image:

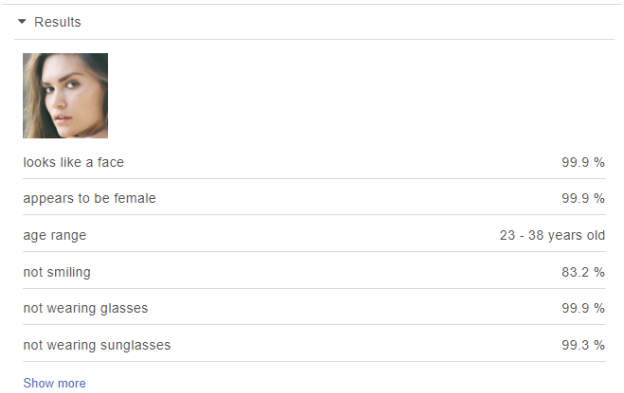


Figure 6.20: Results of the first sample image for Facial analysis.

5. You can click on the **Show** more link in order to show the other attributes which have also been identified.
6. All these identified qualities have an extremely high degree of confidence, showing that the service is very confident of its findings.

Exercise 23: Analyzing Faces in your Own Images

In this exercise, you have been provided with links to three images so that you can try out the service with your own images:

> **Note**
>
> Click on the **Facial analysis** link in the left toolbar to navigate to the **Facial analysis** page.

1. The first image to be analyzed is at the following URL: https://images.unsplash.com/photo-1494790108377-be9c29b29330.

2. You can copy and paste or type this `URL` into the **Use image URL** text box under the **Use your own image** section at the bottom of the page, like so:



Figure 6.21: The image URL input text box.

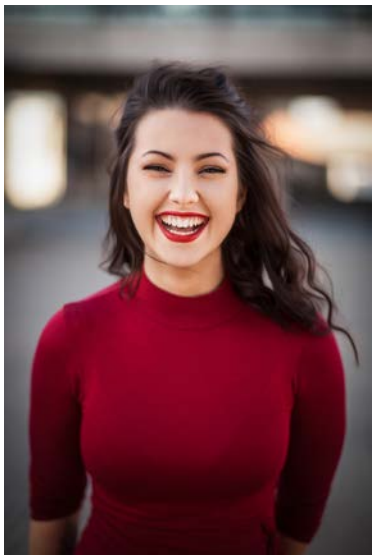


Figure 6.22: First image provided for Facial analysis.

3. You can see from the bounding box and dots that Rekognition is able to recognize the face easily:

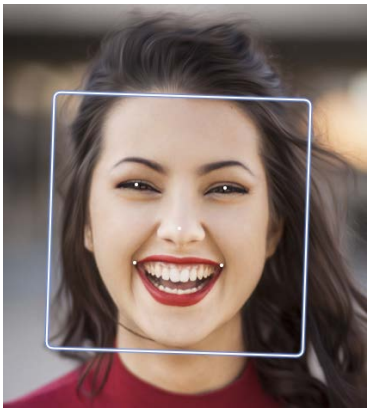


Figure 6.23: Bounding box for first image provided for Facial analysis.

4. Under the **Results** section, it quite accurately displays the attributes of the face in the image as well:

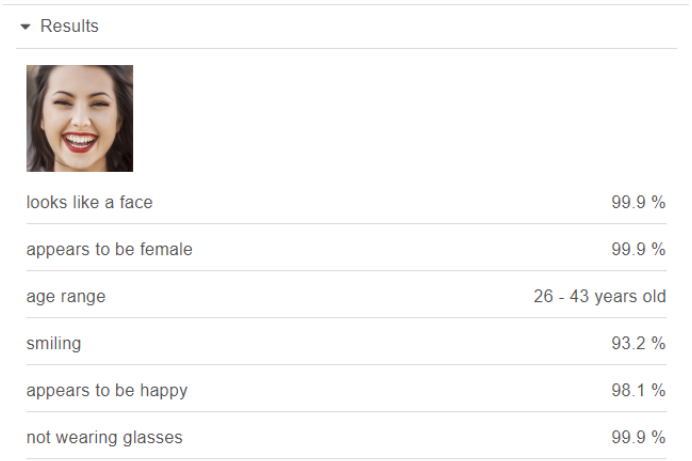


Figure 6.24: Results for first image provided for Facial analysis.

5. The second image is a different face, which is also not smiling. You can enter this URL to provide it to Rekognition: https://images.unsplash.com/photo-1488426862026-3ee34a7d66df?w=800.



Figure 6.25: Second image provided for Facial analysis.

6. Rekognition once again does a good job of identifying the face:



Figure 6.26: Bounding box for the second image provided for Facial analysis

7. The results also reflect that she is not smiling:

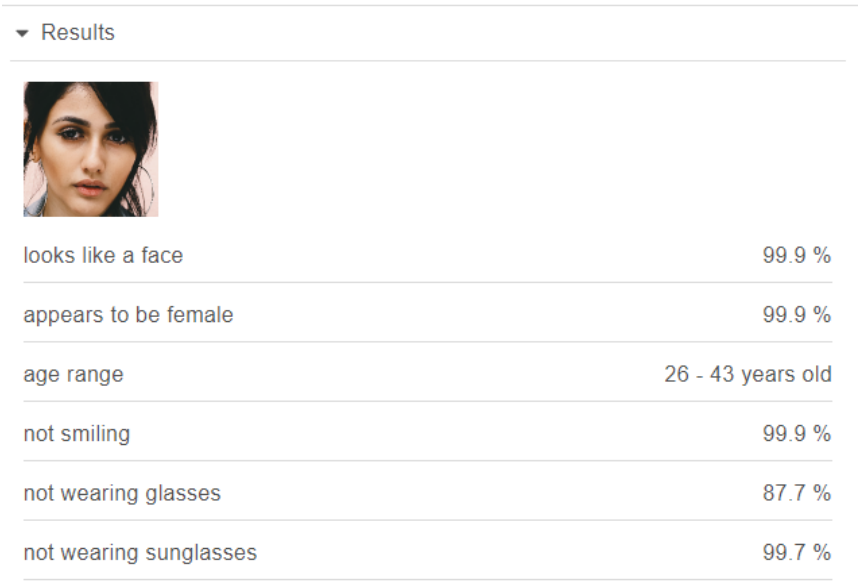


Figure 6.27: Results for the second image provided for Facial analysis.

8. Finally, we will give Rekognition an image with a male to see how it does via the following link: https://images.unsplash.com/photo-1472099645785-5658abf4ff4e?w=1000.



Figure 6.28: Third image provided for Facial analysis.

9. As expected, the service can recognize both males and females. It identifies the face with the bounding box:



Figure 6.29: Bounding box for the third image provided for Facial analysis.

10. And it also displays results with a high degree of confidence (99.9%) that the face is male. It also updates the age range accordingly:

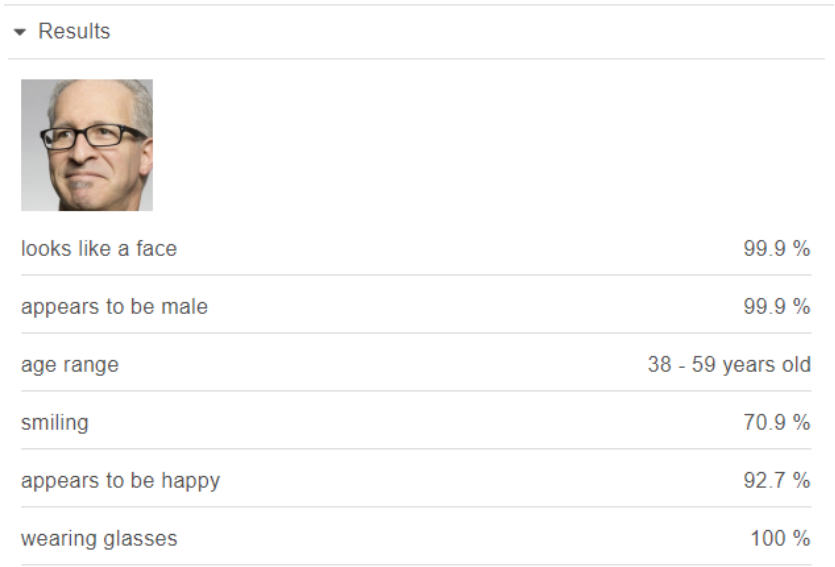


Figure 6.30: Results for third image provided for Facial analysis.

11. It is 70.9% confident that the man is smiling, but we can see from the image that he is not smiling very much, if at all.
12. Finally, the service is 100% sure that he is wearing glasses and 99% sure that he is not wearing sunglasses, which is a highly accurate description of the image.

Celebrity Recognition

Rekognition provides the ability to recognize and label celebrities and other famous people in images. This includes people who are prominent in their field across categories such as politics, sports, business, entertainment, and the media.

It is important to remember that Rekognition can only recognize faces that it has been trained on, and so does not cover a full, exhaustive list of celebrities. However, since Amazon continues to train the system, it is constantly adding new faces to the service.

> **Note**
>
> Click on the Celebrity recognition link in the left toolbar to navigate to the Celebrity recognition page.

To create a celebrity recognition of a sample image you can do the following:

1. Choose the **Text box** under the **Use your own image** panel.
2. Enter the following URL of an image to analyze: https://images.pexels.com/photos/53487/james-stewart-man-person-actor-53487.jpeg.

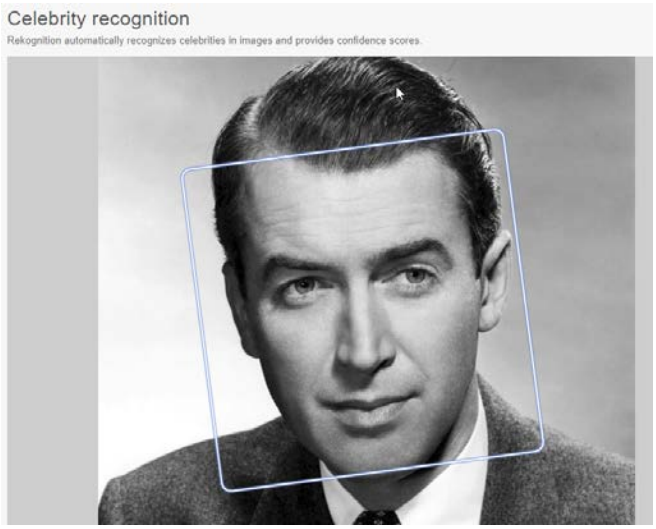


Figure 6.31: First sample image for Celebrity recognition, with results.

3. Clicking on the **Learn More** link takes you to the IMDB page for James Stewart:



Figure 6.32: Learn More link for first sample image.

Exercise 24: Recognizing Celebrities in your Own Images

In this exercise, we will use another site which has a larger collection of celebrity images. You can also use these for free without restrictions. Now, you can try out this service on your own images. We have selected three images that we will try out with this feature.

You may view the license for images from pexels.com here: https://www.pexels.com/creative-commons-images/

> **Note**
>
> Click on the Celebrity recognition link in the left toolbar to navigate to the Celebrity recognition page.

4. Copy and paste or type this `URL` into the **Use image URL** text box under the **Use your own image** section at the bottom of the page, and press the **Go** button to receive results from the service:



Figure 6.33: The Use image URL input text box

5. The first URL to enter into the text box is https://images.pexels.com/photos/276046/pexels-photo-276046.jpeg.



Figure 6.34: First image provided for Celebrity recognition.

6. This is an image of the well-known actress **Charlize Theron**. The Results section will display her name and a **Learn More link**, which will take you to the **IMDB** page about her:

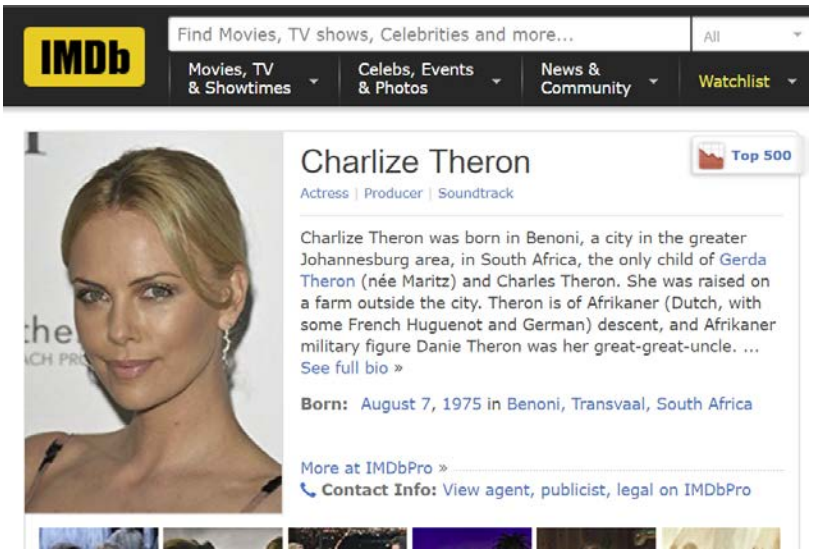


Figure 6.35: Learn More link for the first image provided for Celebrity recognition

7. The second image you can enter into the text box is https://images.pexels.com/photos/2281/man-person-suit-united-states-of-america.jpg?w=800 and this gives us the following image:



Figure 6.36: Second image provided for Celebrity recognition

8. This image displays former US President Barack **Obama**. Rekognition is able to easily detect him as well and displays his name in the Results section. The **Learn More link** once again links to his **IMDB** page:



Figure 6.37: Learn More link for second provided image for Celebrity recognition.

9. The final image contains multiple famous people. You can enter the following URL into Rekognition: https://images.pexels.com/photos/70550/pope-benedict-xvi-president-george-bush-laura-bush-andrews-afb-70550.jpeg?w=800, which gives us the following image:



Figure 6.38: Third image provided for Celebrity recognition, with three bounding boxes

10. You can see in the Results section that Rekognition recognizes all three famous people in the image:

- George W Bush
- Pope Benedict XVI
- Laura Bush

The following image show the result of Celebrity recognition :

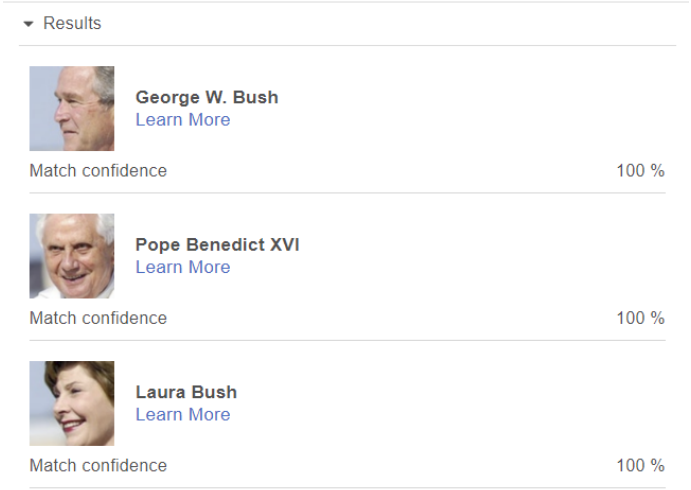


Figure 6.39: Results of the third image provided for celebrity recognition.

11. The Learn More links under their names go to their respective IMDB pages. As we have done previously, you can verify this by clicking on them.

Face Comparison

Rekognition allows you to compare faces in two images. This is mainly for the purpose of identifying which faces are the same in both images. As an example use case, this can also be used for comparing images with people against their personnel photo.

This section demonstrates industry standards so that you can utilize Amazon Rekognition to analyze faces inside an arrangement of pictures with different faces in them. When you indicate a Reference face (source) and a Comparison face (target) picture, Rekognition thinks about the biggest face in the source picture (that is, the reference confront) with up to 100 countenances recognized in the objective picture (that is, the examination appearances), and after that, discovers how intently the face in the source coordinates the appearances in the objective picture. The closeness score for every examination is shown in the Results sheet.

Some restrictions on usage the of this feature are as follows:

- If the source image contains multiple faces, the largest one is used to compare against the target image.
- The target image can contain up to 15 faces. The detected face in the source image is compared against each of the faces detected in the target image.

> **Note**
>
> Click on the Face comparison link in the left toolbar to navigate to the Face comparison page.

With the face comparison feature, there are two sections with images, side by side. You can choose to compare images in the left section with images in the right section. To create a Facial Analysis of a sample image, you can do the following:

1. Choose **Text box** under the **Use your own image** panel on the Left-hand side.
2. Enter the following URL of an image to analyze: https://images.unsplash.com/photo-1524788038240-5fa05b5ee256?w=800.
3. Choose the **Text box** under the **Use your own image** panel on the right side.
4. Enter the following URL of an image to analyze: https://images.unsplash.com/photo-1524290266577-e90173d9072a?w=800:

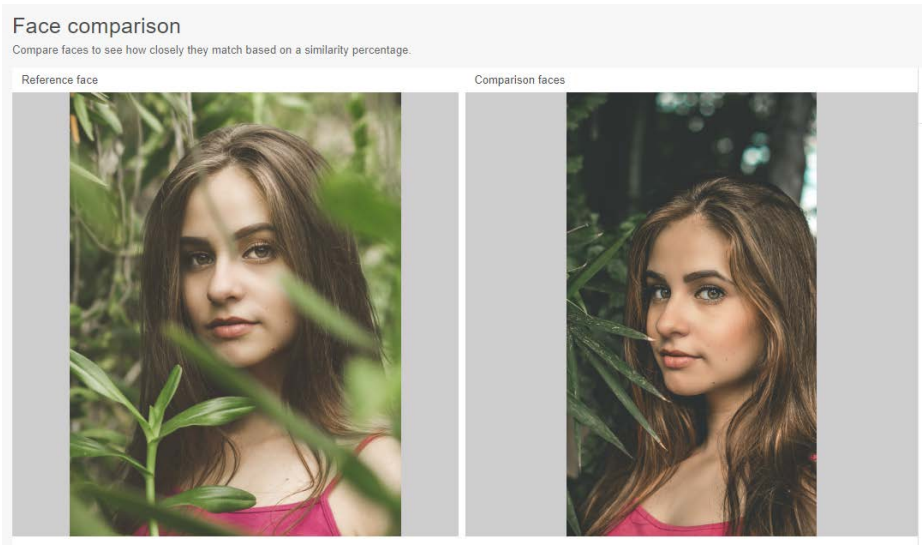


Figure 6.40: First sample images provided for the Face comparison.

