CAB 432 – CLOUD COMPUTING

Assignment 02 – Cloud Scaling Assignment

Greyden Scott

N9935924

Table of Contents

Introduction 3
Development 3
Stage 1: 3
Stage 2: 3
Stage 3: 3
Stage 4: 4
API and Packages 4
Twitter API 4
NLTK 4
Compromise 4
MongoDB 4
Google Charts 4
Use Cases 5
Technical Description 7
Client Side 7
Server Side 7
AWS Elastic Load Balance 8
AWS Auto-Scaling Group 8
AWS EC2 T2 Micro Server 8
MongoDB 8
Scaling and Performance 9
Scaling Up 10
Scaling Down 10
Testing and limitations 11
Test Plan 11
Compromises 11
Graph Intermittently Displays 11
Popular Nouns 11
Possible Extensions 12
Popular Nouns 12
D3JS Graphs 12
Appendix 13
A - User Guide 13
Web Application 13
B - Deployment Instructions 14
Pre-Requisites 14
Auto-Scaling Group 15
Load Balancer 15
Stream Server 16
Client Server 16
C -Test Cases 17

Introduction

TweetStorm aims to provide users with a visual representation of the sentimental value of tweets on a search subject. Utilising Twitter API for streaming tweets into a webserver, Compromise to clean the tweet body and NLTK to perform the sentiment analysis, the average sentiment of a chosen topic is then displayed on a graph which is generated on the client side.

When a user initiates a query with TweetStorm a request is made to a Server which begins a twitter stream based on the query provided, when a tweet is received by that server it is passed to a sentiment analysis server for processing, between these two servers is a load balancer to assist in handling the high volume of traffic that can be generated by popular trending topics and queries. The server processing the sentiment analysis, stores the tweet, the search query into a database. Meanwhile the client begins querying this same server for the average sentiment of the queried topic. The sentiment analysis server sends a request to a remotely hosted MongoDB to aggregate the average sentiment based on the query. Which is returned to the client and displayed as a graph that updates every 3 seconds with the current average for that query.

Development

TweetStorm was developed in four stages, in order to have tight grip on the scope of the application and to reduce the number of potential issues towards the end of development.

Stage 1:

Stage 1 involved setting up a rough prototype of the application, this version consisted of two servers, a stream server and a sentiment analysis server that also offered a client for controlling the stream and seeing the data, during this phase I had implemented google charts as the graphing tool but planned on moving to D3JS later in development for greater polish. It was during this phase of development that I realised persistence would be required for multiple requests being made at once and that separating the client from the sentiment analysis server would produce more requests to the server and be better for load generation when it came to auto scaling.

Stage 2:

Stage 2 of development involved splitting the sentiment analysis server and client from one another and providing better handling of requests at the stream from the client to ensure multiple users could make differing queries at once. This involved a rewrite of most of the back-end code and the structure in which requests were being made from one server to another. I also had to redesign the network architecture and begin preparing for deployment to the cloud.

Stage 3:

Stage 3 involved deploying to the cloud, testing and bug fixing. During this process it was discovered that the application was not producing enough CPU Utilisation load for scaling. As a result, additional features were added to the application, such as counting the number nouns in a tweet. This feature was not fully implemented due to time constraints. Unfortunately, I was still not able to generate enough CPU utilisation load.

Stage 4:

Stage 4 was involved monitoring the application hosted in the cloud to find the most optimal way for ensure auto-scaling. This involved setting up CloudWatch alarms and monitoring the application as it was being used.

API and Packages

Twitter API

https://developer.twitter.com/en/docs.html

Twitter is a social network platform that allows users to share short messages, images and videos on a feed. These messages can be categorised by hashtags and searches can be conducted on those hashtags or keywords. The Twitter API allows access to searching by keywords, hashtags or users and allows for results to be returned in a bulk search format or via a stream. Twitter storm uses this API to stream twitters directly into a server for sentiment analysis.

