Understanding Consumer Perception For A Product

A solution deployed on AWS, to explore tweets related to iPhoneX

MSBA 6330 - Big Data Analytics

Section 1 - Team 6

Hui Dong

Jing Liu

Neelakanteshwar Reddy Rebala

Prathamesh Dalvi

Udit Ranjan

Table Of Contents

Introduction

The goal of this project is to demonstrate, as a proof-of-concept, a methodology to easily and effectively understand how consumers tend to perceive a product. For our project, we used freely available data, in the form of Twitter tweets, to try and gauge whether we can build a product that has large-scale text processing at it's heart, is easily scalable, has applications across domains, and is intuitive to use for non-technical users.

The motivation to work on this project stemmed from the fact that companies need to understand how customers are perceiving their products and services. Social media is a channel that customers often use to voice their opinions about products, discuss features with their network and interact directly with the companies. This, combined with internal company data like sales figures and promotion campaigns, can give companies a more holistic view about their products.

We have used Amazon AWS services to design and implement the various parts of our project. We will describe in this paper, in detail, how we used services like AWS Kinesis, DynamoDB, Lambda and Elasticsearch to design and create a serverless architecture for our solution.

Background

Problem Context

Apple recently launched it's latest phone, the iPhoneX, which the company claims is the best rendition of the popular iPhone brand. As a company, Apple would want to track what features consumers love and dislike, what's the sentiment towards the brand and the phone, and which locations are contributing to the chatter the most. We used Apple as an example in our project, however, this analysis can be extended to multiple domains, and prove to be valuable to any B2C company.

Type of Data

We used semi-structured data obtained from the social media website, Twitter. The data is usually available through an API, in the form of JSON objects. We specifically tracked data pertaining to iPhoneX and written in English language.

Big Data tools

All Big Data tools used in this project are AWS services. We used the following services:

1. **Kinesis** - Consumes streaming data from Twitter, and performs pre-processing
2. **DynamoDB** - The NoSQL database used to store tweets spewed out by Kinesis
3. **Lambda** - Custom python code deployed on Lambda, used to generate additional features and copy data from DynamoDB to Elasticsearch
4. **Elasticsearch** - An open-source search engine, used to index text data like tweets, locations, usernames etc.
5. **Kibana** - Dashboard to visualize the data indexed in Elasticsearch

Data

The Twitter Streaming API, by default, has several fields in the JSON object it provides. For the purpose of our analysis, we chose to work with only a subset of the fields. In addition, we created some new features like brand, company and sentiment. These features were added as fields. Following is a table with all the fields used, along with their descriptions.

S No.	Field	Description
1	entities.hashtags	Hashtags present in tweets
2	text	The actual content of the tweet
3	created_at	The time a tweet was created
4	user.location	Location of the user
5	id	Unique ID of the tweet, provided by Twitter
6	in_reply_to_status_id	If a tweet was a reply to another tweet
7	reply_count	Number of replies a tweet got
8	in_reply_to_screen_name	Name, if a tweet was a reply to another tweet
9	user.name	A user's unique name
10	user.followers_count	Number of followers a user has
11	@timestamp	An Elasticsearch specific field

12	user.id	A user's unique ID
13	user.created_at	A user's unique name
14	retweet_count	The count of the particular tweet has been redirected
15	user.screen_name	user's screen name
16	favorite_count	The count of likes
17	sentiment	Sentiment of a tweet
18	brand	The brand mentioned in a tweet. Eg: iPhoneX
19	company	The company related to the brand. Eg: Apple
20	features	Features being tracked. Eg: camera, OS, Face ID