5. With the default selections, you can see in the Results section that it identifies the girl in the left image on the right side with a **97%** degree of confidence:



Figure 6.41: Results for first sample images provided for the Face comparison.

Activity 1: Creating and Analyzing Different Faces in Rekognition

In this activity, you can try out Rekognition with your own images. For example, we have provided links to two sets of images that display the same people. You can enter the sets of images into the left (comparison) and right (comparison) sides by using the **Use image URL** text boxes. Remember, there are two this time, so there are two **Go** buttons to press as well. The links for these images are https://images.unsplash.com/photo-1484691065994-f1b2f364d7a6?w=800 and https://images.unsplash.com/photo-1485463598028-44d6c47bf23f?w=800. To ensure that you do this correctly, you will need to navigate to Amazon Rekognition services:

1. Upload the first set of images to Rekognition so that it can recognize the faces.
2. Compare the first set of images with the following parameters:
   - Degree of confidence
   - Comparing with different angles
   - Lighting
   - Position of glasses on the face
3. Upload the second set of mages to Rekognition.
4. Compare the second set of images for face similarity.

Text in Images

Text in image is a feature of Amazon Rekognition that allows you to detect and recognize text such as **street names**, **captions**, **product names**, and **vehicular license plates** in an image. This feature is specifically built to work with real-world images rather than document images.

For each image provided, the service returns a text label and bounding box, along with a confidence score. This can be extremely useful for searching text across a collection of images. Each image can be tagged with the corresponding text metadata based on the results from this and other capabilities of the service.

For now, text supported is Latin scripts and numbers (Western script). Up to 50 sequences of characters can be recognized per image. The text must be horizontal with **+/-** 90 degrees rotation.

> **Note**
>
> Click on the Text in image link in the left toolbar to navigate to the Text in image page.

To identify a Text in Image of a sample image, you can do the following:

1. Choose **Text box** under the **Use your own image** panel on the Left side.
2. Enter the following URL of an image to analyze: https://images.unsplash.com/photo-1527174744973-fc9ce02c141d?w=800 and you will see the following image:



Figure 6.42: First sample image provided for the Text in images.

3. Rekognition surrounds detected text with borders so you can identify which text regions it has recognized. You can see the results of text extraction in the Results section:

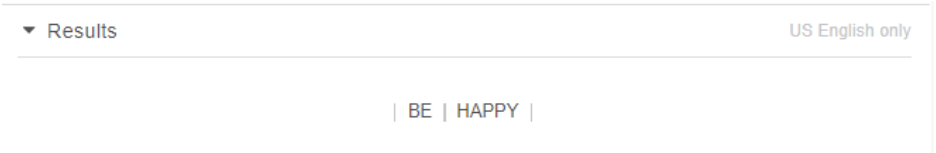


Figure 6.43: Results for first sample image provided for the Text in images.

4. You can see that Rekognition has extracted text from the image with separators | between words in separate regions.

Exercise 25: Extracting Text from your Own Images

In this exercise, you will extract text from your own images. We have provided three royalty-free images for you to use:

> **Note**
>
> Click on the Text in image link in the left toolbar to navigate to the Text in image page.

1. Copy and paste or type the URL into the **Use image URL** text box under the **Use your own image** section at the bottom of the page: https://images.unsplash.com/photo-1521431974017-2521d1cdcd65?w=800 and you will see the following image:



Figure 6.44: First image provided for Text in images

2. You can see the bounding boxes around the image, which signifies that Rekognition has recognized the text in the image. The results can be viewed in the Results panel to the right of the image:

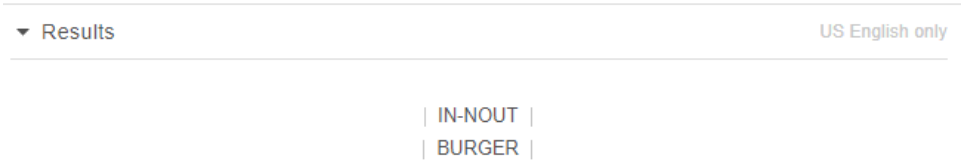


Figure 6.45: Results of the first image provided for Text in images

3. The next image is at the following URL. Copy and paste or type it into the **Use image URL** text box as before: https://images.unsplash.com/photo-1528481958642-cd4b4efb1ae1?w=800 and you will see the following image:



Figure 6.46: Second image provided for Text in images.

4. You can see that Rekognition has recognized the main text in the window of the shop: **OCEAN GIFTS SPORTFISHING WHALE WATCHING**.

5. However, it has also placed a rectangle around the fuzzy text on the left-hand side of the image. When you view the results, you will see the extra text in the text output:



Figure 6.47: Results of the second image provided for Text in images

6. It has recognized **ac** from the blurry text, and nothing more. This is something you should be aware of and watch out for in your results. It is possible for Rekognition to get confused and return spurious results.

7. Finally, copy and paste or type the following URL into the **Use image URL** text box: https://images.unsplash.com/photo-1456668609474-b579f17cf693?w=800 and you will see the following image:



Figure 6.48: Third image provided for the Text in images.

8. This is another example of a license plate. However, if you look closely, you might see that the service has placed a bounding box not just around the license plate but also around other text on the upper grille of the car, which it believes it is able to identify. The results are as follows:



Figure 6.49: Results of the third image provided for Text in images.

9. It has identified the text **e** from the region that it outlined in addition to the license plate text. Once again, this is something to be aware of when relying on the results, and a manual validation process is clearly beneficial in this case.

> **Note**
>
> To refer to the detailed steps, go to the *Appendix A* at the end of this book on Page no. 232

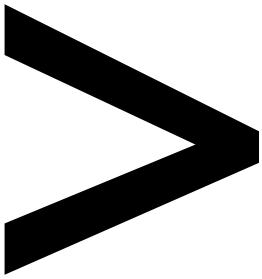
Summary

In this chapter, you learned how to use the various features of the Amazon Rekognition service and applied this to images. First, you used the service to recognize objects and scenes in images. Next, you moderated images that might have objectionable content by using Rekognition to recognize the objectionable content in the images.

You were able to analyze faces with Rekognition and were also able to identify their gender, age range, whether they were smiling, and whether they were wearing glasses.

You also recognized celebrities and famous people with the service, and compared faces in different images to see whether they were the same. Finally, you were able to extract text that was displayed in images.

These types of features would have seemed unbelievable just a few years ago. The nature of machine learning and artificial intelligence is that immense strides have been made recently, today, and into the foreseeable future in terms of what computers are going to be able to do – and AWS will be able to provide these services to you and your applications!



Appendix A

About

This section is included to assist the students to perform the activities present in the book. It includes detailed steps that are to be performed by the students to complete and achieve the objectives of the book.

Chapter 1: Introduction to Amazon Web Services

Activity 1: Importing and exporting the data into S3 with the CLI.

1. Verify the configuration is correct by executing, "**`aws s3 ls`**" to output your bucket name (bucket name will be unique):

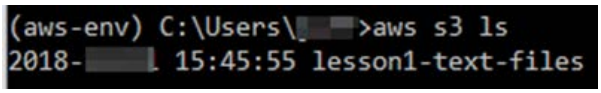


Figure 1.34: Command line

2. Execute "aws s3 ls <bucket-name>" to output the text file **`pos_sentiment__leaves_of_grass.txt`**" in the bucket.

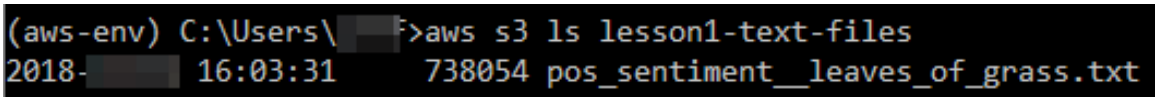


Figure 1.35: Command line

3. Create a new S3 bucket with the following command (to note: your bucket name needs to be unique. Refer to the S3 "Rules for Bucket Naming" for specific details):

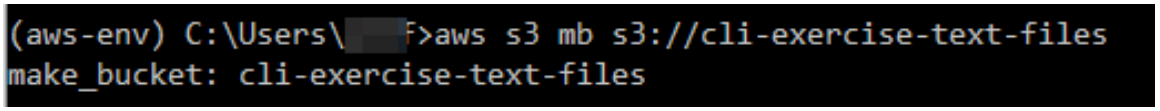


Figure 1.36: Command line

4. In the command prompt, navigate to the "**`neg_sentiment__dracula.txt`**" location Execute, "**`aws s3 cp neg_sentiment__dracula.txt`**" to import the text file to your S3 bucket.

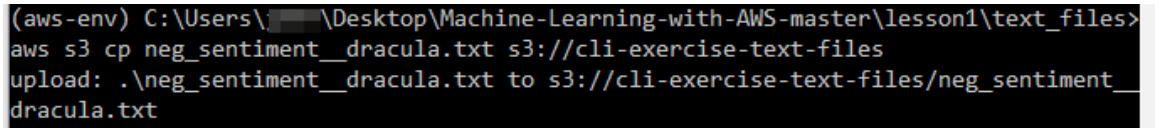


Figure 1.37: Command line

5. Navigate to the "**peter_pan.txt**" file location with your command line. Import the file "**peter_pan.txt**" to your S3 bucket (named "**aws-test-ml-and-ai-two**" in this example) with the following command:

```
(aws-env) C:\Users\    \Desktop\Machine-Learning-with-AWS-master\lesson1\text_files
aws s3 cp peter_pan.txt s3://cli-exercise-text-files
upload: .\peter_pan.txt to s3://cli-exercise-text-files/peter_pan.txt
```

Figure 1.38: Command line

6. Navigate to your Desktop in the command line. Create a new local folder named "**s3_exported_files**" with the command "**mkdir s3_exported_files**"

```
(aws-env) C:\Users\    \Desktop\s3_exported_files>
```

Figure 1.39: Command line

7. Next, recursively export both files ("**neg_sentiment__dracula.txt**" and "**peter_pan.txt**") from the S3 bucket to your local directory with the "- -recursive" parameter. See below for the command's execution.

```
(aws-env) C:\Users\    \Desktop\s3_exported_files>aws s3 cp s3://cli-exercise-text-files . --recursive
download: s3://cli-exercise-text-files/peter_pan.txt to .\peter_pan.txt
download: s3://cli-exercise-text-files/neg_sentiment__dracula.txt to .\neg_sentiment__dracula.txt
```

Figure 1.40: Command line

8. Verify the objects were exported successfully to your local folder with the, "**dir**" command (see below):

```
(aws-env) C:\Users\    \Desktop\s3_exported_files>dir
 Volume in drive C is Windows8_OS
 Volume Serial Number is [redacted]

 Directory of C:\Users\    \Desktop\s3_exported_files

[redacted]  06:54 PM    <DIR>          .
            06:54 PM    <DIR>          ..
            06:41 PM           842,169 neg_sentiment__dracula.txt
            06:44 PM           262,704 peter_pan.txt
               2 File(s)      1,104,873 bytes
               2 Dir(s)  33,521,958,912 bytes free
```

Figure 1.41: Output

Activity 2: Test Amazon Comprehends API features.

In this section, we will learn about display text analysis output using a partial text file input in the API explorer. We will be exploring API's is a skill to save development time by making sure the output is in a desired format for your project. Thus, we will test Comprehend's text analysis features.

We will consider an example suppose you are an entrepreneur creating a chatbot. You identified a business topic and corresponding text documents with content that will allow the chatbot to make your business successful. Your next step is to identify/verify an AWS service to parse the text document for sentiment, language, key phrases, and entities. Before investing time in writing a complete program, you want to test the AWS service's features via the AWS management console's interface. To ensure that it happen correctly, you will need to have search the web for an article (written in English or Spanish) that contains a subject matter (sports, movies, current events, etc.) that you are interested. And, also AWS Management Console accessible via the root user's account

You are aware exploring API's is a skill to save development time by making sure the output is in a desired format for your project.

1. Click in the **Search** bar of AWS Services to search the service name

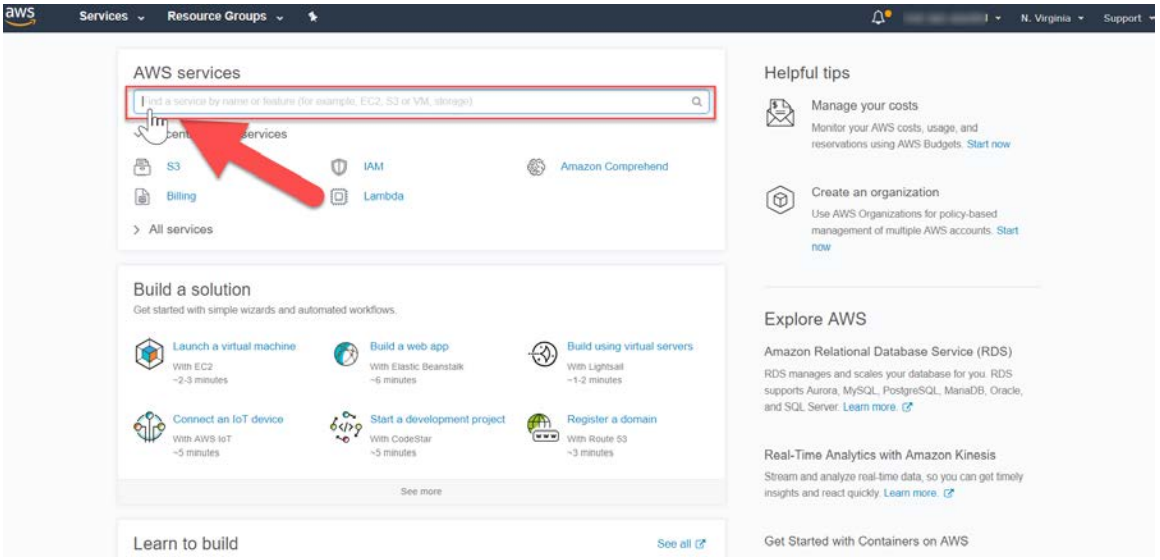


Figure 1.42: Searching of AWS Service

2. Search the **`Amazon Comprehend`** option and select the option you will be directed. Get started screen.

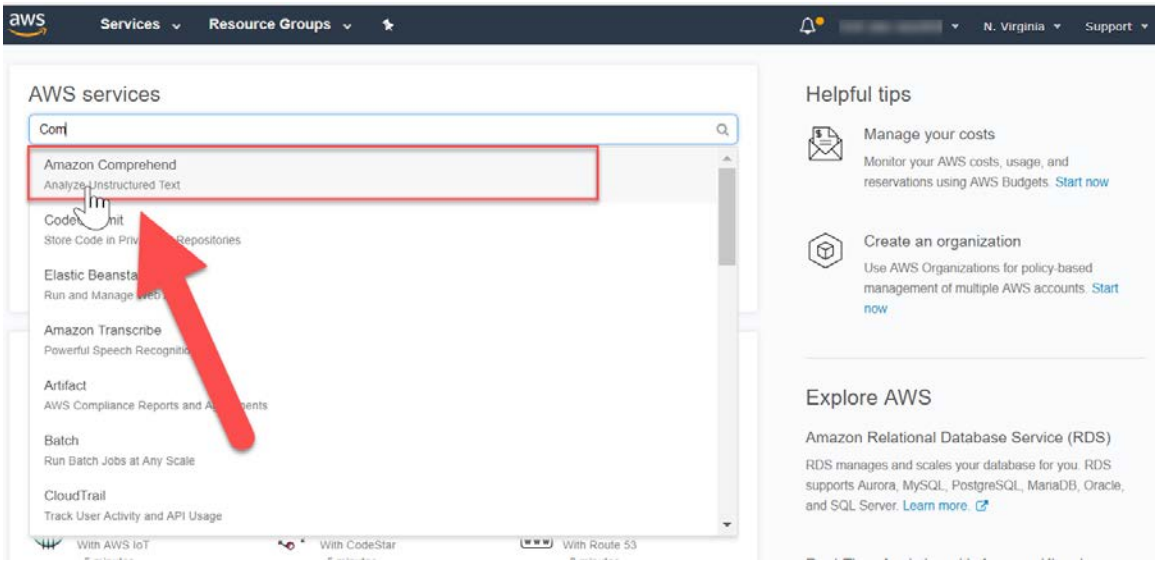


Figure 1.43: Selecting the Service

3. You will be directed to the **`API explorer`**. Navigation to Topic modeling and Documentation. The middle is a GUI to explore the API, and the right side provides real-time output for text input.