NLTK

https://www.nltk.org

NLTK is a platform for building programs to work with human language data. NLTK has a number of useful modules that allow for all types of text analysis. However, for the purposes of TweetStorm it was utilised to purely provide sentiment analysis, returning a value between -1 for negative and 1 for positive.

Compromise

http://compromise.cool/LZjbZ/

Compromise is another natural language platform, with extensive capabilities. Tweetstorm is utilising compromise to clean tweets of invalid and unusual characters before performing sentiment analysis. This increases the likelihood of running a proper sentiment result from NLTK.

MongoDB

https://www.mongodb.com & https://mlab.com

MongoDB is an open source cross platform document-orientated database program. Utilising JSON-like documents with schemata, twitter storm stores two tables into a MongoDB providing by mLabs. One table stores current trending topics for Australia. The other table stores the tweet body, sentiment and associated trending topic.

Google Charts

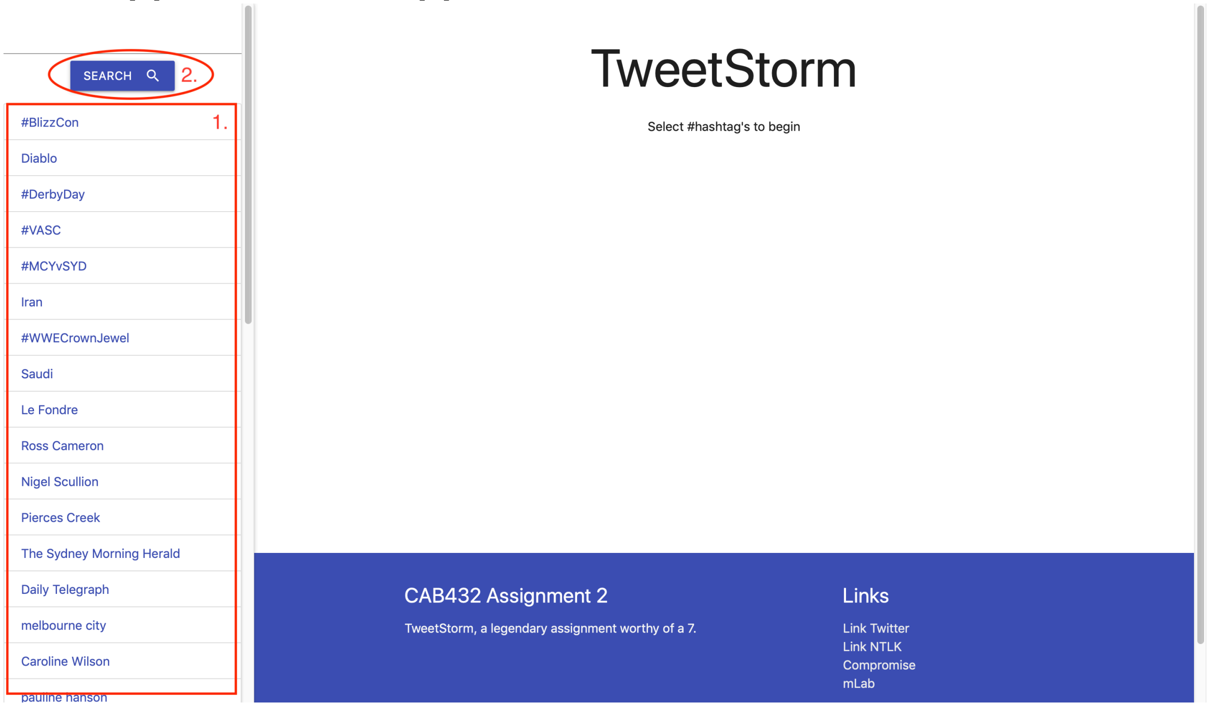
https://developers.google.com/chart/

Google Charts is a JavaScript API that allows structured data to be displayed in a visual diagram. The API is well documented and easily allows for the creation of interactive diagram charts.
The Google Charts library is included within the client-side index and is obtain via CDN. The diagrams are initialised and displayed when the web page has completed loading.