Methodology

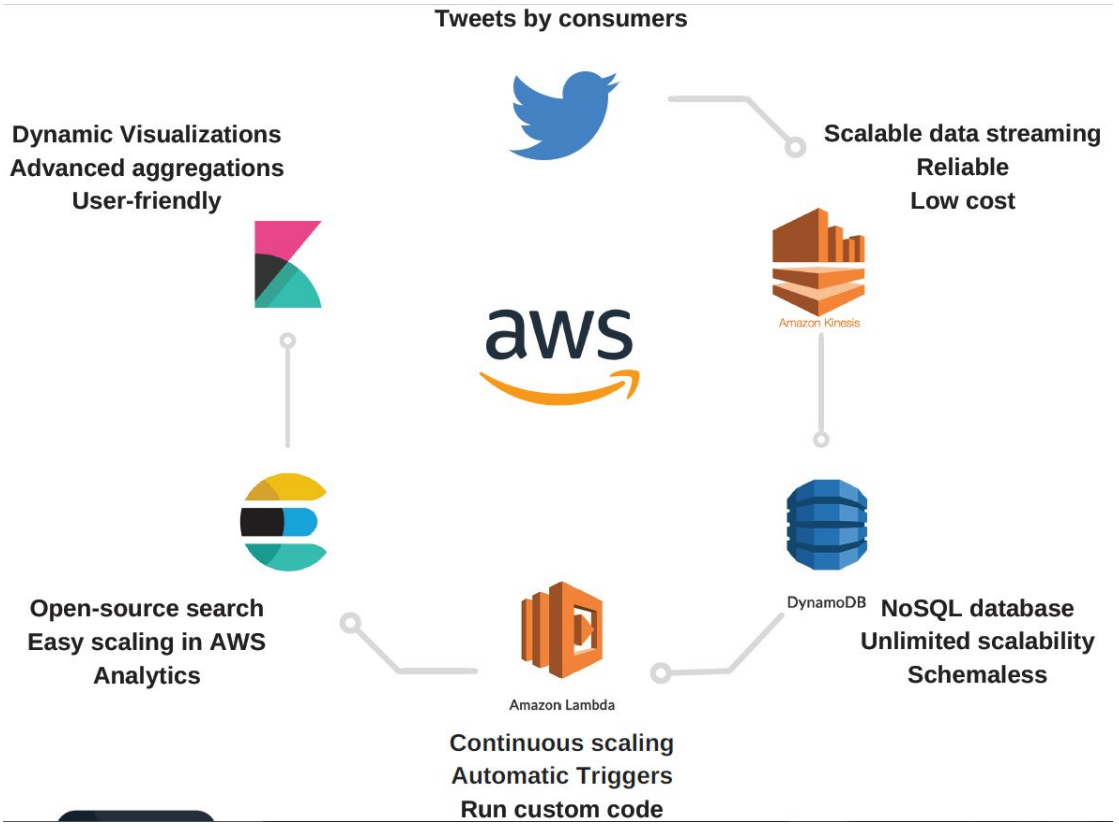


Fig: The Solution Architecture, deployed on AWS

To implement our solution, we broke our process down into 4 distinct steps:

Data Retrieval

The first step in the process is to retrieve data from Twitter. We setup a twitter developer app on https://apps.twitter.com/, and created certain access keys, to start using the Twitter streaming service. The tweets were consumed by the Amazon Kinesis streaming service, in real-time, that we setup on AWS. We used the boto3 Python library to create this Kinesis stream and pre-process the data by keeping only those fields that were essential for our analysis. In addition, we calculated the sentiments associated with each tweet using the TextBlob library.

Data Storage

Once we had the relevant data from the Kinesis stream, the next step in the process was storing this data. We chose the NoSQL database, DynamoDB as our primary data store, as it's schemaless and works well with unstructured, textual data. Therefore, as soon as a new tweet becomes available in the Kinesis stream, it is stored in DynamoDB, for future use.

Data Indexing

The most challenging, important and fulfilling part of this project was to figure out how to make the data that we were collecting in real-time, available for search and analysis. We first setup AWS Lambda service to write custom Python code. Then, we wrote a script that runs every few minutes and indexes the data from DynamoDB to Elasticsearch.

To map the data in DynamoDB into an index in Elasticsearch (ES), we needed to define the schema of the ES index, using something called as a 'mapping'. Mapping is a JSON file that has information about how an index is constructed, what are the field types (based on the data that is expected), and how the data is to be analyzed at index time and search time. For instance if we want to perform word stemming for data in a particular field, we would define this requirement in the mapping file.