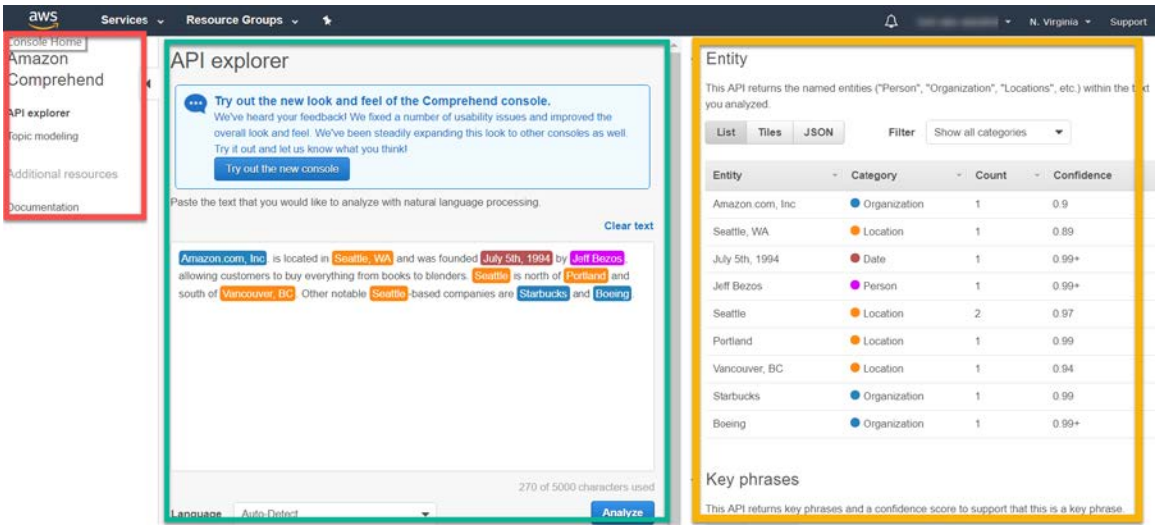


Figure 1.44: API Explorer

4. Click clear text to clear all default services. Navigate to open the following URL in a new tab http://www.gutenberg.org/cache/epub/1322/pg1322.txt

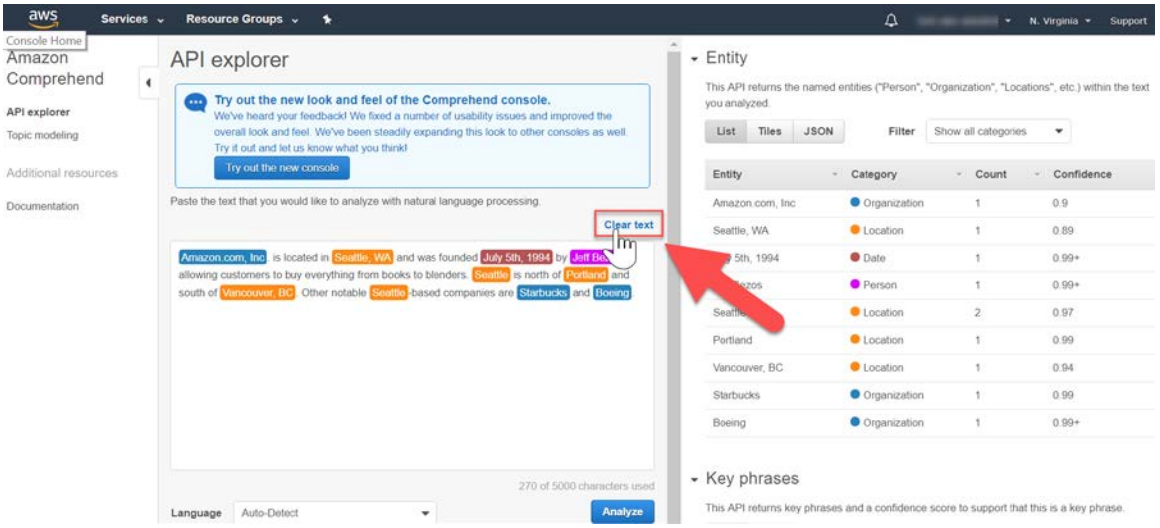


Figure 1.45: API Explorer Screen

5. Copy the first poem, and paste it in the Explorer and click Analyze to see the output

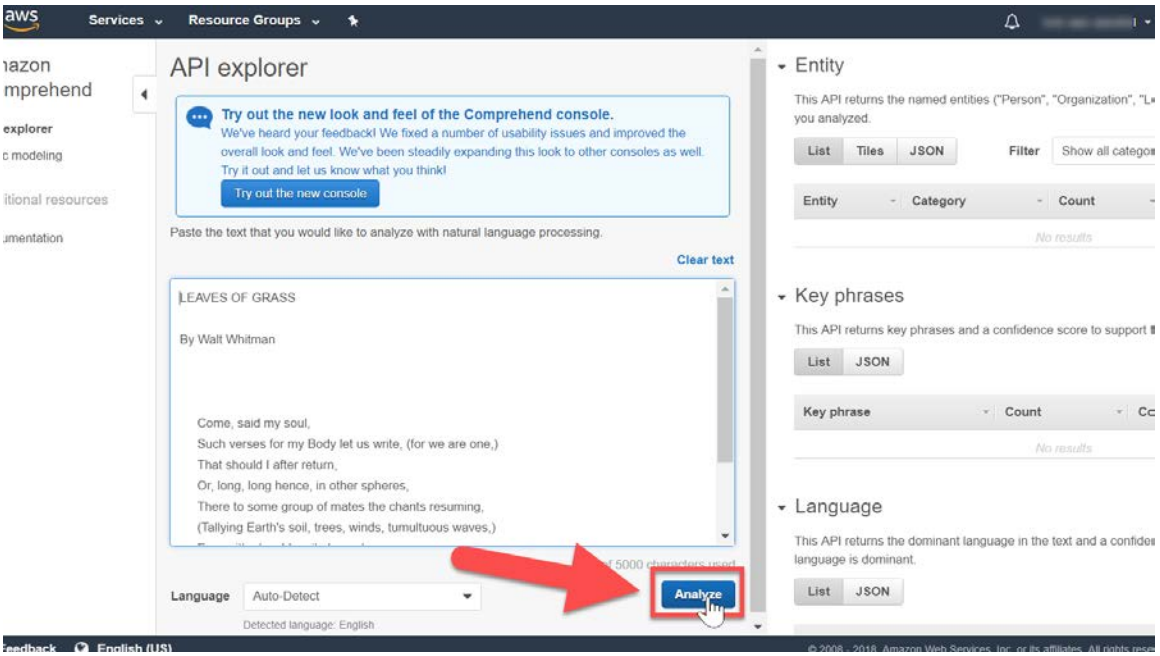


Figure 1.46: Analyzing the Output

6. Review the right side of the screen for Entity, Key phrases, Language, and scroll down to view Sentiment.

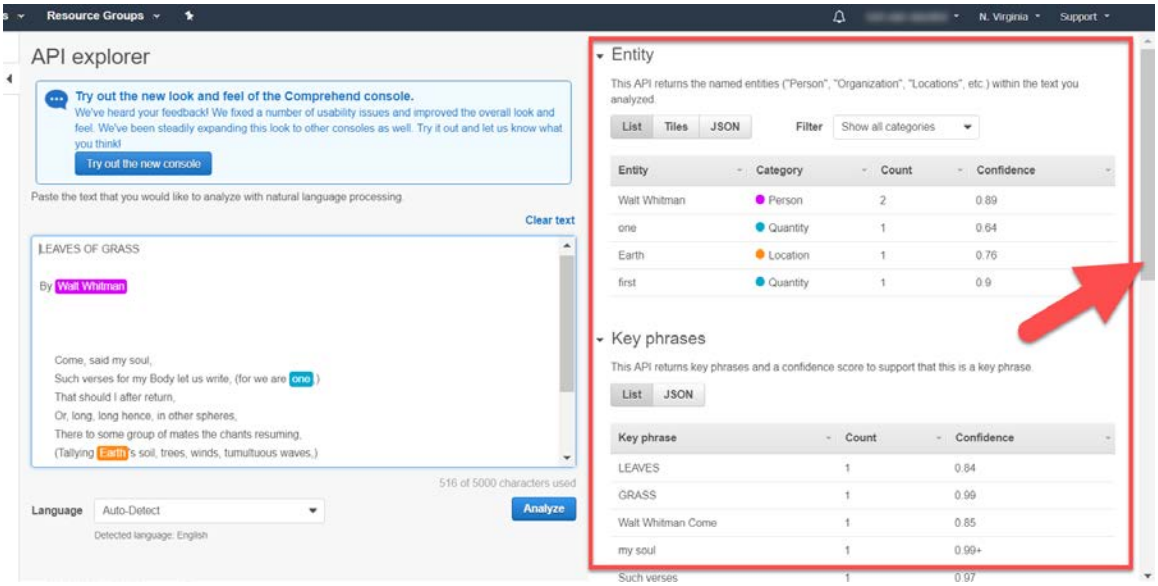


Figure 1.47: Result

You now know how to explore an API, and how to display sentiment output for the same.

Chapter 2: Summarizing Text Documents Using NLP

Activity 3: Integrating Lambda with Amazon Comprehend to perform text analysis

1. Next, we will upload the "**`test_s3trigger_configured.txt`**" file to our S3 bucket to verify the lambda s3_trigger function was configured successfully.
2. Navigate to the s3 page: https://console.aws.amazon.com/s3/
3. Click the bucket name you are using to test the **`s3_trigger`** function (in my case: "**`aws-ml-s3-trigger`**").

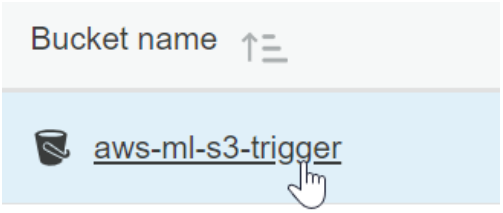


Figure 2.37: S3 bucket list

4. **Click** Upload.

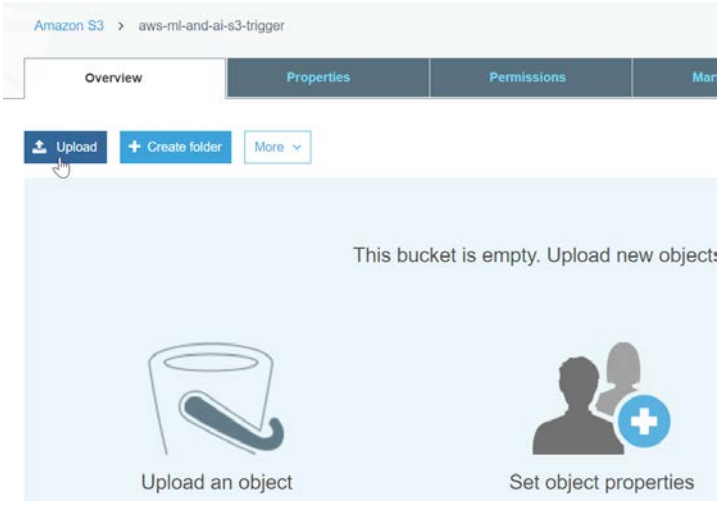


Figure 2.38: S3 bucket list Upload screen

5. The following screen will display.

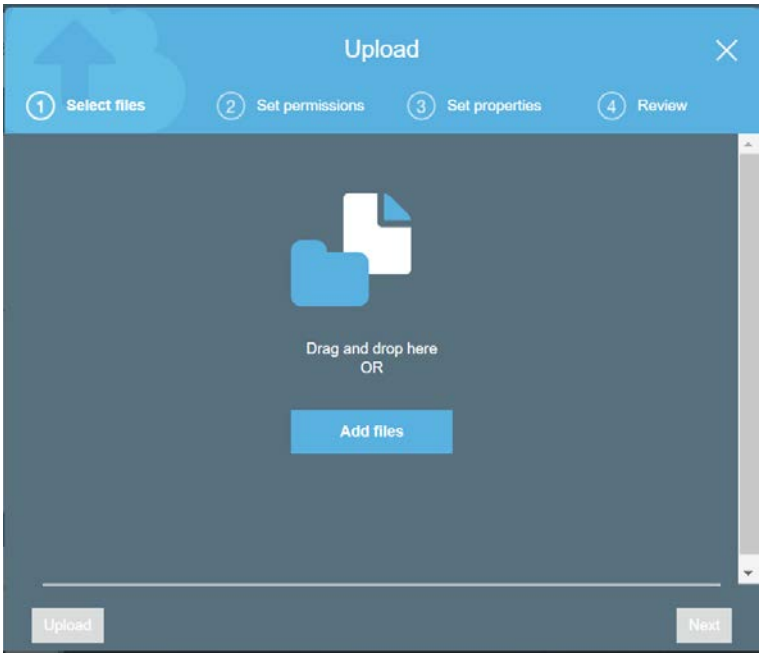


Figure 2.39: S3 Upload bucket "add files" screen.

6. **Click** Add files.

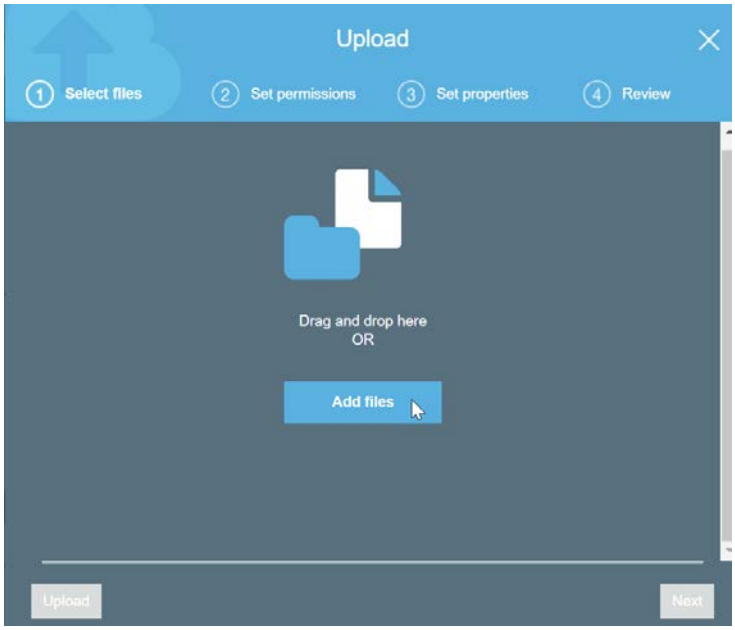


Figure 2.40: S3 Add files selection screen.

7. Navigate to the "**`test_s3trigger_configured.txt`**" file location. Select the file.
8. **Navigate** to the text file's location and open the file. The file contains the following text:
9. "I am a test file to verify the s3 trigger was successfully configured!"
10. Before we execute the s3_trigger, consider the output based on the following aspects of the text: sentiment (positive, negative, or neutral), entities (quantity, person, place, etc.), and key phrases.

> **Note**
>
> The "test_s3trigger_configured.txt" is available at the following GitHub repository: https://github.com/TrainingByPackt/Machine-Learning-with-AWS/blob/master/lesson2/topic_c/test_s3trigger_configured.txt

11. Click **Upload**.

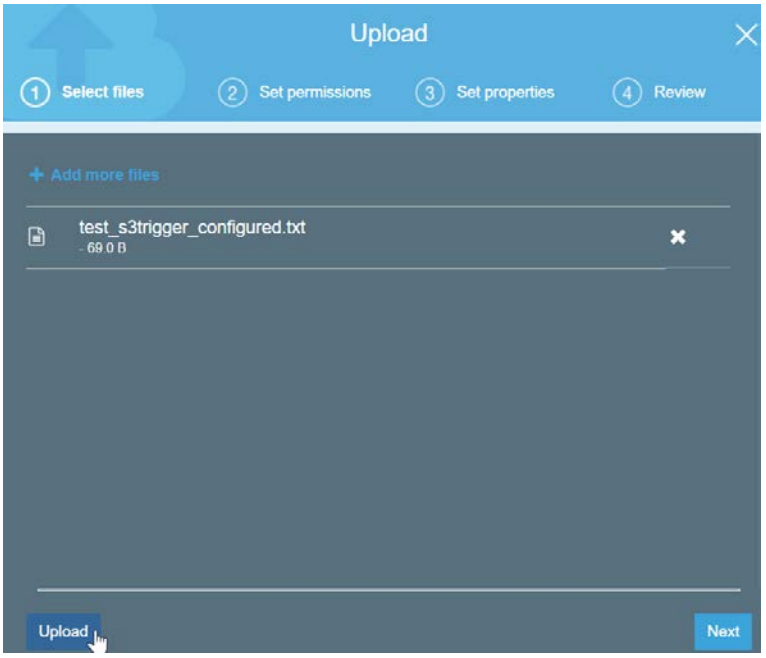


Figure 2.41: S3 file added to bucket for Lambda trigger test

12. Navigate back to the **`s3_trigger`**. **Click** Monitoring

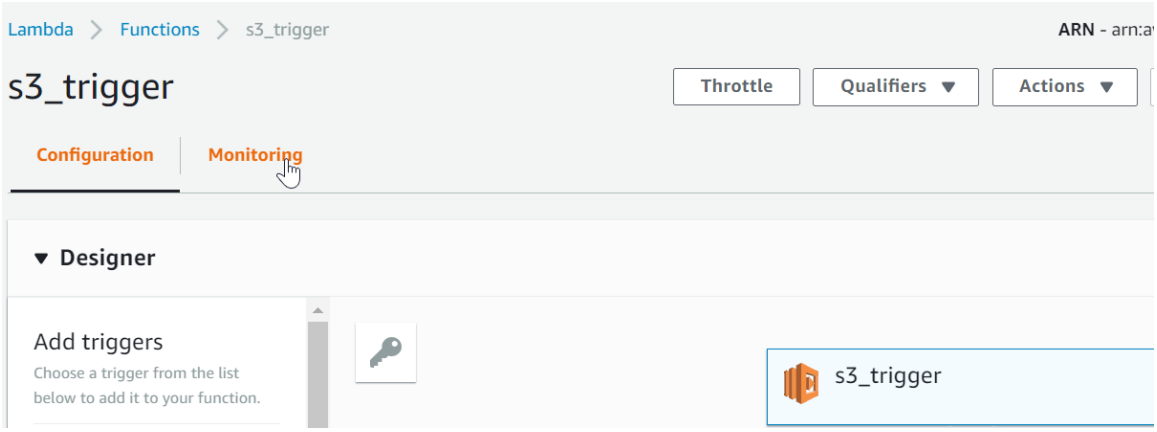


Figure 2.42: Select Monitoring tab

13. Click View logs in CloudWatch.

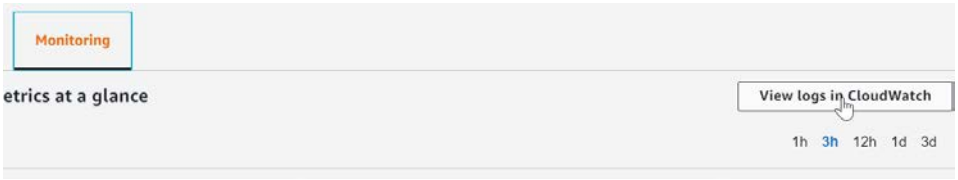


Figure 2.43: Select the View logs in CloudWatch

14. Click on **the Log Stream**.

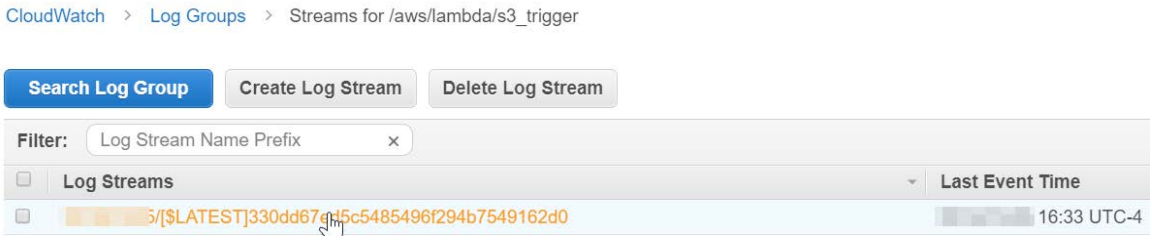


Figure 2.44: Select the Log Stream

15. Select the circle option next to Text to expand the output:



Figure 2.9: Click the circle option to expand the lambda output

16. Below is the first few lines of the output, and to see the entire output you need to Scroll down to view all of the results (see below). We will interpret the total output in the next step.