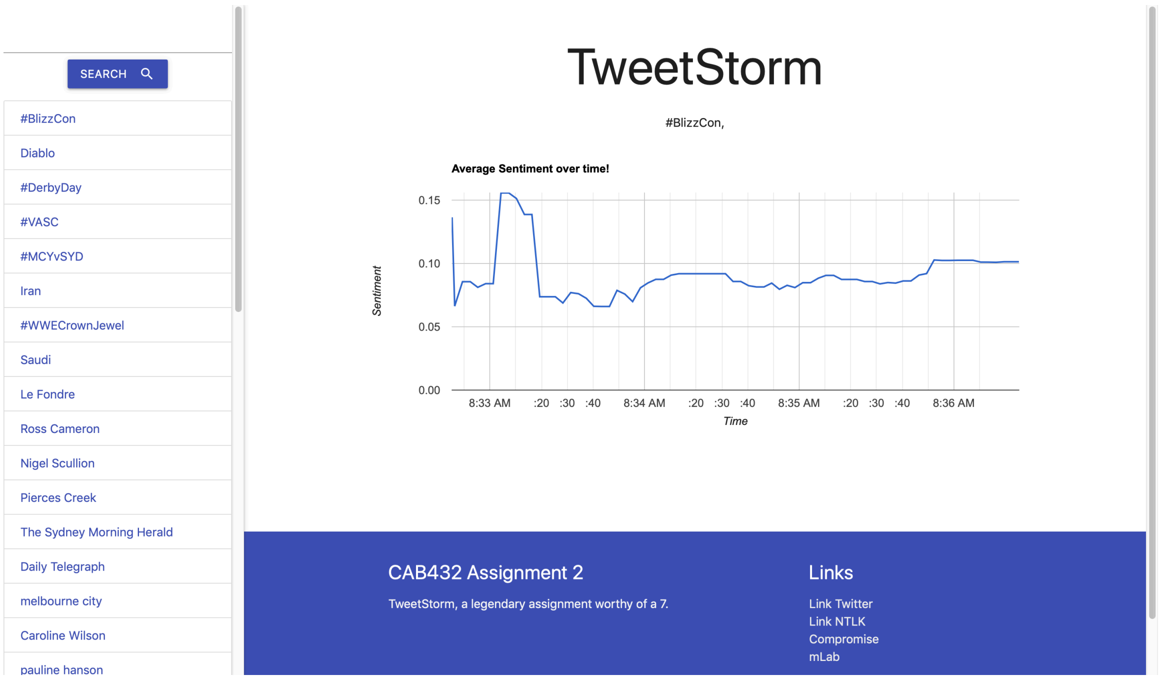
Use Cases

1. Selecting Trending Topics
   <u>As a user of TweetStorm, I want to see a list of trending topics and select one to see the average sentiment in real time.</u>

   The user navigates to the index page of TweetStorm, with the intent to see the average sentiment of a trending topic. The user selects a trending topic form the left menu[1] and selects Search [2] to initiate the stream and see the results.