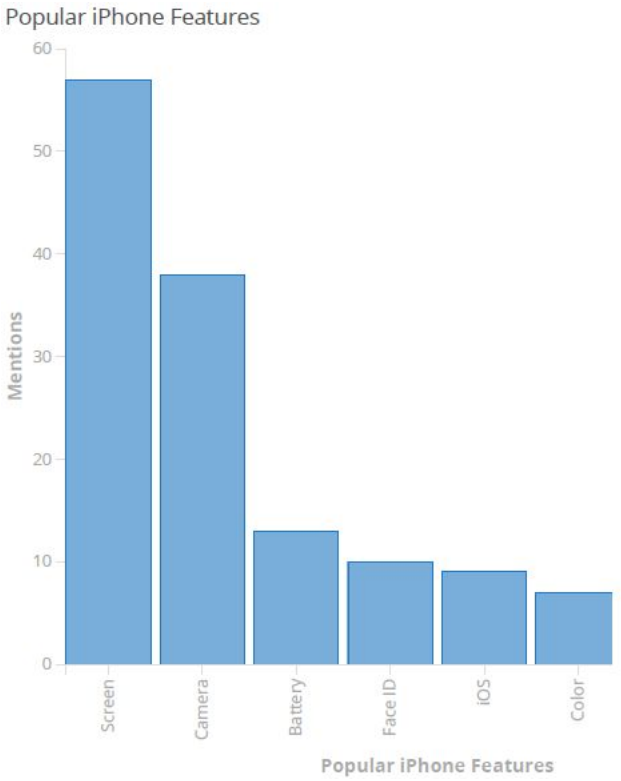
Our objective was to explore some advanced text-analytics capabilities for our project, that led us to implementing features like full-text search, partial search, case-insensitivity, contextual search.

Data visualization

Once the data was indexed in Elasticsearch, according to how we wanted to search and analyze it, it was now time to visualize the information. We used Kibana, a visualization engine that works on top of Elasticsearch to create a dashboard that was designed to monitor tweets in real-time and help users understand the

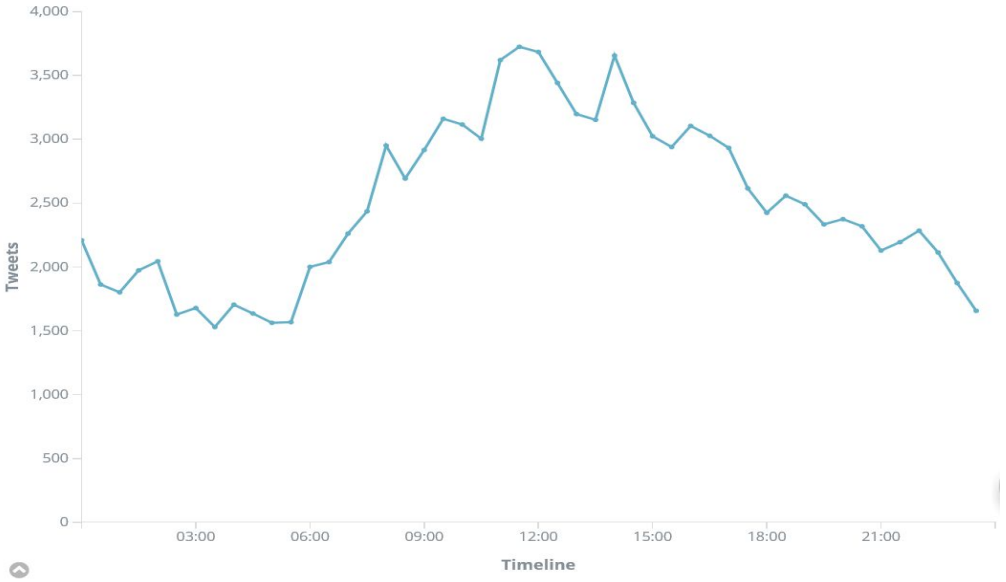
perception related to the iPhone. Following are some of the visualizations that we created:

Identifying Popular features discussed by Twitter Users



The above chart captures the number of times a particular phone feature was mentioned in a tweet.