Figure 2.45: The top portion of the s3_trigger output

17. **Sentiment_response** -> Classified as 60.0% likely to be Positive

18. Sentiment_response:

    {'Sentiment': 'POSITIVE','SentimentScore':{'Positive': 0.6005121469497681,'Negative': 0.029164031147956848, 'Neutral': 0.3588017225265503, 'Mixed': 0.01152205839753151},

    entity_response --> Classified as 70.5% likely the type is Quantity

    entity_response:

    {Entities':[{'Score':0.7053232192993164, 'Type': 'QUANTITY','Text': '3 trigger', 'BeginOffset': 35, 'EndOffset': 44}],

    key_phases_response -> Classified as 89.9% likely "a test file" and 98.5% likely 'the s3 trigger" are the key phrases.

    key_phases_response:

    {'KeyPhrases': [{'Score': 0.8986637592315674, 'Text': 'a test file', 'BeginOffset': 8, 'EndOffset': 19}, {'Score': 0.9852105975151062, 'Text': 'the s3 trigger', 'BeginOffset': 30, 'EndOffset': 44}],

Chapter 3: Perform Topic Modeling and Theme Extraction

Activity 4: Perform Topic modeling on a set of documents with unknown topics

1. Navigate to following link for obtaining the text data file that contain negative review comments https://github.com/TrainingByPackt/Machine-Learning-with-AWS/blob/master/lesson3/activity/localfoldernegative_movie_review_files/cv000_29416.txt
2. Navigate to S3: https://s3.console.aws.amazon.com/s3/home
3. Click the bucket for the "**`input-for-topic-modeling`**".

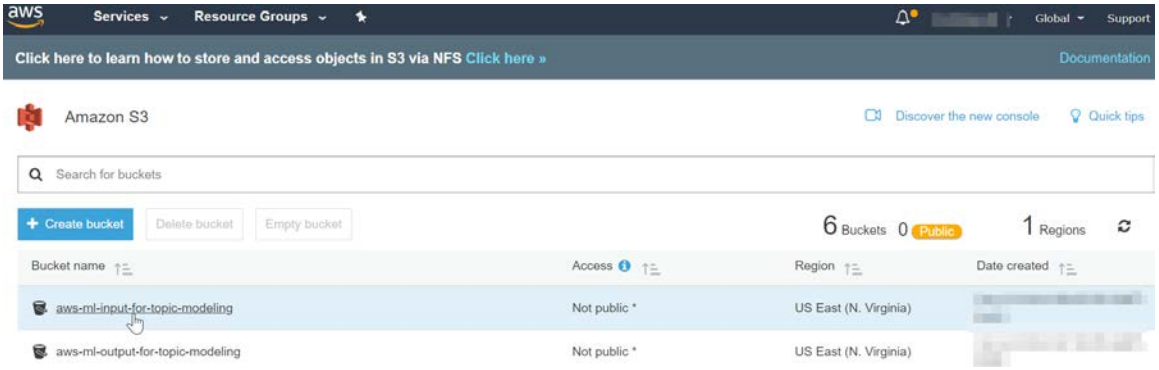


Figure 3.43: S3 home screen for 'input-for-topic-modeling'

4. Click Create folder.

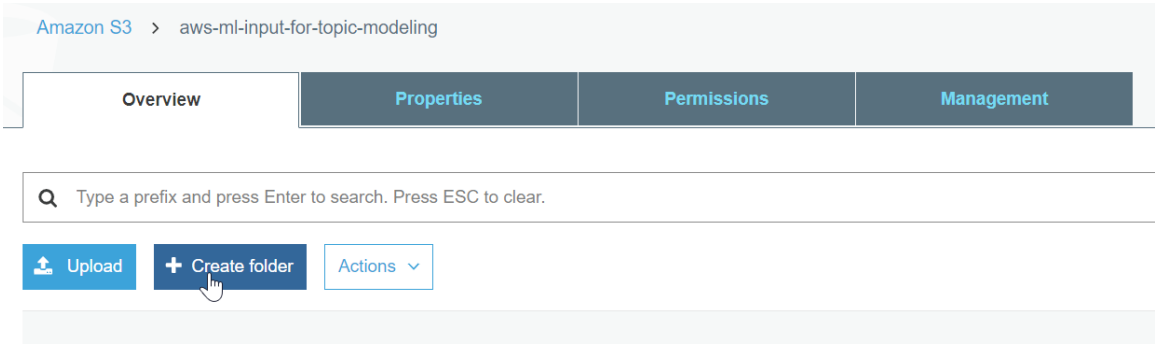


Figure 3.44: Click Create folder

5. Type "**negative_movie_review_files**", and click **Save**.

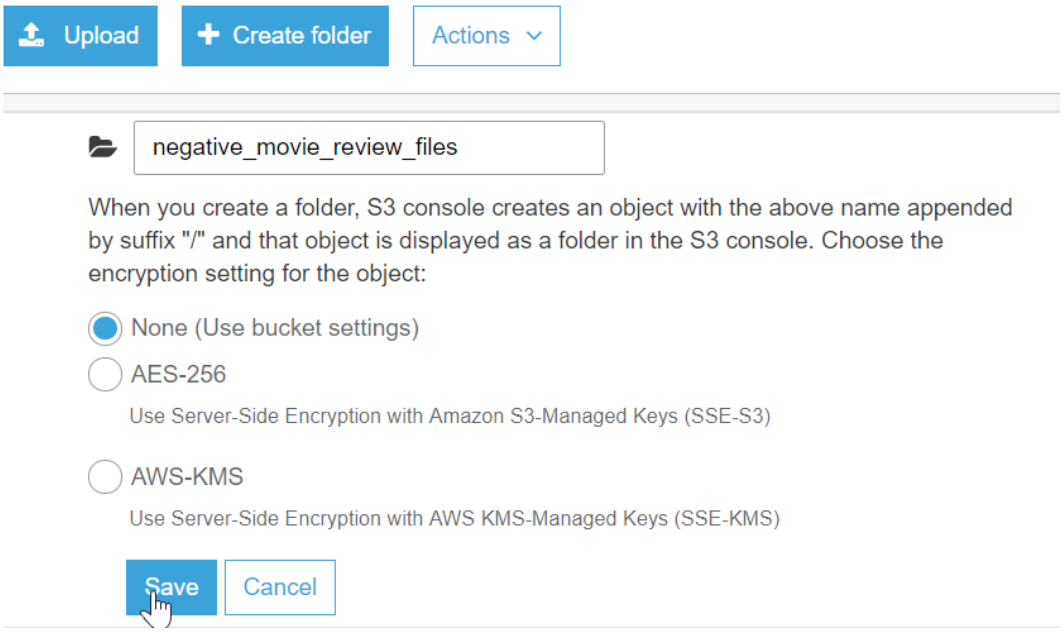


Figure 3.45: Click Save

> **Note:**
>
> For this step, you may either follow along the exercise and type in the code or obtain it from the source code folder **`text_files_to_s3.py`** and paste it into the editor. The source code is available via GitHub in the following repository: https://github.com/TrainingByPackt/Machine-Learning-with-AWS/blob/master/lesson3/activity/text_files_to_s3.py

6. Firstly, you will import the os and boto3 packages using the following comment

```
import os
import boto3
```

7. Next, type in your unique bucket name.

```
BUCKET_NAME = '<insert a unique bucket name>'
BUCKET_FOLDER = 'negative_movie_review_files/'
```

8. Next, get the working directory of the local path to the text files:

```
LOCAL_PATH = os.getcwd() +'\\local_folder__negative_movie_review_files\\'
```

9. Create a list of all text files:

```
text_files_list = [f for f in os.listdir(LOCAL_PATH) if f.endswith('.txt')]
```

10. Iterate on all files, and upload each to s3:

```
for filename in text_files_list:
    s3.upload_file(LOCAL_PATH + filename, BUCKET_NAME, BUCKET_FOLDER +
filename)
```

11. Next, in your command prompt, navigate into to the "**activity__topic_modeling_on_documents**" directory and execute the code with the following: python **text_files_to_s3.py**

12. The result is 1000 text files uploaded to the S3 **negative_movie_review_files** folder. See below for the top S3 output (see below):

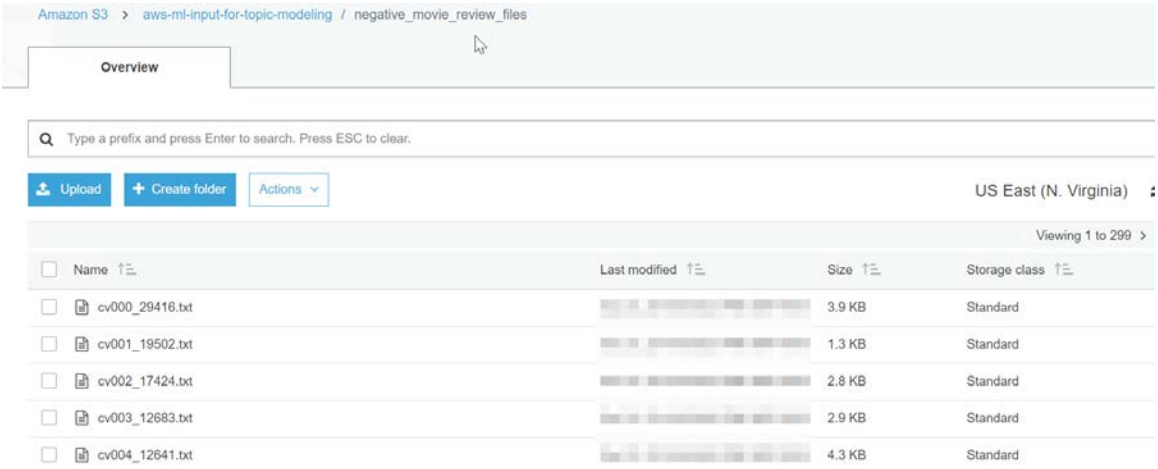


Figure 3.47: negative_movie_review_files in S3

13. Next, navigate to AWS Comprehend. Click the comprehend link: https://console.aws.amazon.com/comprehend/home.

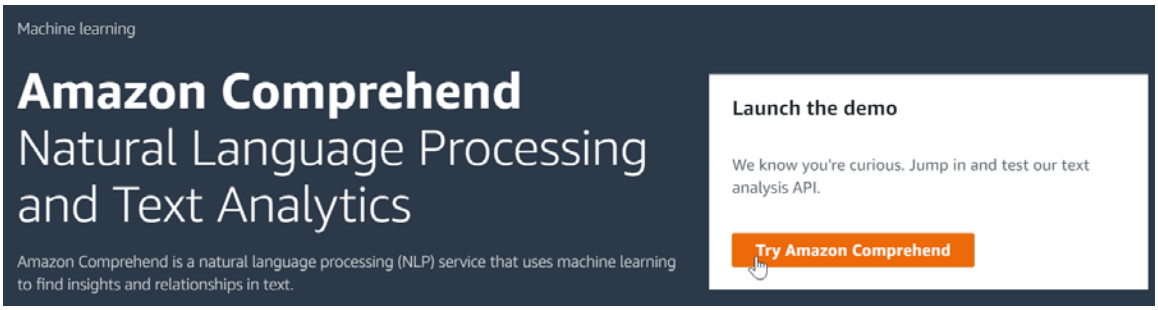


Figure 3.48: Amazon Comprehend home screen

14. Now, click on the **Organization**.

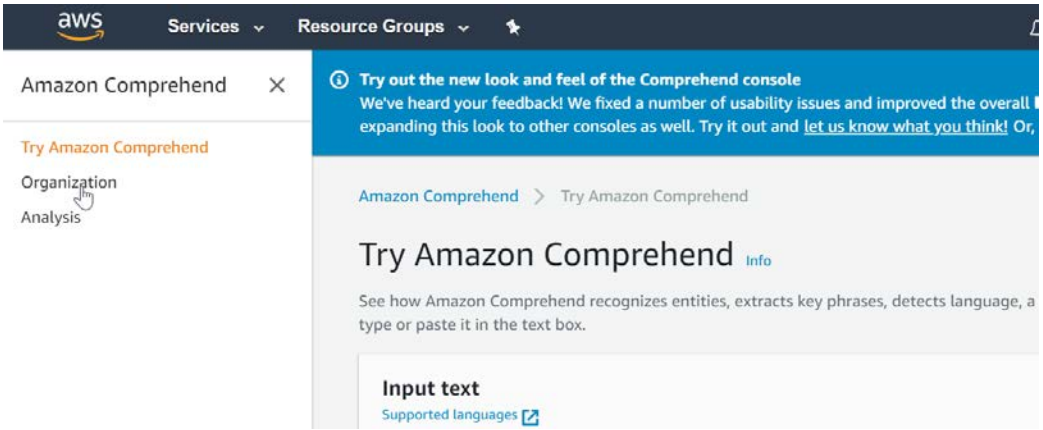


Figure 3.49: Select Organization

15. Now, click on the Create job.

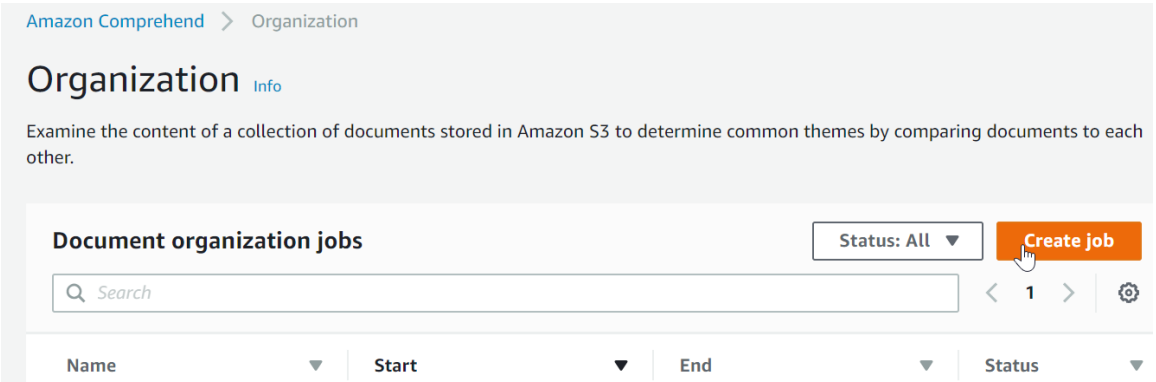


Figure 3.50: Click Create job

16. Now, type "**`unknown_topic_structure_job`**" in the Name input field.



Figure 3.51: Enter 'unknown_topic_strucutre_job'

17. Now, scroll down to the Choose input data section, and click Search.

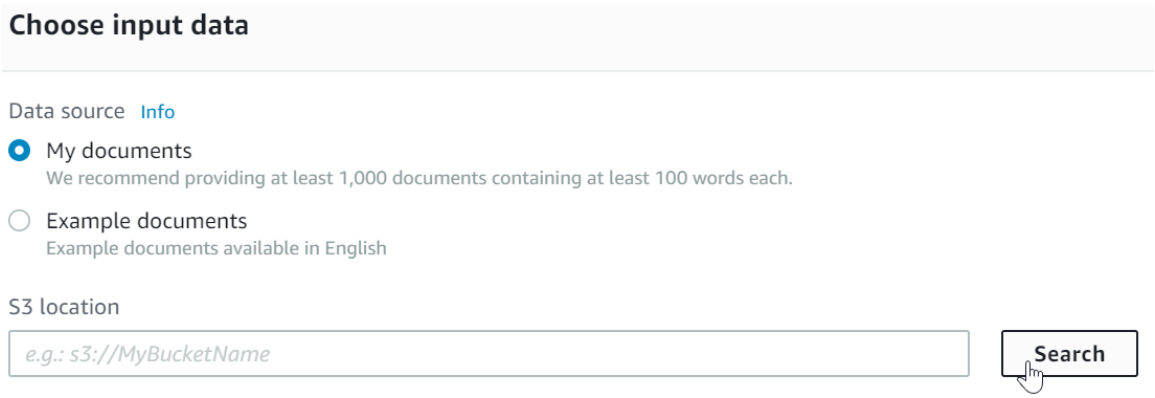


Figure 3.52: Select Search button

18. Click the arrow next to bucket you selected to input files for topic modeling ("**`aws-ml-input-for-topic-modeling`**").