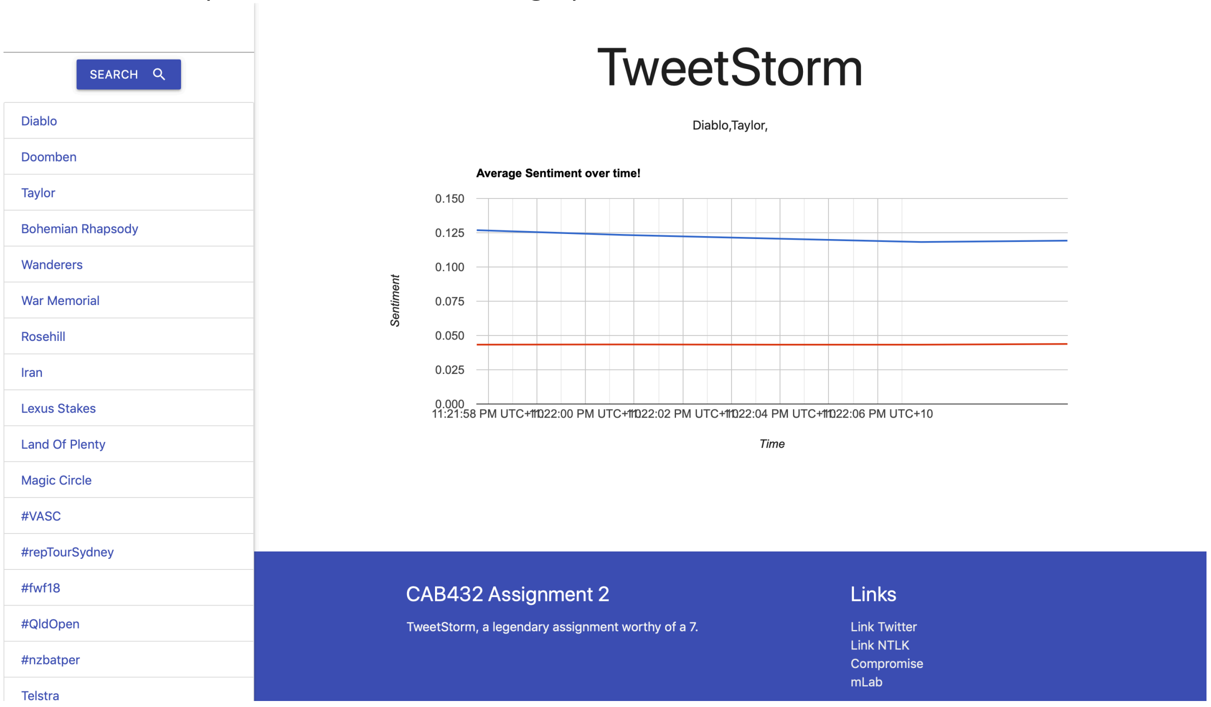


The page will reload and begin displaying the graph for select query which will update every 3seconds.



2. Comparing Multiple Trending Topics
   <u>As a user of TweetStorm, I want to select multiple trending topics, or search on multiple trending topics to compare the sentiment of those topic in real time.</u>

   If the user selects multiple queries the graph will show two lines for each query. And list the queries search above the graph.