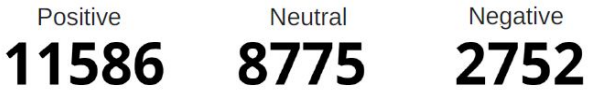
Brand Engagement



Understanding Consumer Perception For A Product - GitHub

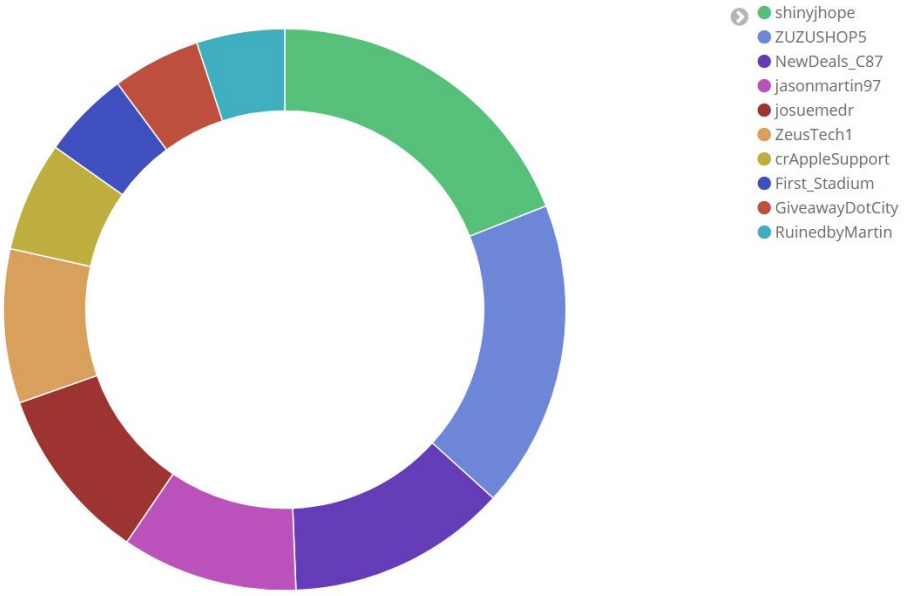
This chart captures the timeline of tweets, or the frequency with which people are tweeting about iPhone.

Customer perception



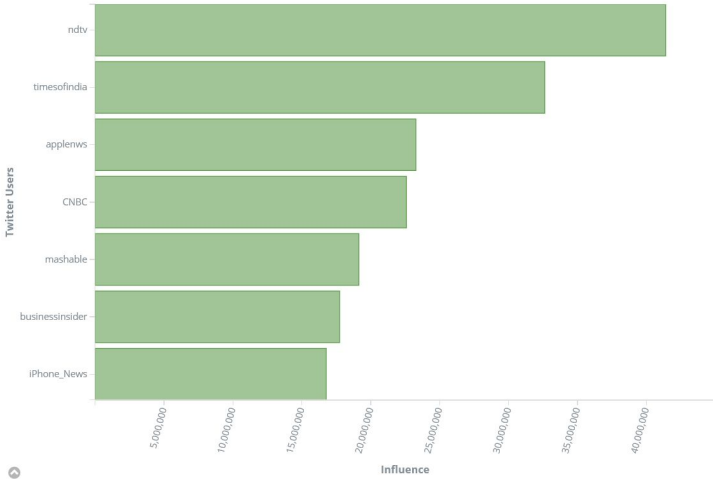
We tried to understand the consumer perception using sentiment analysis, in combination with features that people were talking about.

Brand Enthusiasts



People who tweet the most were tagged as Brand Enthusiasts

Potential Key Influencers



People who tweet often, and had a large network, were identified as key influencers. These are people that can be targeted by brands and companies to become Key Influencers or Brand Ambassadors.

Popular Hashtags

The word cloud was created to represent the most popular and trending hashtags.

Findings

Some of the key findings of our project are listed below:

1. We found that creating a dashboard to monitor any product or service, can prove to valuable for any company that believes in understanding their customers.
2. Deploying our solution on AWS meant that the solution can be scaled to incorporate vasts amount of data, and would provide companies a relatively less riskier option than deploying a solution with in-house hardware and services.
3. Customers' complain on Social Media - Though the iPhoneX was generally warmly welcomed by customers, we found several instances of complaints based on our analysis. Some of these complaints were related to people not being used to and being uncomfortable with the new Face ID feature, and others facing battery issues.

Future Scope - Project TakeAway

360 Product Overview

For companies, this solution represents an opportunity to build a holistic understanding of your products, by integrating social media with internal data like CRM and sales

Key Influencers

Find people who are brand advocates and have a large network, to be potential brand ambassadors.

Competitive Analytics

Understand your competitors strategy and engagement on social media. Such a platform can provide great competitive advantage to companies that can effectively measure their competitors' performance and strategies.

Policy Making

Gauge how the public reacts to important policy decisions. For politicians and government officials, social media can be a source of unbiased views, that can be used to understand public sentiment.

Bibliography

AWS
http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2_GetStarted.html
http://docs.aws.amazon.com/streams/latest/dev/introduction.html

http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html
http://docs.aws.amazon.com/lambda/latest/dg/welcome.html
http://docs.aws.amazon.com/elasticsearch-service/latest/developerguide/what-is-amazon-elasticsearch-service.html

Elasticsearch
https://www.elastic.co/guide/en/elasticsearch/guide/current/index.html
https://chrome.google.com/webstore/detail/elasticsearch-head/ffmkiejjmecolpfloofpjologoblkegm?hl=en-US

Kibana
https://www.elastic.co/guide/en/kibana/5.5/dashboard-getting-started.html

Appendix

GitHub
https://github.umn.edu/dalvi012/twitter-streaming-analytics
or
https://github.com/prathameshsdalvi/twitter-streaming-analytics