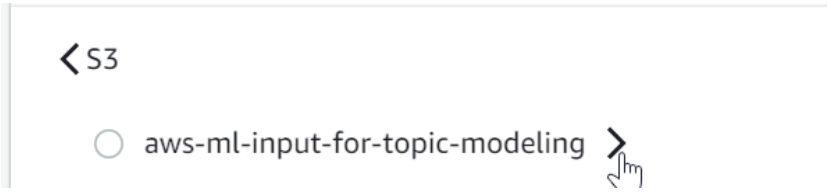


Figure 3.53: Expand S3 bucket sub-folders

19. Click the circle next to the "**`negative_movie_review_files`**" folder.

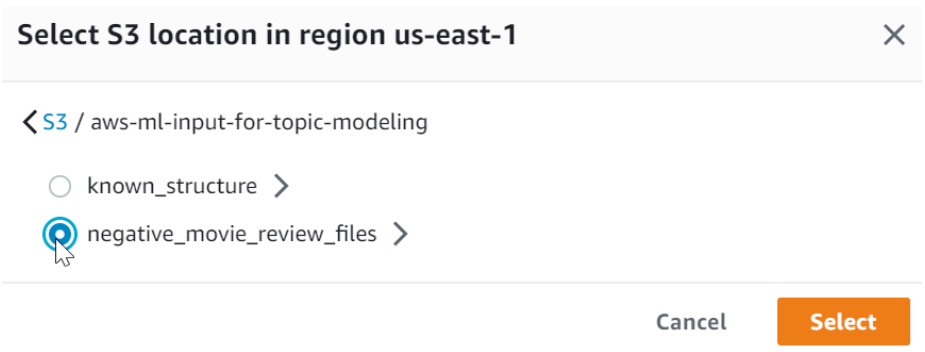


Figure 3.54: Select the negative_movie_review_files folder

20. Now, click Select to choose the file.

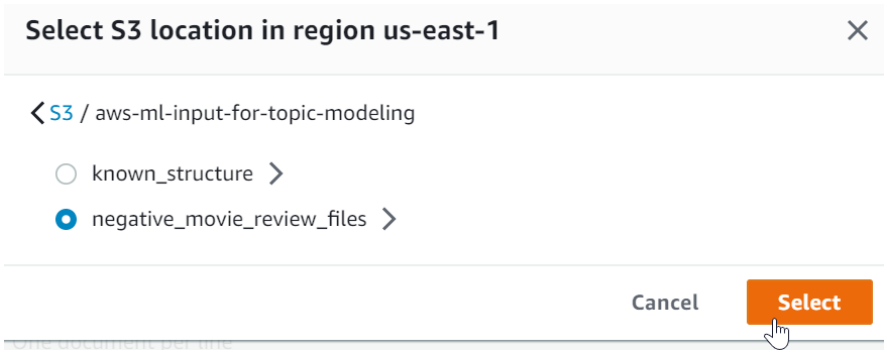


Figure 3.55: Click the Select button

21. You will be redirected to the Amazon Comprehend main page. Select "**One document per file**" selection from the Input format drop down.

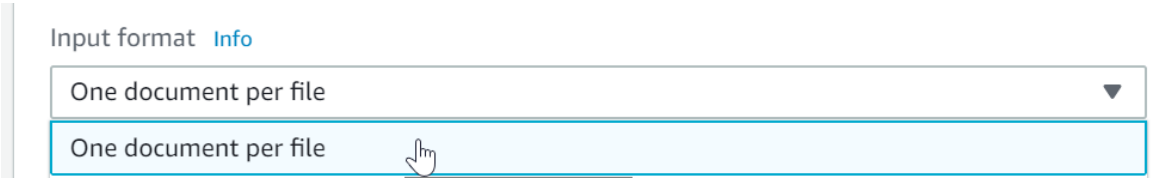


Figure 3.56: Select One document per file option

22. Next, enter 40 in the Number of topics input field.



Figure 3.57: Enter 40 topics

23. Scroll down to the Choose output location, and click Search

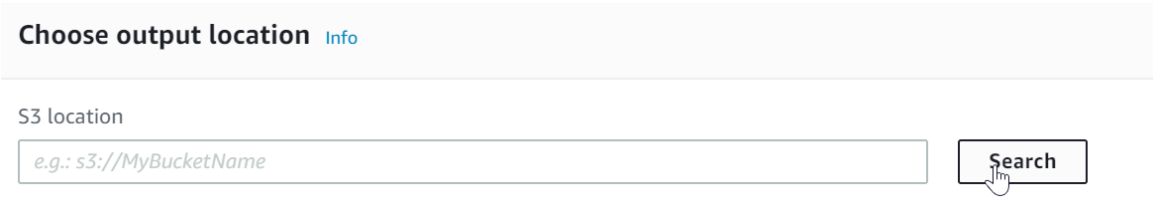


Figure 3.58: Click Search

24. Select the output bucket you uniquely named for the topic modeling output.

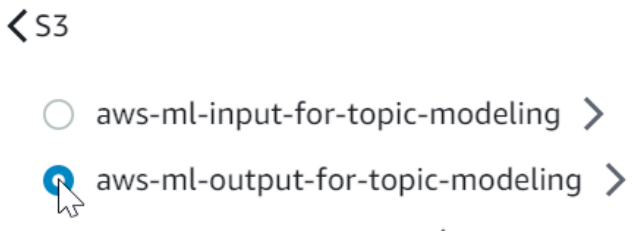


Figure 3.59: Select the S3 bucket for the topic modeling outtput

25. Click on the Select button.



Figure 3.60: Confirm by clicking the Select button

26. Click the dropdown and select the "**`AmazonComprehendServiceRole-myTopicModeing-Role`**" IAM role.

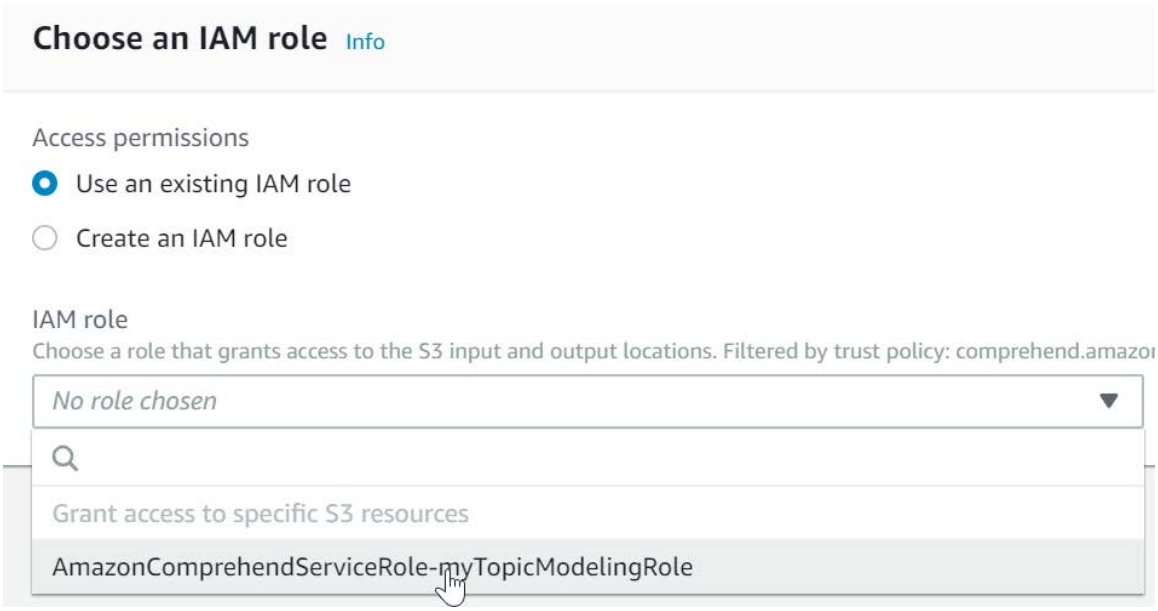


Figure 3.61: Select the existing IAM role

27. Click on **Create** job button.

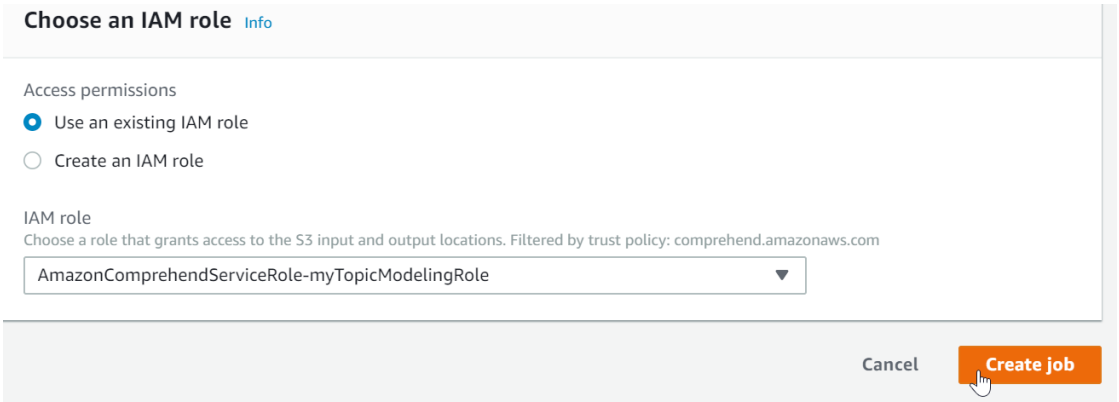


Figure 3.62: Click Create job

28. The topic modeling job status will first display "**Submitted**."

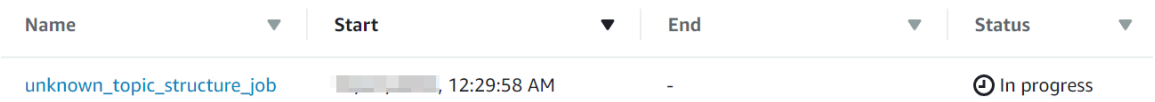


Figure 3.63: Status 'Submitted"

29. The topic modeling job status will next display "**In progress**". The topic modeling job duration is about 6 minutes.

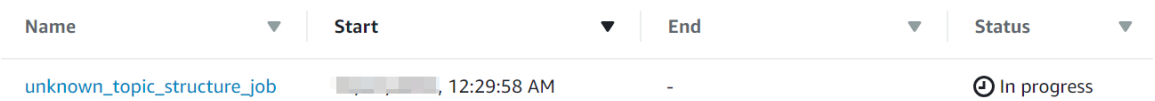


Figure 3.64: "In progress" status

30. When the status changes to "Completed." Click the "**unknown_topic_structure_job**" link.

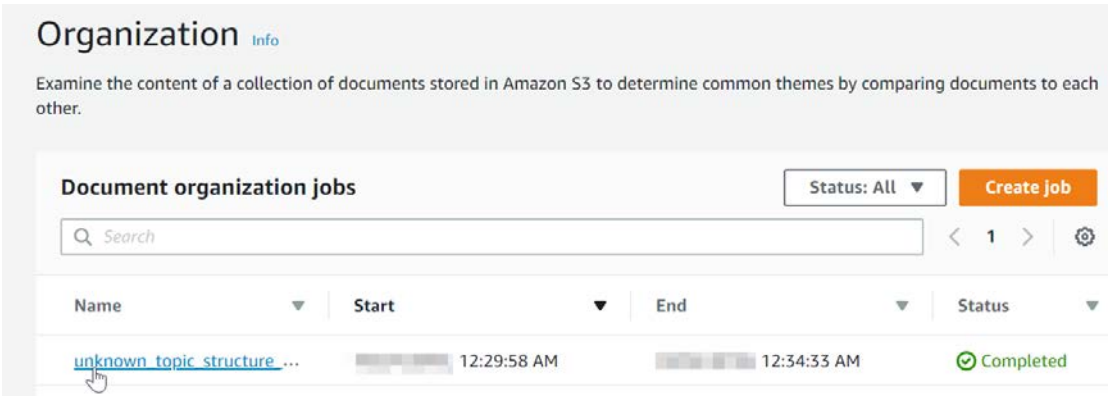


Figure 3.65: Select the hyperlinked topic modeling link

31. Scroll down and click the topic modeling output hyperlink (*yours will display a different unique topic modeling job alphanumeric character string).