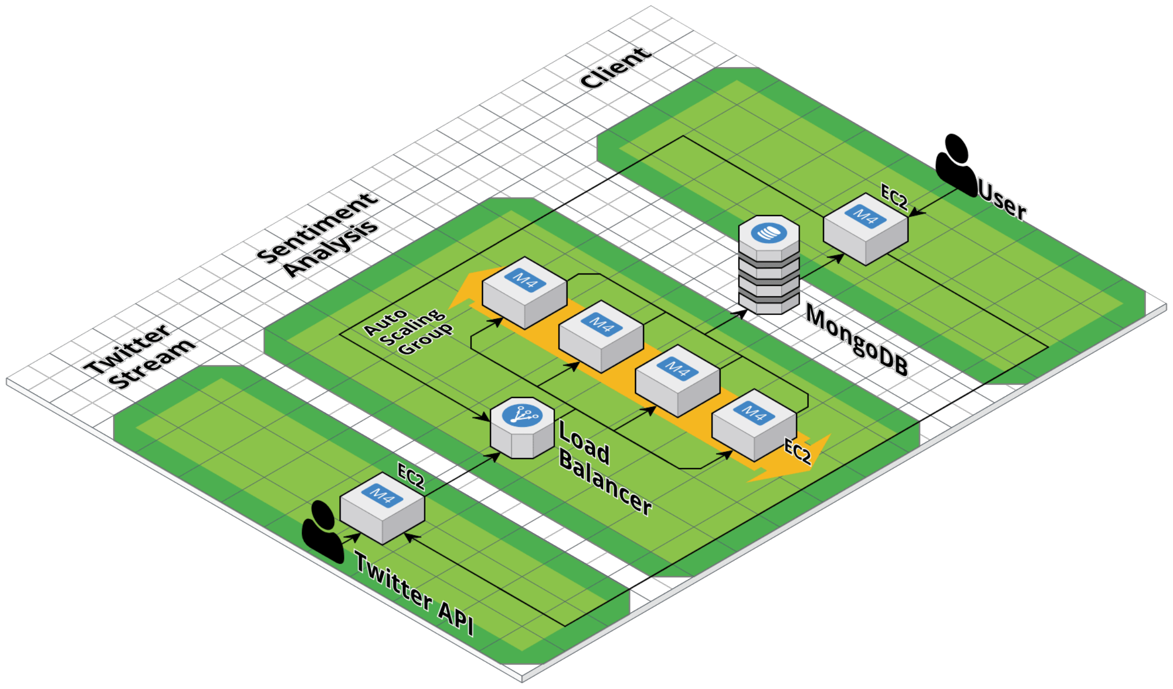


CAB432 Assignment 2

TweetStorm, a legendary assignment worthy of a 7.

Technical Description

TweetStorm utilises three servers running Node JS in order to achieve its task. The Client server, The Twitter Stream server and the Sentiment Analysis server. In addition to this MongoDB is utilised to provided persistence.



Client Side

That Client side uses Mongoose to access the MongoDB database to retrieve trending topics and to present to the user, the user can select a trending topic or perform their own search. In which case, the request is server to the Twitter Stream server to begin the stream with the request. Sentiment values are retrieved from the Auto Scaling group through the load balancer via an AJAX request on the client side. This request is performed every 3 seconds and produces load and traffic on the servers performing sentiment analysis.

Server Side

There are two servers performing tasks for this application, a Twitter Stream server and a Sentiment Analysis server. The Twitter Stream server handles the twitter stream. This server gets incoming tweets based on a query passed by the client server when the user initiates a request.

One of two events will occur in this instance, the server will start a new stream with the request if no stream presently exists. If a stream is already running, it will destroy the stream, append the request to what was previously being streamed and start a new stream. Whilst the stream is running it strips the body of the tweet as it comes in and sends it to the Analysis servers along with the terms the user initiated the stream with.

The Sentiment Analysis server main purpose is to perform sentiment analysis on the incoming tweets from the Twitter Stream Server. This is initiated when a stream begins and starts passing the body of tweets and searched terms to the Sentiment Analysis Server. It firms performs a check to see if the search terms are stored in the MongoDB, if the term does not exist it will add it to the MongoDB. It will then clean the text using Compromise to ensure sentiment analysis can be performed. However, prior to this sentiment analysis it searches the tweet for the tags stored in the database and categorises the tweet that that tag. Then it performs the sentiment analysis with the body of the tweet. Once all tasks are complete it then pushes an entry into the MongoDB that contains the tag, tweet body and sentiment analysis score.

In addition to the above the Sentiment Analysis server also handles queries from the client. When the client makes a request, it will request the average sentimental value of the requested query from the Sentiment Analysis server via a post request. This is returned in JSON format for the client to using Google Charts.

AWS Elastic Load Balance

An AWS Elastic Load Balancer automatically distribute incoming application traffic across multiple targets. Tweetstorm utilises a load balancer in front of the Auto Scaling Group, this handles the large amounts of traffic coming in from the Twitter Stream to the Sentiment Analysis Servers and balances the traffic.

AWS Auto-Scaling Group

An AWS Auto-Scaling Group allows EC2 instances to be created dynamically based on defined conditions, this was utilised to balance the required number of instances in order to provide an application that is consistent despite the volume of task being completed.

AWS EC2 T2 Micro Server

All servers are running on an AWS T2 Micro Server in the Asia Pacific (Sydney) region. Each server has a custom bash script that is run on reboot. This screen removes all docker images and containers from the server and downloads the latest version of the docker image from docker hub and runs the image. This was used throughout development, allowing me to make changes locally on my machine, push the new docker image to docker hub and reboot the AWS servers to have them running the up-to-date code.