Set Up

Setting up EC2 Instance

Follow the instructions provided in the ec2_machine_setup.bash file. The instructions help you:

- Connect to your EC2 instance
  - Go to your AWS EC2 dashboard and ensure that your EC2 instance is running (you may use the EC2 instance available as a part of the free tier)
  - Right click on the instance, click on Connect and note down the Public DNS which will be used to SSH to the EC2 instance
  - Open Bash
  - Follow the steps as per ec2_machine_setup.bash to connect to the machine
- Install Python 2.7 and the Python libraries
- Set your AWS region and store your AWS Keys as environment variables
- Create project directories
- Create file with your Twitter App credentials
- Create keywords file with variables that store keywords for various analyses (refer to sample file keywords.py)

Creating AWS Kinesis Data Stream

The file create_kinesis_stream.py creates a kinesis stream with 1 shard. Read more about AWS Kinesis Data Streams here.

We use the Python boto3 library to do this. It will be used for creating most of the architecture.

```
# Create a kinesis stream
import boto3

client = boto3.client('kinesis')
response = client.create_stream(
   StreamName = 'twitter',
   ShardCount = 1
)
```

Feeding Twitter Stream to Kinesis

The file twitter_to_kinesis.py creates a producer which ingests the Twitter stream into the Kinesis stream that we created above.

```
# Feed data into a kinesis stream
from TwitterAPI import TwitterAPI
import boto3
import json
from twitterCreds import *
from keywords import *

api = TwitterAPI(consumer_key, consumer_secret, access_token_key, access_token_secret)

kinesis = boto3.client('kinesis')

# Keywords
terms = iphone

# Tweet Language
language = 'en'

# Request Twitter api and filter on keywords and language
request = api.request('statuses/filter', {'track':terms, 'language':language})

# Ingest into kinesis stream
while True:
  try:
    for item in request:
      if 'text' in item:
                kinesis.put_record(StreamName = "twitter", Data = json.dumps(item), PartitionKey = "filler")
  except:
    pass
```

Creating a Consumer to Test the Kinesis Stream

Create the simple_consumer.py file. This script creates a simple consumer that prints the data from the Kinesis stream.

```
# Consume the data
import boto3
import time
import json

kinesis = boto3.client("kinesis")
shard_id = "shardId-000000000000"
shard_it = kinesis.get_shard_iterator(StreamName = "twitter", ShardId = shard_id, \
    ShardIteratorType = "LATEST")["ShardIterator"]

# Infinite loop to read data from Kinesis stream
while True:
  out = kinesis.get_records(ShardIterator = shard_it, Limit = 1)
  shard_it = out["NextShardIterator"]
  print out;
  time.sleep(1.0)
```

To test your stream, run the following on the your current Bash window.

```
python ~/big_data_project/final_pipeline/twitter_to_kinesis.py
```

This starts feeding Twitter stream into the Kinesis Stream

Open another Bash window, SSH to the same EC2 machine and run your consumer.

```
python ~/big_data_project/testing/simple_consumer.py
```

If everything works well, executing the above commands should start printing the output to your BASH window which has the consumer running. Your output should look similar to the sample output given in kinesis_tweets.txt.

Creating DynamoDB Table

We now create a DynamoDB table which stores data from our stream (create_dynamodb_table.py).

```
# Create DynamoDB table
# Just id is required
# Schema need not be given since DynamoDB is built for unstructured data
import boto3
dynamodb = boto3.resource('dynamodb')
table = dynamodb.create_table(
  TableName='big_data_tweets',
  KeySchema=[
    { "AttributeName": "id", "KeyType": "HASH" }
  ],
  AttributeDefinitions=[
    { "AttributeName": "id", "AttributeType": "N" }
  ],
  # pricing determined by ProvisionedThroughput
  ProvisionedThroughput={
    'ReadCapacityUnits': 5,
    'WriteCapacityUnits': 5
  }
```

```
)
table.meta.client.get_waiter('table_exists').wait(TableName='big_data_tweets')
```

Processing and Storing Stream in DynamoDB

Before we store our stream in the DynamoDB table, we process it to extract the fields from the raw tweet which are useful to us. We also add additional features to our processed tweet. Refer to kinesis_to_dynamodb.py for the code.

Execute this code:

```
python ~/big_data_project/final_pipeline/kinesis_to_dynamodb.py
```

To check if data is being processed and stored in your DynamoDB table, go to your DynamoDB dashboard in AWS, click on Tables, then on big_data_tweets. Go to Items tab and you should see your data being indexed here.