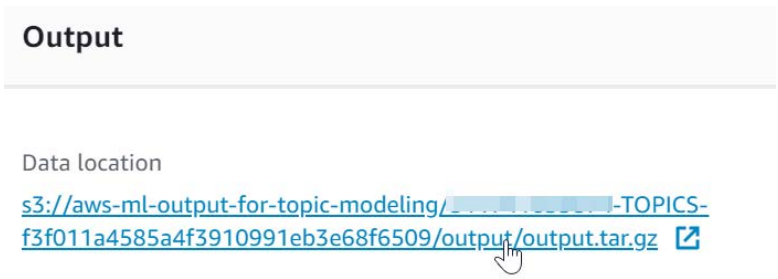


Figure 3.66: Click the topic modeling output S3 location

32. You will be directed to the S3 output folder for the topic modeling job. Click the hyperlinked folder.

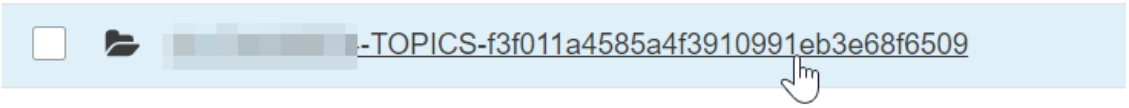


Figure 3.67: Hyperlinked folder location

33. Click the output folder.

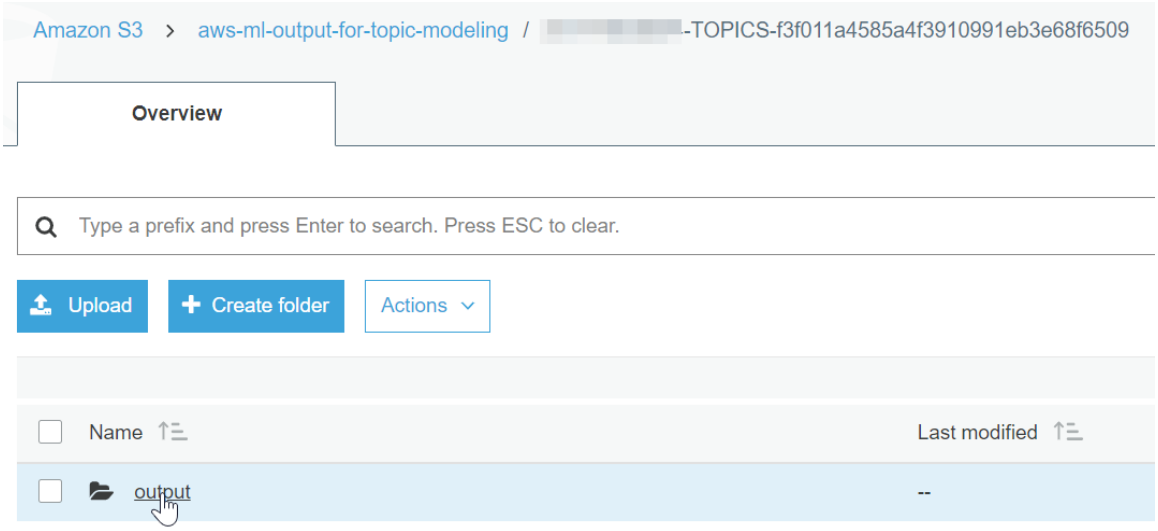


Figure 3.68: Click output

34. Click the **`output.tar.gz`** file.

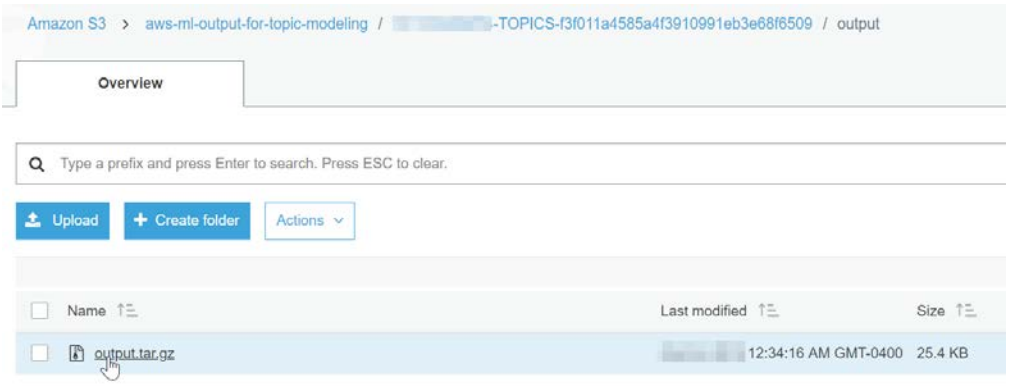


Figure 3.69: Click output.tar.gz file

35. Click on Download as.

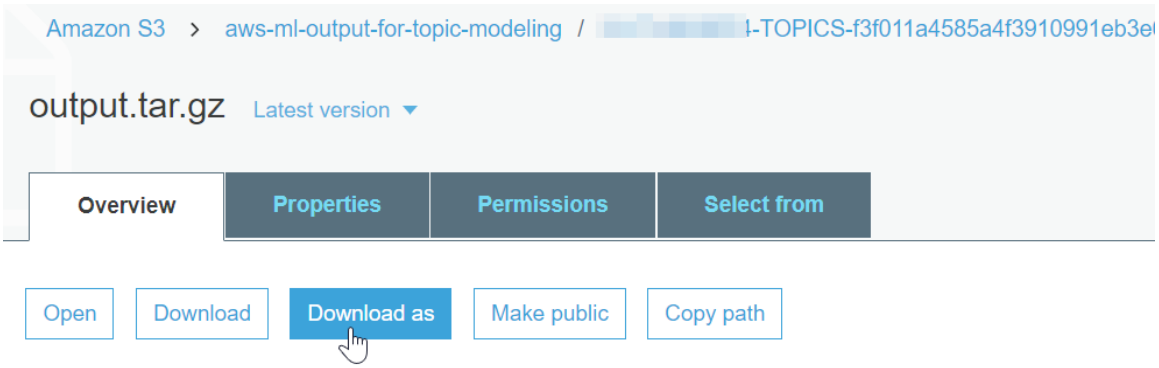


Figure 3.70: Click Download as

36. Right-click the **`output.tar.gz`** file and click **Save** link as...

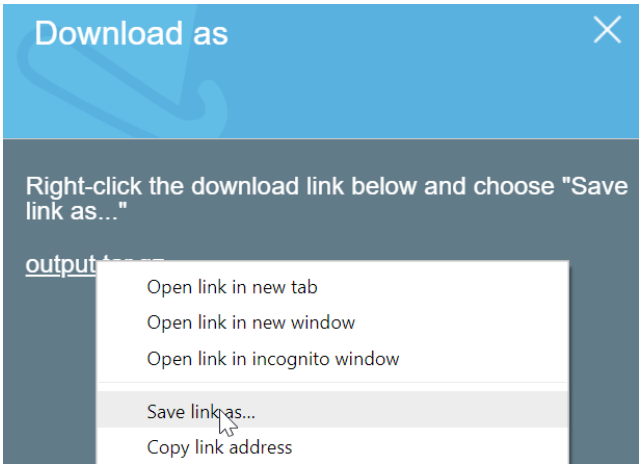


Figure 3.71: Select Save link as...

37. Select the Desktop and click Save.

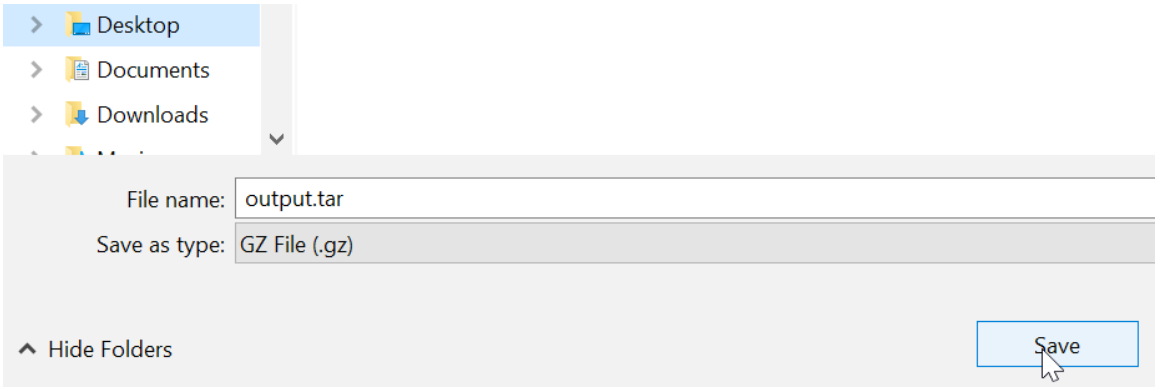


Figure 3.72: Click Save

38. Navigate to the Desktop. Right-click the output.tar.gz file and select **Extract Here**. Right-click the output.tar file and select Extract Here

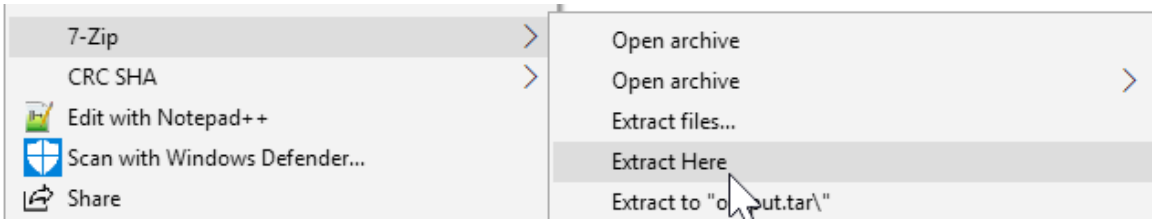


Figure 3.73: Select Extract Here

39. The result is two CSV files: **`doc-topics.csv`** and **`topic-terms.csv`**. Click and download the topic modeling output.

For Reference the extracted CSV files are available via the following GitHub directory:

https://github.com/TrainingByPackt/Machine-Learning-with-AWS/blob/master/lesson3/activity/SOLUTION__topic_modeling_output/doc-topics.csv

https://github.com/TrainingByPackt/Machine-Learning-with-AWS/blob/master/lesson3/activity/SOLUTION__topic_modeling_output/topic-terms.csv

> **Note**
>
> For this step, you may either follow along the exercise and type in the code or obtain it from the source code folder in the file **`local_csv_to_s3_for_analysis.py`** and paste it into an editor. For reference, the source code is available via GitHub in the following repository:https://github.com/TrainingByPackt/Machine-Learning-with-AWS/blob/master/lesson3/activity/local_csv_to_s3_for_analysis.py

40. Firstly, we will Import boto3

```
import boto3
```

41. Next, import pandas

```
import pandas as pd
```

42. Create the S3 client object.

```
s3 = boto3.client('s3')
```

43. Next, create a unique name for the s3 bucket to store your source CSV files. Here, the selected "**`unknown-tm-analysis`**" but you will need to create a unique name.

```
bucket_name = '<insert a unique bucket name>' #
```

44. Next, create a new bucket.

```
s3.create_bucket(Bucket=bucket_name)
```

45. Create a list of the CSV file names to import.

```
filenames_list = ['doc-topics.csv', 'topic-terms.csv']
```

46. Iterate on each file to upload to S3

```
for filename in filenames_list:
    s3.upload_file(filename, bucket_name, filename)
```

47. Next, check if the filename is '**doc-topics.csv**'

```
if filename == 'doc-topics.csv':
```

48. Now, get the **doc-topics.csv** file object and assign it to the '**obj**' variable.

```
obj = s3.get_object(Bucket=bucket_name, Key=filename)
```

49. Next, read the csv obj and assign it to the **doc_topics variable**.

```
        doc_topics = pd.read_csv(obj['Body'])
    else:
        obj = s3.get_object(Bucket=bucket_name, Key=filename)
        topic_terms = pd.read_csv(obj['Body'])
```

50. Merge files on topic column to obtain the most common terms per document.

```
merged_df = pd.merge(doc_topics, topic_terms, on='topic')
```

51. Print the **merged_df** to the console

```
print(merged_df)
```

52. Next, navigate to the location of the CSV's in a command prompt, and execute the code with the following command:

```
"python local_csv_to_s3_for_analysis.py"
```

53. The console output is a merged dataframe that provides the docnames with their respective terms and the term's weights (see below):

	docname	topic	proportion	term	weight
0	cv535_21183.txt:18	11	1.0	fall	0.055911
1	cv535_21183.txt:18	11	1.0	love	0.053333
2	cv535_21183.txt:18	11	1.0	flat	0.030461
3	cv535_21183.txt:18	11	1.0	joke	0.024031
4	cv535_21183.txt:18	11	1.0	eddie	0.010589
5	cv535_21183.txt:18	11	1.0	woman	0.009641
6	cv535_21183.txt:18	11	1.0	comedy	0.011022
7	cv535_21183.txt:18	11	1.0	attempt	0.009758
8	cv535_21183.txt:18	11	1.0	eventually	0.008582
9	cv535_21183.txt:18	11	1.0	story	0.011074
10	cv053_23117.txt:47	11	1.0	fall	0.055911
11	cv053_23117.txt:47	11	1.0	love	0.053333
12	cv053_23117.txt:47	11	1.0	flat	0.030461
13	cv053_23117.txt:47	11	1.0	joke	0.024031
14	cv053_23117.txt:47	11	1.0	eddie	0.010589
15	cv053_23117.txt:47	11	1.0	woman	0.009641
16	cv053_23117.txt:47	11	1.0	comedy	0.011022
17	cv053_23117.txt:47	11	1.0	attempt	0.009758
18	cv053_23117.txt:47	11	1.0	eventually	0.008582
19	cv053_23117.txt:47	11	1.0	story	0.011074
20	cv849_17215.txt:29	11	1.0	fall	0.055911

Figure 3.74: Activity merged topic modeling output

Chapter 4: Creating a Chatbot with Natural Language

Activity 5: Creating a custom bot and configure the bot

1. If you are creating your first bot, choose **Get Started**. Otherwise, choose Bots, and then choose **Create**.
2. On the **Create** your **Lex bot** page, choose Custom bot and provide the following information:

- **App name**: PizzaOrderingBot
- **Output voice**: Salli
- **Session timeout** : 5 minutes.
- **Child-Directed**: Choose the appropriate response.

3. Choose **Create**.
4. The console sends Amazon Lex a request to create a new bot. Amazon Lex sets the bot version to **`$LATEST`**. After creating the bot, Amazon Lex shows the bot **`Editortab`**:

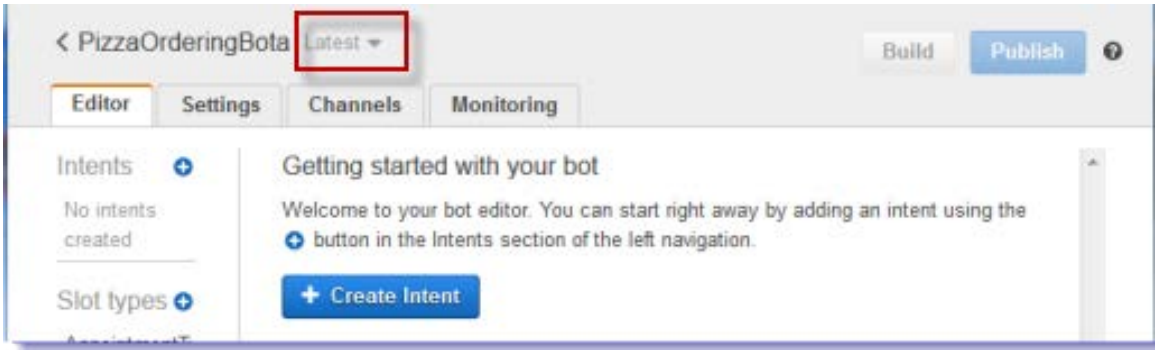


Figure 4.50: EditorTab

5. In the Amazon Lex console, choose the plus sign (+) next to Intents, and then choose **Create** new intent.
6. In the **Create** intent dialog box, type the name of the intent (**`OrderPizza`**), and then choose **Add**.

7. In the left menu, choose the plus sign (**+**) next to Slot types. In the **Add** slot type dialog box, add the following:

- **Slot type name** – Crusts
- **Description** – Available crusts

8. Choose Restrict to Slot values and Synonyms

- **Value** – Type thick. Press tab and in the Synonym field type stuffed. Choose the plus sign (+). Type thin and then choose the plus sign (**+**) again.

  The dialog should look like this:

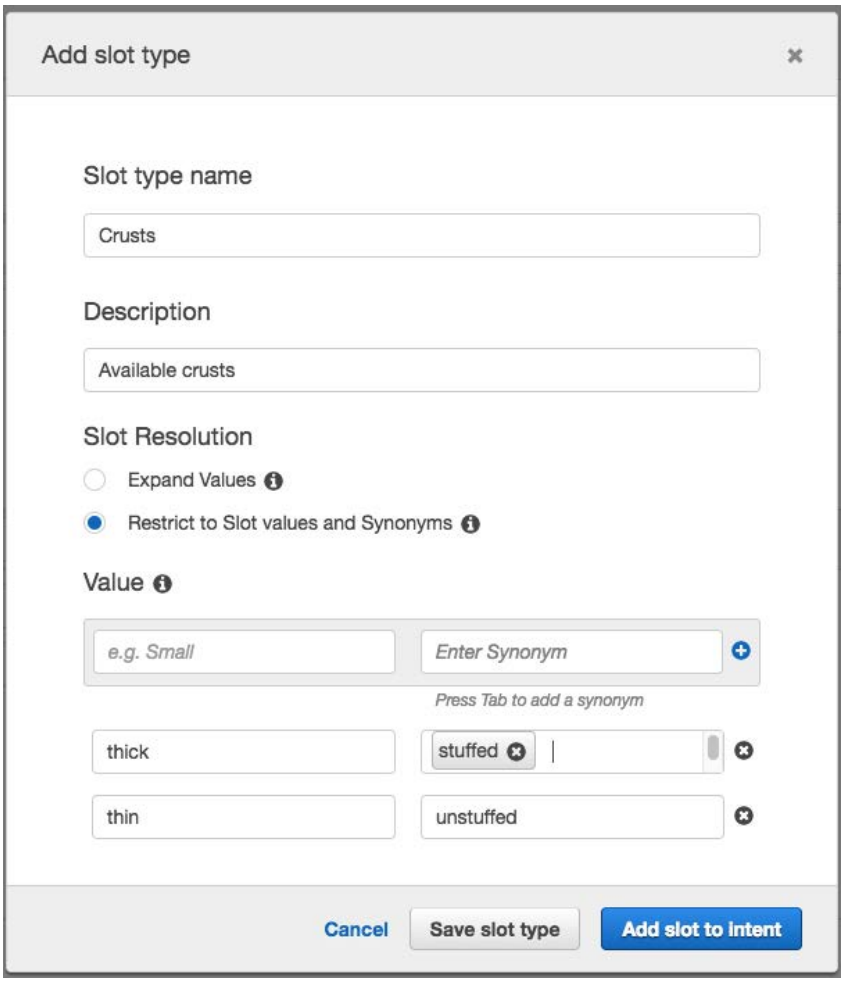


Figure 4.51: Dialog box for slot

9. Choose **Add slot to intent**.
10. On the **Intent** page, choose **Required**. Change the name of the slot from **`slotOne`** to **`crust`**. Change the prompt to **`What kind of crust would you like`**?
11. Repeat Step 1 through Step 4 using the values in the following table:

Name	Description	Values	Slot name	Prompt
Sizes	Available sizes	small, medium, large	size	What size pizza?
PizzaKind	Available pizzas	veg, cheese	pizzaKind	Do you want a veg or cheese pizza?

Figure 4.53: Value for slot and prompt

12. On the OrderPizza configuration page, configure the intent as follows:
13. Sample utterances – Type the following strings. The curly braces {} enclose slot names.

- I want to order pizza please
- I want to order a pizza
- I want to order a {pizzaKind} pizza
- I want to order a {size} {pizzaKind} pizza
- I want a {size} {crust} crust {pizzaKind} pizza
- Can I get a pizza please
- Can I get a {pizzaKind} pizza
- Can I get a {size} {pizzaKind} pizza

14. **Lambda initialization and validation** – Leave the default setting.
15. **Confirmation prompt** – Leave the default setting.

16. **Fulfillment** – Perform the following tasts:

- Choose AWS **`Lambda function`**.
- Choose **`PizzaOrderProcessor`**.
- If the **Add** permission to Lambda function dialog box is shown, choose **OK** to give the **`OrderPizza`** intent permission to call the **`PizzaOrderProcessorLambda`** function.
- Leave **None** selected.