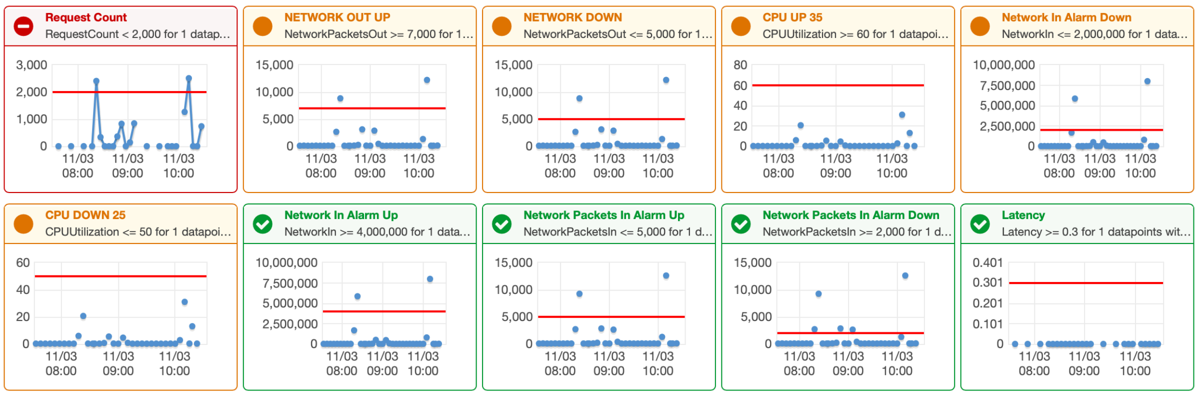
MongoDB

A remotely hosted MongodDB through mLabs which the Sentiment Analysis server writes and retrieves records from, further to this the client also retrieves records from the MongoDB, however this only occurs when the client loads the webpage.

Scaling and Performance

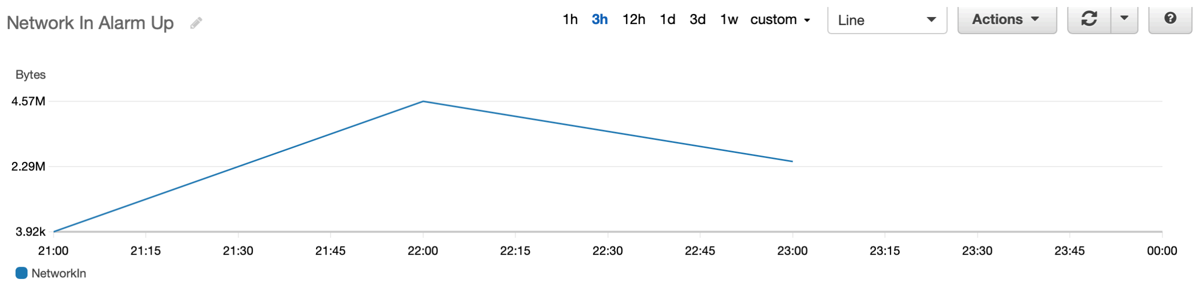
Auto scaling has been configured on the Network Packets going In to the Auto Scaling group by the loader balancer. This metric is dependent on the volume of tweets going form the Stream sever to the Auto Balancer. This was chosen over Network Packets going out from the Auto Scaling group, as the packets going out would be for the most part slightly less, and the packets going in was safer value to predict the volume of information being processed.

Thorough investigation went into the metrics used to scale the application. CPU Utilisation never hit peaks that warranted extra server capacity as it managed to handle the processing of requests as they came in. However, the server was unable to handle all the incoming data from the Twitter Stream server whilst also processing data to the database and to the client during high volume streams. Ideally it would be much better to scale on latency. However, this only occurs when the database contains considerable volume and the processes for retrieving data took significantly longer to perform. This analysis was conducted by setting up alarms in AWS cloud watch platform and monitoring the server behaviour and configuring scaling policies based on what was being observed.