Setting Up DynamoDB Stream

Go back to the Overview tab on the DynamoDB dashboard and click on Manage Stream. Select New Image and click on Enable to enable the DynamoDB stream.

Note down your DynamoDB table ARN and the DynamoDB stream ARN.

Similarly, go to Elasticsearch Service and note down the ARN and Endpoint and Kibana URLs.

Creating IAM Roles for Lambda

Go to IAM under Services and click on Roles and then Create role. Select AWS service role and Lambda, as service. Select AWSLambdaBasicExecutionRole as permission policy and click Next. Name your role, *dynamodb-to-es*.

Now we add policies to our role which will let the Lambda function access the DynamoDB table, its stream and the Elasticsearch Service.

Click Add inline policy and select Custom Policy. Create the following inline policies:

Access DynamoDB table *Access-to-the-DynamoDB*

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:Scan"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dynamodb:us-east-1:your-dynamodb-table-arn:table/elements"
            ]
        }
    ]
```

```
}
```

Access DynamoDB stream *Access-to-the-DynamoDB-stream*

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "dynamodb:DescribeStream",
                "dynamodb:GetRecords",
                "dynamodb:GetShardIterator",
                "dynamodb:ListStreams"
            ],
            "Effect": "Allow",
            "Resource": [

"arn:aws:dynamodb:us-east-1:your-dynamodb-stream-arn:table/elements/stream/
         2017-09-14T14:16:12.788"
            ]
        }
    ]
}
```

Access Elasticsearch *Lambda-to-Amazon-ES*

```
 {
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "es:ESHttpPost"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:es:us-east-1:your-es-arn:domain/elementsdomain/*"
        }
    ]
}
```

Note the /* after the ARN.

Creating Lambda Function

Go to Lambda under Services and click on Create function then Author from scratch. Name the function *dynamodb_to_es*, select runtime as Python 2.7, Choose an existing role as role, select the dynamodb_to_es IAM role as the existing role and create the function.

Scroll down and paste the code in lambda.py in the function code window labeled lambda_function. Ensure to change the ES_ENDPOINT in the code to your Elasticsearch Endpoint. It should be of the form

search-your-es-endpoint-xxxxxxxxxxxxxxxxxxxxxxxxxx.us-east-1.es.amazonaws.com/ without the https://.

Scroll down further and under Basic settings, set timeout to 5 minutes.

Scroll back up and add DynamoDB as a trigger from the list of triggers given in the left panel. Scroll down and select big_data_tweets as the DynamoDB table, check Enable trigger and click on Add.

Scroll back up and DynamoDB should be added on the left. The right panel lists the resources the function's role has access to. There should be three resources listed here, CloudWatch Logs, DynamoDB and Elasticsearch Service. Click on Save to deploy the Lambda function.

Creating Elasticsearch Mapping

A mapping in Elasticsearch is a schema that defines how the data is indexed for search and aggregation. Here we create custom analyzers which enable various text fields to be analyzed in multiple ways. Phone features and Hashtags have multi-fields, one each for aggregation and search. They support partial search and are not case sensitive. The tweet text also uses multi-fields, with aggregation supported for the complete tweet and a search which supports searching for words with the same root and is not case sensitive.

To create the mapping, click on the ElasticSearch Head extension icon in Google Chrome. Paste the Elasticsearch endpoint in the empty box at the top and click Connect. After a few seconds, you should be connected to the instance.

Go to the Any Request tab, expand Query, paste the Elasticsearch endpoint link in the top-most box, big_data_tweets in the box below it and select PUT from the drop-down menu. PUT is used to create an index named big_data_tweets in the mentioned Elasticsearch instance.

In the big white box below, paste the mapping given in es_mapping.json and click Request. If you get an output as below on the right, your index is created:

```
{
"acknowledged": true,
"shards_acknowledged": true
}
```

You can go back to Overview tab, click on Refresh on the upper right corner and you should see the index created.

Deploying Codes

Go back to your EC2 and finally deploy your codes to complete the setup using the following commands

```
python ~/big_data_project/final_pipeline/twitter_to_kinesis.py &
python ~/big_data_project/final_pipeline/kinesis_to_dynamodb.py &
```

Once the codes start running, go back to the DynamoDB table and check if the new items are getting indexed.

Then go to the ElasticSearch Head extension window and refresh the overview tab. You should see the data getting indexed in your index.