  The intent should look like the following:

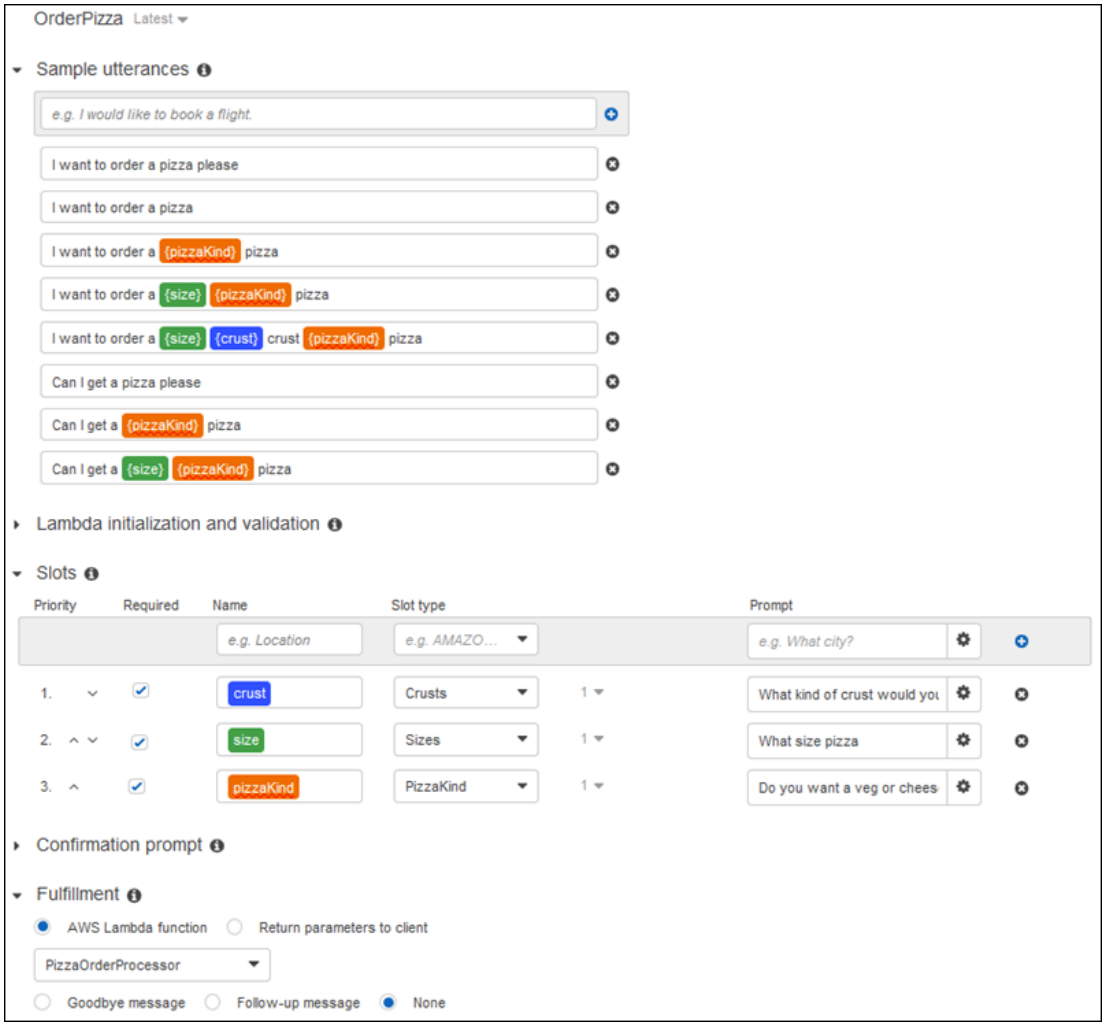


Figure 4.54: Utterance and slot editor

17. Configure error handling for the **`PizzaOrderingBot`** bot.
18. Navigate to the **`PizzaOrderingBot`** bot. Choose Editor. And then choose **`Error Handling`**.

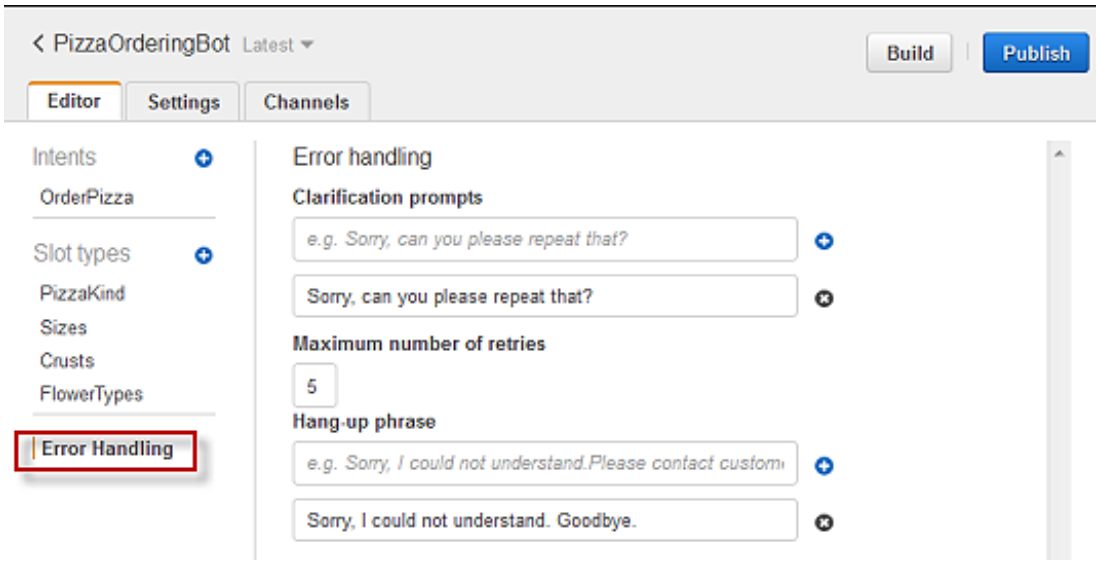


Figure 4.55: Editor for bot

19. Use the Editor tab to configure bot error handling.

- Information you provide in Clarification Prompts maps to the bot's clarificationPrompt configuration.
- When Amazon Lex can't determine the user intent, the service returns a response with this message
- Information that you provide in the Hang-up phrase maps to the bot's abortStatement configuration.
- If the service can't determine the user's intent after a set number of consecutive requests, Amazon Lex returns a response with this message.

20. Leave the defaults.
21. To build the **`PizzaOrderingBot`** bot, choose **Build**.
22. Amazon Lex builds a machine-learning model for the bot. When you test the bot, the console uses the runtime API to send the user input back to Amazon Lex. Amazon Lex then uses the machine learning model to interpret the user input.It can take some time to complete the build.

23. To test the bot, in the Test Bot window, start communicating with your Amazon Lex bot. For example, you might say or type:

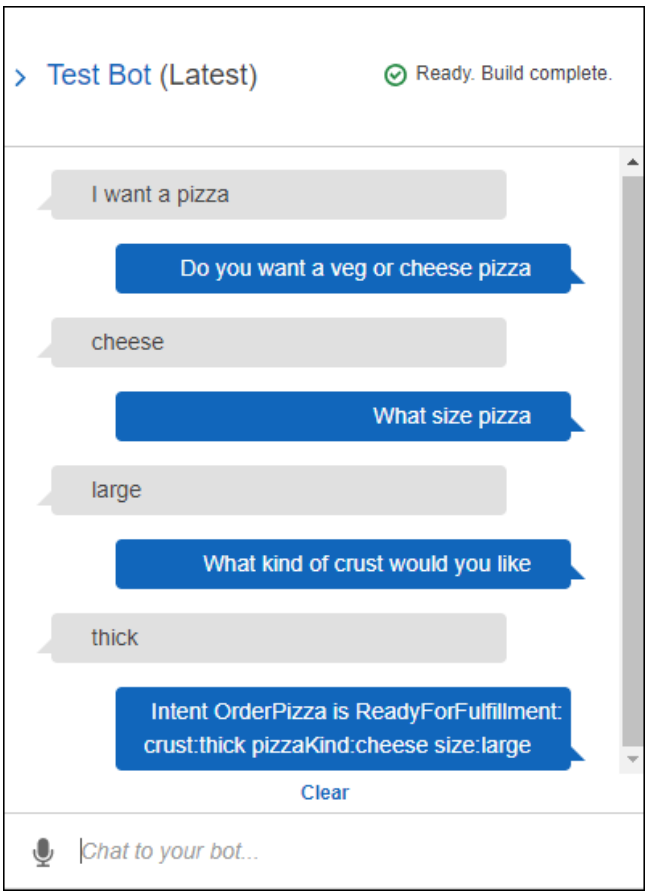


Figure 4.56: Test Bot

24. Use the sample utterances that you configured in the **`OrderPizza`** intent to test the bot. For example, the following is one of the sample utterances that you configured for the PizzaOrder intent:

```
I want a {size} {crust} crust {pizzaKind} pizza
```

To test it, type the following:

```
I want a large thin crust cheese pizza
```

When you type "I want to order a pizza," Amazon Lex detects the intent (**`OrderPizza`**). Then, Amazon Lex asks for slot information.After you provide all of the slot information, Amazon Lex invokes the Lambda function that you configured for the intent.The Lambda function returns a message ("Okay, I have ordered your ...") to Amazon Lex, which Amazon Lex returns to you..

Inspecting the Response

Underneath the chat window is a pane that enables you to inspect the response from Amazon Lex. The pane provides comprehensive information about the state of your bot that changes as you interact with your bot.

The contents of the pane show you the current state of the operation.

> **Dialog State** – The current state of the conversation with the user. It can be ElicitIntent, ElicitSlot, ConfirmIntent or Fulfilled.

Summary – Shows a simplified view of the dialog that shows the slot values for the intent being fulfilled so that you can keep track of the information flow. It shows the intent name, the number of slots and the number of slots filled, and a list of all of the slots and their associated values.

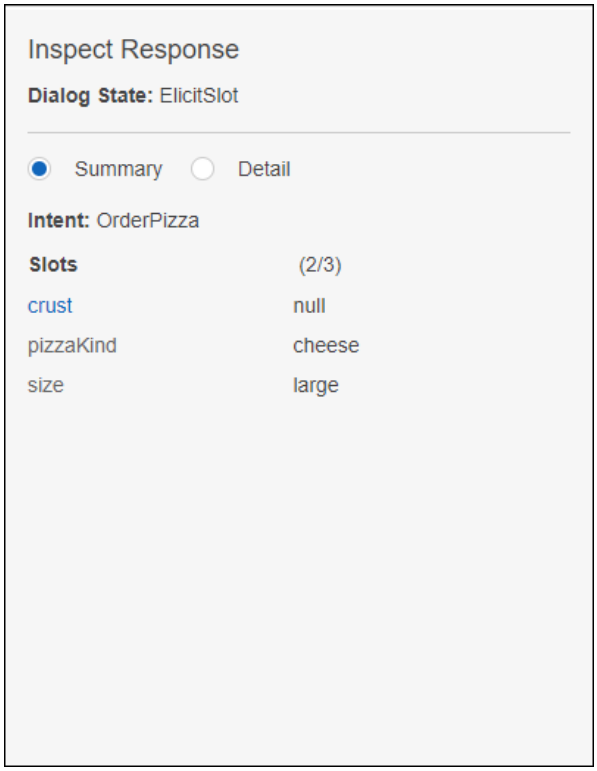


Figure 4.57: Summary inspect Response

Detail – Shows the raw JSON response from the chatbot to give you a deeper view into the bot interaction and the current state of the dialog as you test and debug your chatbot. If you type in the chat window, the inspection pane shows the JSON response from the PostText operation. If you speak to the chat window, the inspection pane shows the response headers from the PostContent operation.



Figure 4.58: Detail inspect response

Chapter 5: Using Speech with the Chatbot

Activity 6: Creating a custom bot and connect the bot with Amazon Connect

Create an Amazon Lex bot

Create a custom bot to demonstrate the Press or Say integration with Amazon Connect. The bot prompts callers to press or say a number that matches the menu option for the task to complete. In this case, the input is checking their account balance.

1. Open the Amazon Lex console.
2. If you are creating your first bot, choose Get Started. Otherwise, choose Bots, Create.
3. On the Create your Lex bot page, choose Custom bot and provide the following information:

- **Bot name** – For this demo, name the bot AccountBalance.
- **Output voice** – Select the voice for your bot to use when speaking to callers. The default voice for Amazon Connect is Joana.
- **Session timeout** – Choose how long the bot should wait to get input from a caller before ending the session.
- **COPPA** – Choose whether the bot is subject to the Child Online Privacy Protection Act.
- **User utterance storage** – Choose Store

4. Choose **Create**.

Configure the Amazon Lex bot

Determine how the bot responds to customers by providing intents, sample utterances, slots for input, and error handling.

1. **Create** intents

   For this example, configure the bot with two intents: one to look up account information, and another to speak with an agent.

- Choose the **+** icon next to Intents, and choose Create new intent.
- Name the intent **`AccountLookup`**.
- Create another **`intent`**, and name it **`SpeakToAgent`**.

2. **Add** sample utterances

   After defining the intents, add some sample utterances.

- Select the **`AccountLookup`** intent.
- Add a sample utterance, such as "**`Check my account balance`**" (don't include the quotes), and choose the **+** icon.
- Add a second utterance, "**`One`**" (without the quotes), and choose the **+** icon.
  This assigns the utterance of "one" or key press of "1" to the AccountLookup intent.

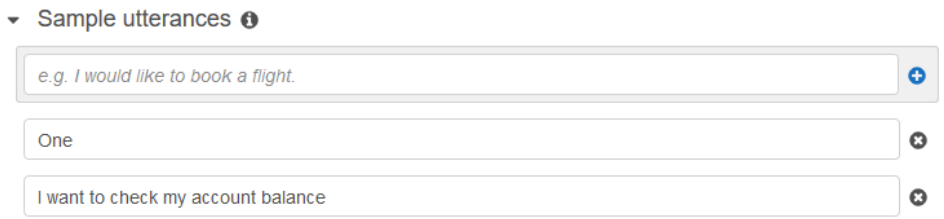


Figure 5.29: Sample utterance

- Select **`SpeakToAgent`**.
- Add a sample utterance, such as "**`Speak to an agent`**," and choose **+**.
- Add a second utterance, "**Two**" (without the quotes), and choose + icon.

3. **Add** slots

   Before the bot can respond with the caller's account balance, it needs the account number.

- Under Slots, add a slot named AccountNumber.
- For Slot type, select AMAZON.NUMBER.
- For Prompt, add the text to be spoken when the call is answered. To highlight the new DTMF support, ask callers to enter their account number using their keypad. For example, "**`Using your touch-tone keypad, please enter your account number`**."
- Make sure that the Required check box is selected, and choose the + icon.

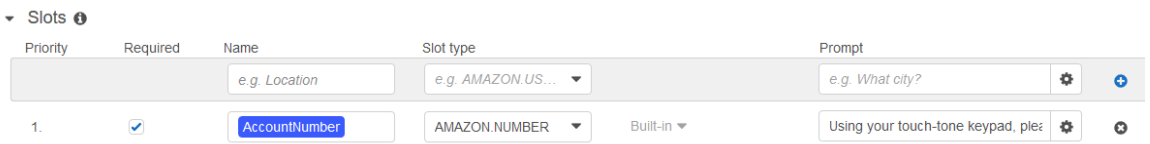


Figure 5.30: Slot Addition

4. **Add** responses

   Now that you have intents, utterances, and a slot, add the responses that the bot provides to callers. Because you are creating a simple bot for this example, you are not hooking up the bot to look up real customer data. The example bot responds with text strings that you add, regardless of the account number that a caller provides.

- Select the **`AccountLookup`** intent.
- In the Response section, add a message for the bot to say to customers. For example, "**`The balance for your account is $2,586.34`**."
- Choose Save Intent.
- For the **`SpeakToAgent`** intent, add a message that lets callers know that their call is being connected to an agent. For example, "**`Okay, an agent will be with you shortly`**."
- Choose **Save** Intent.

5. Build and Test the Amazon Lex bot

   After you create your bot, make sure it works as intended before you publish it.

- To enable the Test Bot window, choose Build. It may take a minute or two.
- When it is finished building, choose Test Chatbot.

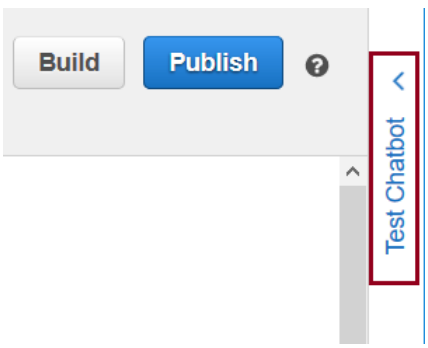


Figure 5.31: Publish bot

- In the Test Chatbot pane, type messages in the chat window.
- To test the AccountLookup intent, type "1" (without the quotes), and then type an account number.

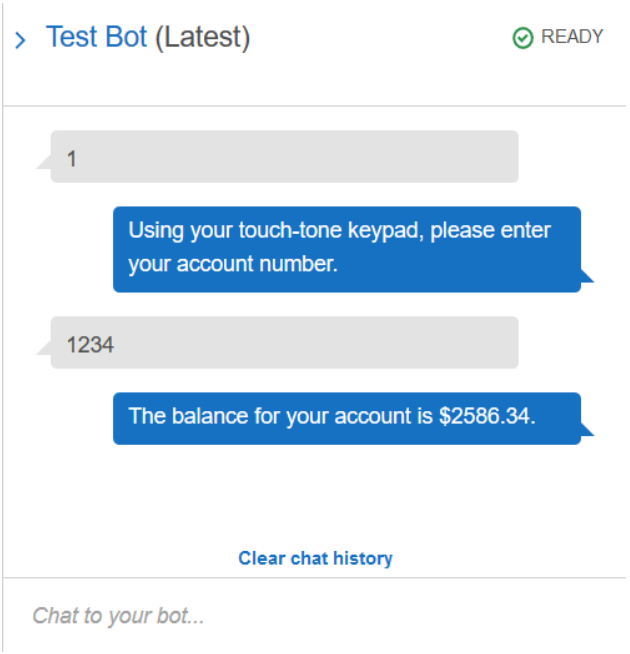


Figure 5.32: Test Bot

- To confirm that the SpeakToAgent intent is working, type "2" (without the quotes).

6. Publish the Amazon Lex bot and create an alias

- Next, publish the bot so that you can add it to a contact flow in Amazon Connect.
- Choose **Publish**.
- Provide an alias for your bot. Use the alias to specify this version of the bot in the contact flow, for example, Blog.

- Choose **Publish**.