Request Count
RequestCount < 2,000 for 1 datap...
NETWORK OUT UP
NetworkPacketsOut >= 7,000 for 1...
NETWORK DOWN
NetworkPacketsOut <= 5,000 for 1...
CPU UP 35
CPUUtilization >= 60 for 1 datapoi...
Network In Alarm Down
NetworkIn <= 2,000,000 for 1 data...
CPU DOWN 25
CPUUtilization <= 50 for 1 datapoi...
Network In Alarm Up
NetworkIn >= 4,000,000 for 1 data...
Network Packets In Alarm Up
NetworkPacketsIn <= 5,000 for 1 d...
Network Packets In Alarm Down
NetworkPacketsIn >= 2,000 for 1 d...
Latency
Latency >= 0.3 for 1 datapoints wit...
11/03 08:00
11/03 09:00
11/03 10:00

CloudWatch Alarms

Time of day was a factor, running the stream during America's peak usage times caused significantly higher volume of traffic from the Twitter Stream server to the Sentiment Analysis server, during which latency spiked significantly higher, along with CPU Utilisation and Incoming and Outgoing traffic from the Twitter Stream server. However, as Americas peak usage times did not coincide with the time of day I was developing and testing TweetStorm this was not detected until late in development. By using more than one scaling policy I would handle and treat Americas peak usage times differently that what has been configured. With CPU Utilisation and latency-based scaling policy TweetStorm would be better prepared for handling Twitter traffic during Americas peak usage times.

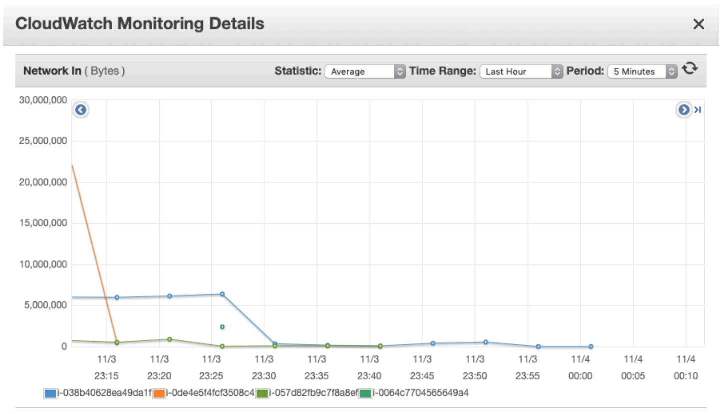


Network In Alarm

For the purpose of this assignment, scaling policy based on incoming network traffic was the best way to demonstrate scaling up and down, as it could be controlled by the interactions with the client. As trending topics with a high volume could be queried, resulting in a spike of traffic occurring which initiates scaling.

Scaling Up

To increase the auto scaling group size, a policy was created based on the Network Traffic coming in. If the network traffic incoming was greater than four million bytes for a period greater than 60 seconds it was to add 1 instance to the auto scaling group.

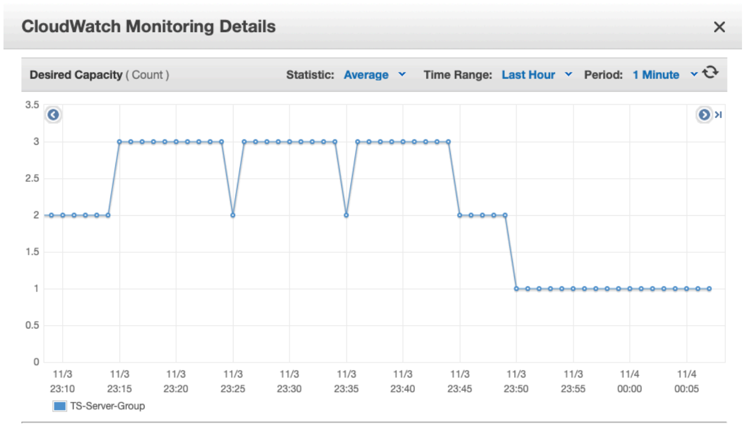


Statistic: Average Time Range: Last Hour Period: 5 Minutes

Sentiment Analysis Server Network In AVG Graph

Scaling Down

To decrease the auto scaling group size, a policy was created based on the Network Traffic coming in. If the network traffic incoming was less than two million bytes for a period greater than 60 seconds it was to remove 1 instance from the auto scaling group.