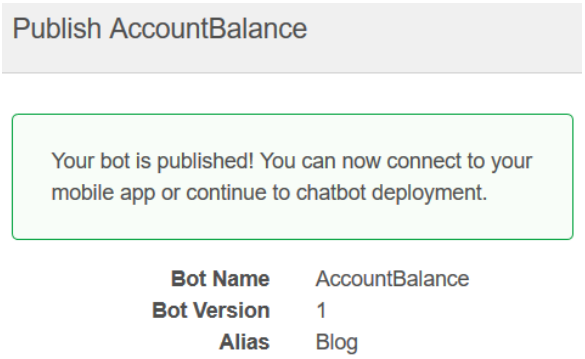


Figure 5.33: Publish Bot

7. Add the Amazon Lex bot to an Amazon Connect instance

- To use a bot in your contact flow, add the bot to your Amazon Connect instance. You can only add bots created under the same AWS account and in the same Region as your instance.
- Open the Amazon Connect console.
- Select the Instance Alias of the instance to which to add the bot.
- Choose Contact flows.
- Under Amazon Lex, choose **+ Add** Lex Bot.
- Select the **`AccountBalance`** bot and choose Save Lex Bots. If you published the bot after you opened the settings for your instance, reload the page to get it to show up.

8. Create a contact flow and add your Amazon Lex bot

   Next, create a new contact flow that uses your Amazon Lex bot. Create the contact flow When you create the contact flow, you can configure the message played to callers.

- Log in to your Amazon Connect instance with an account that has permissions for contact flows and Amazon Lex bots.

- Choose **`Routing`**, **`Contact flows`**, **`Create contact flow`**, and **`type a name`**.
- Under Interact, drag a Get customer input block onto the designer, and connect it to the Entry point block.
- Open the Get customer input block, and choose Text to speech (Ad hoc), Enter text.
- Type a message that provides callers with information about what they can do. For example, use a message that matches the intents used in the bot, such as "To check your account balance, press or say 1. To speak to an agent, press or say 2."

9. Add the Amazon Lex bot to your contact flow

   The bot is defined as the method of getting customer input.

- In the Get customer input block, select Amazon Lex.
- For Name, use **`AccountBalance`**. For Alias, use **Blog**.
- To specify the intents, choose Add a parameter under Intents.
- Type AccountLookup, and choose Add another parameter.
- Type SpeakToAgent, and choose Save.

10. Finish the contact flow

- After the caller interacts with the bot, finish the contact flow to complete the call for the customer.
- If the caller presses 1 to get their account balance, use a Prompt block to play a message and disconnect the call.
- If the caller presses 2 to speak to an agent, use a Set queue block to set the queue and transfer the caller to the queue, which ends the contact flow.
- To complete the AccountLookup intent:
- Under Interact, drag a Play prompt block to the designer, and connect the AccountLookup node of the Get customer input block to it. After the customer gets their account balance from the Amazon Lex bot, the message in the Play prompt block plays.
- Under Terminate/Transfer, drag a Disconnect / hang up block to the designer, and connect the Play prompt block to it. After the prompt message plays, the call is disconnected.

- To complete the **SpeakToAgent** intent:
- Add a Set queue block and connect it to the **SpeakToAgent** node of the Get customer input block.
- Add a Transfer to queue block and connect the Set queue block Success and Error nodes to it. You could also add a message that plays when the call cannot be transferred because the queue is full or an error occurs.
- Choose **Save** & **Publish**.

  Your finished contact flow looks something like the following:

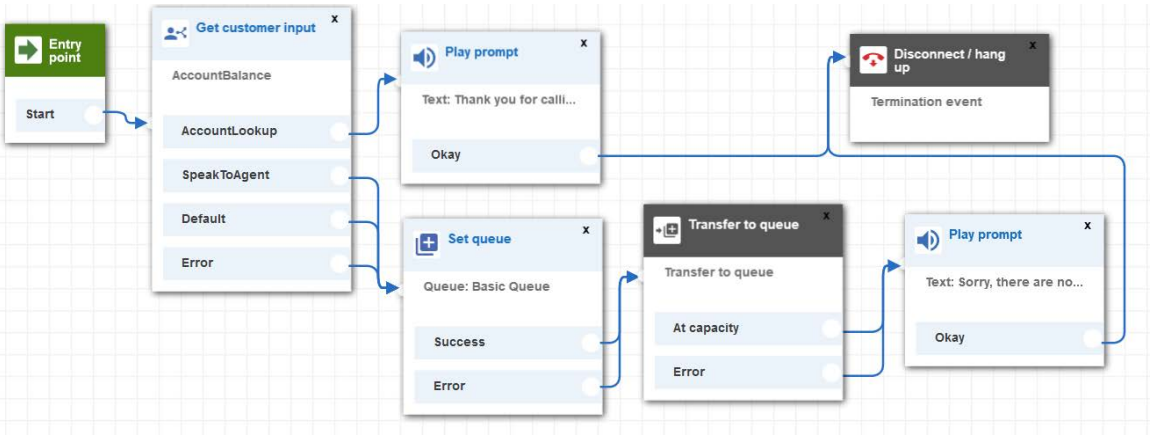


Figure 5.34: Contact Flow

11. Assign the contact flow to a phone number

- When callers call in to your contact center, the contact flow to which they are sent is the one assigned to the telephone number that they dialed. To make the new contact flow active, assign it to a phone number for your instance.
- Open the Amazon Connect Dashboard.
- Choose View phone numbers.
- Select the phone number to which to assign the contact flow.
- Add a description.
- In the Contact flow/IVR menu, choose the contact flow that you just created.
- Choose **Save**.

Chapter 6: Analyzing Images with Computer Vision

Activity 7: Compare faces in your own images

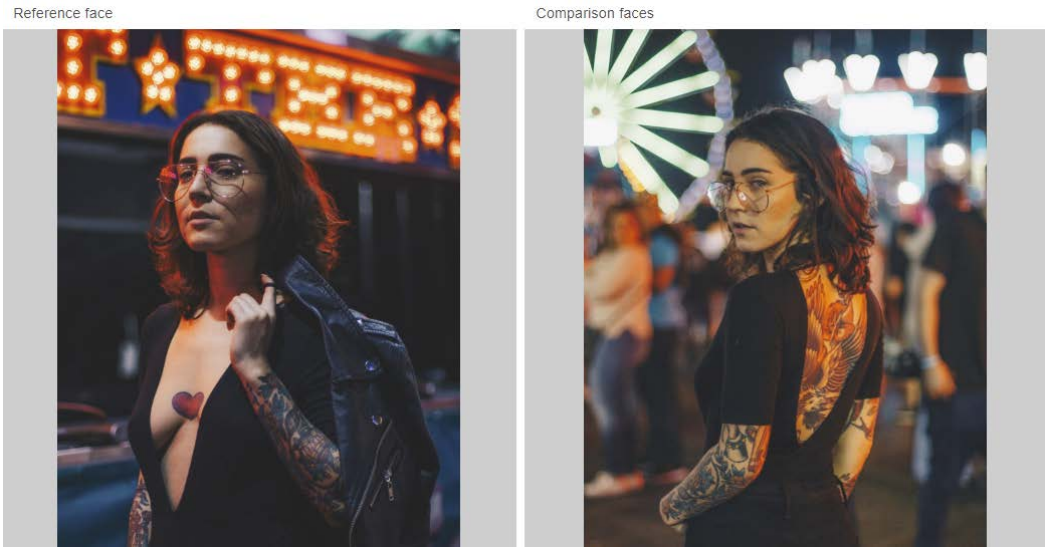


Figure 6.49: First provided images for Face comparison

1. Rekognition is able to recognize that the faces are of the same person with a 98% degree of confidence, even with different angles, lighting and position of glasses on the face



Figure 6.50: Results for first provided images for Face comparison

2. The second set of images are: https://images.unsplash.com/photo-1526510747491-58f928ec870f?w=600 and https://images.unsplash.com/photo-1529946179074-87642f6204d7?w=600 .

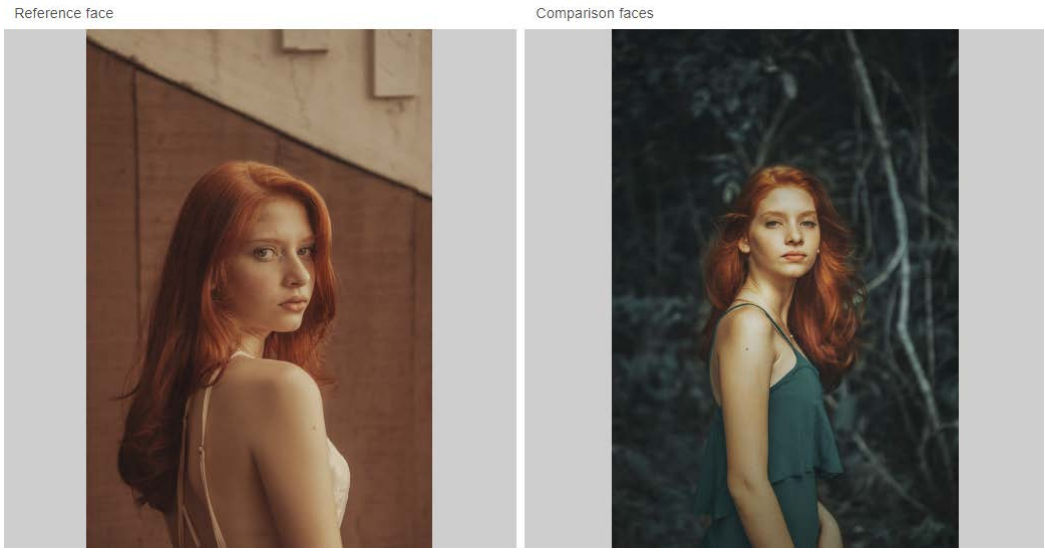
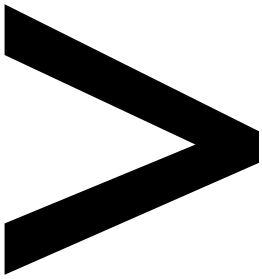


Figure 6.51: Second provided images for Face comparison

3. Once again, Rekognition recognizes the faces with a **96%** degree of confidence, even at different angles.



Figure 6.52: Results for second provided images for Face comparison



Index

About

All major keywords used in this book are captured alphabetically in this section. Each one is accompanied by the page number of where they appear.

A

algorithm: 36, 64
algorithms: 2
allocation: 64
amazon: 1-6, 8-10, 12, 17, 24-27, 29-33, 35, 38, 41-42, 44, 46, 48, 50-51, 56, 58, 60, 63-64, 66-69, 73, 75, 87, 91-92, 95-97, 105, 108, 110, 120-121, 127, 129, 131-139, 141, 143-144, 146, 150-151, 154-155, 157-161, 163, 166-168, 177, 182, 184-185, 189
amazonaws: 39
amazons: 10
amount: 3, 133, 163
amounts: 3, 5
analytics: 3-4
analyze: 3, 29-31, 39, 41, 43, 63-64, 87-88, 92, 155, 157-158, 163, 168, 171, 178, 182-183, 185, 189
analyzed: 4, 33, 35, 168, 172
analyzing: 44, 63-64, 88, 157, 172, 184
awsapps: 141
aws-based: 60
aws-ml-s: 47, 52, 54

B

browser: 5, 9, 146
browsing: 92
bucket: 5-13, 15, 22-23, 44, 46-47, 52-54, 58-59, 66, 69-71, 74, 77-78, 84-88, 138
buckets: 5-6, 10, 79, 138

C

chatbot: 3, 23, 26, 58, 87, 91-93, 96-97, 99, 102-107, 109, 120-121, 125-129, 131-135, 144-145, 148, 150-152, 154-155, 158
chatbots: 2-3, 88, 91-93, 129, 131, 143-144, 155
chrome: 5
cloudwatch: 51, 59, 138
comprehend: 4, 26, 29-33, 35-44, 46, 55-58, 60, 63-68, 75, 79, 87-88, 92
copyright: 164
corpus: 64, 66

D

dashboard: 10, 56, 127, 143, 145-147, 153
database: 2, 30
dataframe: 85
datasets: 5
decrypt: 138
docnames: 85
doc-topics: 67-68, 82-86
document: 4, 26, 30-33, 38, 42, 46, 55, 58, 60, 63-68, 77, 85, 87, 185
documented: 95, 105
documents: 4, 23, 26-27, 29-31, 35-36, 40, 58, 63-64, 66-68, 87-88
dominant: 31-35
dominates: 2
download: 16, 19, 73, 81, 87, 164
downloaded: 20
dragging: 148
dropdown: 146, 149, 153-154
dual-tone: 143
duplicates: 135
durability: 3, 8

E

encrypt: 138
encryption: 11, 14
endoffset: 38, 40, 59
endpoint: 5, 126
errors: 4
export: 1, 5, 12, 16, 23, 27
exported: 23
exporting: 2-3, 5, 12, 16, 23
extent: 92

F

filename: 54, 84-85, 87-88
filenames: 84, 88
filesystem: 5-6
frameworks: 65
free-tier: 3-4
function: 4, 29, 33, 44-46, 48-51, 53-54, 56, 58, 69, 104, 107, 120-121, 123-128, 145, 158

G

github: 33, 35, 38, 40-42, 54, 73, 82, 84, 87
google: 2

I

import: 1, 5, 9, 12, 22-23, 27, 33, 35, 39, 41, 43, 54, 84, 86-88, 110, 113, 126

important: 3-4, 36, 64, 127, 132, 159, 177
imported: 4, 22, 29, 44, 110
importing: 2-3, 5, 12, 22-23, 110
interface: 1-3, 7-8, 17, 23-24, 26-27, 33, 35, 92, 96-97, 107, 132, 158
interfaces: 8, 158
iterate: 84, 87
itself: 22, 158, 160

K

keyphrases: 59

L

lambada: 29
lambda: 4, 29-30, 44-46, 48-55, 58, 104, 107, 120-121, 123-125, 127-128, 133, 144-145, 155, 158
lex-based: 108, 127
library: 33, 35, 39, 41, 43, 158

M

mybucket: 5-7, 23

N

network: 5
networks: 93, 159
neural: 159
neutral: 42, 58-59

O

object: 3, 5-8, 14-15, 22, 36, 50, 54, 84-85, 104-105, 125, 160-161, 163-164, 166
offbeat: 64
on-demand: 2
operate: 60
operating: 44
operation: 23, 35, 38, 41-42, 166
operations: 7-8, 31, 44ptimize: 4
output: 20, 26, 32, 34, 38-40, 42, 44, 55, 58-59, 64, 66-69, 74, 77-85, 87, 109, 148-149, 152, 187
outputs: 67, 152

P

packages: 107
pandas: 84, 88
parsed: 23
parses: 107
partial: 26
phrase: 41, 102
phrases: 26, 29-30, 36, 40-42, 55-56, 58-60
pickupdate: 100
pickuptime: 100
processing: 2, 27, 30, 45, 60, 88, 124, 145, 158-159
produce: 30, 34, 65
product: 30, 185
products: 3, 92
program: 7, 23, 26
programmed: 2
progress: 80
progressed: 92
prompt: 20-21, 34, 85, 94, 100, 106-107, 114-115, 118-119, 128, 145, 152
prompted: 20, 72, 100
prompts: 94, 102, 115, 145
properties: 14, 69, 148-152, 158
property: 11, 14, 149, 152
protect: 20, 109, 159
protection: 109
provide: 2-3, 26, 30, 32, 39, 41, 43, 60, 64, 66, 87, 92, 94, 106, 111, 114-115, 139, 143, 160, 169, 174, 189
provisions: 109
public: 5, 17, 158
python: 17, 23, 33-36, 39-45, 49, 54, 85, 122, 124-125

Q

quotes: 105

R

read-only: 50
receiving: 23
recently: 151, 189
recognize: 1, 106-108, 110-112, 117, 157-158, 160, 173, 176-177, 184-185, 189
recognized: 100, 104, 112, 118, 182, 185-189
recurrent: 159
replicated: 8
replicates: 8
report: 64-66
request: 20, 94, 125-126, 160
requests: 8
resolved: 51
resource: 51, 136-139

resources: 2, 26, 105
restores: 64
retain: 36
retries: 102, 115
retrieval: 3
retrieve: 2-4, 16, 105, 125-126, 159
retrieved: 24
retrieves: 158
revealed: 144
revealing: 167
reveals: 108
review: 26, 40, 64, 87, 139
reviewed: 36
reviews: 30, 64, 87
112, 148, 164
rootkey: 20
routing: 147, 153
running: 4, 96, 107
runtime: 45, 49, 122, 124

S

saving: 53, 71, 116
savings: 4
scheduling: 92
secure: 2
security: 3, 18
seniment: 44
server: 3-4, 11, 44-45
serverless: 133
servers: 8, 44
session: 109
setting: 14, 44, 46, 96, 136, 139, 144, 150
settings: 5, 14, 53, 69, 133-134, 138-139, 144, 158
setups: 17
shared: 134
sharing: 167
storable: 5
storage: 2-6, 9, 14
stored: 3, 5, 7, 138
stores: 6
storing: 4-5, 45
syntax: 30
system: 3, 5, 44, 94, 104-105, 107, 111, 160, 177
systems: 92

T

technology: 129, 158
trigger: 47, 52, 54
triggers: 51-52

U

urllib: 126
urlopen: 126
utterance: 94, 106-107, 117-120, 144-145, 155
utterances: 94, 99-100, 105, 111-112, 115, 117, 119

V

version: 126, 128, 151, 155
versioning: 11

W

web-based: 24
web-page: 133
website: 4-6, 24, 30, 64
window: 11, 109-110, 139, 187
workflow: 106, 129
workflows: 4, 132