Statistic: Average Time Range: Last Hour Period: 1 Minute

Auto Scaling Desired Capacity Count Graph

Testing and limitations

When the TweetStorm was first deployed, it was expected that the application would scale based on CPU Utilisation, when this was not the case, considerably more time was invested into analysing the infrastructure and server to determine the best method for scaling. As a result, minimal bug fixing was conducted during development. Due to this, extra features developed were not included in the final version and the front end is not as polished as it could have been. However, due to the development process an end product was still able to be produced that demonstrates the application and performs scaling based on network traffic volume.

Test Plan

Purpose	Expected Outcome	Result	Screenshot/s (Appendix)
View & Use TweetStorm	Trending Topics Display	PASS	01: A
	Search Button sends query	PASS	02: A - B
	Graph Displays	PASS	03: A
	Graph Updates based on value	PASS	03: B
	Graph displays: Legend	FAIL	04: A
	Graph displays: Labels	FAIL	04: A
	Graph displays time	PASS	04: B
	Initiate Second Search	PASS	05: A
	Initiate Search with more than one topic	PASS	05: A
Load Balancer	LB Health Check	PASS	06: A
Auto Scaling	Auto Scale Up	PASS	07: A
	Auto Scale Down	PASS	07: A

Compromises

Graph Intermittently Displays

On some occasions the graph does not display when a query is performed. This is due the ajax query on the client side making a request to the server for an average sentiment before any records are stored in the database. It would be possible to avoid this by creating a small delay or better yet handle the data a bit differently, as in if nothing returns from the server to display a different message. It was my intention to do exactly that. However due to time constraints the work was not performed.

Popular Nouns

It was my intention to display a list of nouns, this was added to generate extra load and add more use cases to the site. The functionality is coded into the Sentiment Analysis server; however, no query is being made from the client to display the information. This was dropped at last minute due to time constraints.

Possible Extensions

Popular Nouns

With more time permitting the application could be extended to show the popular nouns for a particular topic with sentiment analysis run on the words to show their disposition. A portion of the code already exists, it would just require a query by the client and formatting.

D3JS Graphs

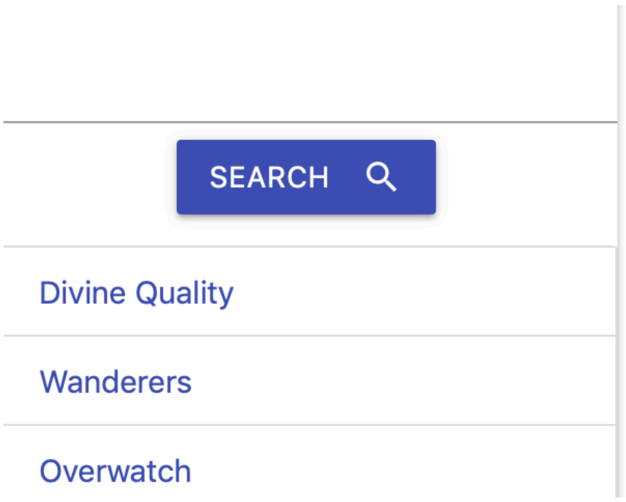
The current chart API is Google Charts. It does not handle live data well and often renders the chart incorrectly as a result. Substituting out Google Charts for something more expansive as D3JS would greatly improve the chart quality as the variety of charting options is far greater.

Appendix

A - User Guide

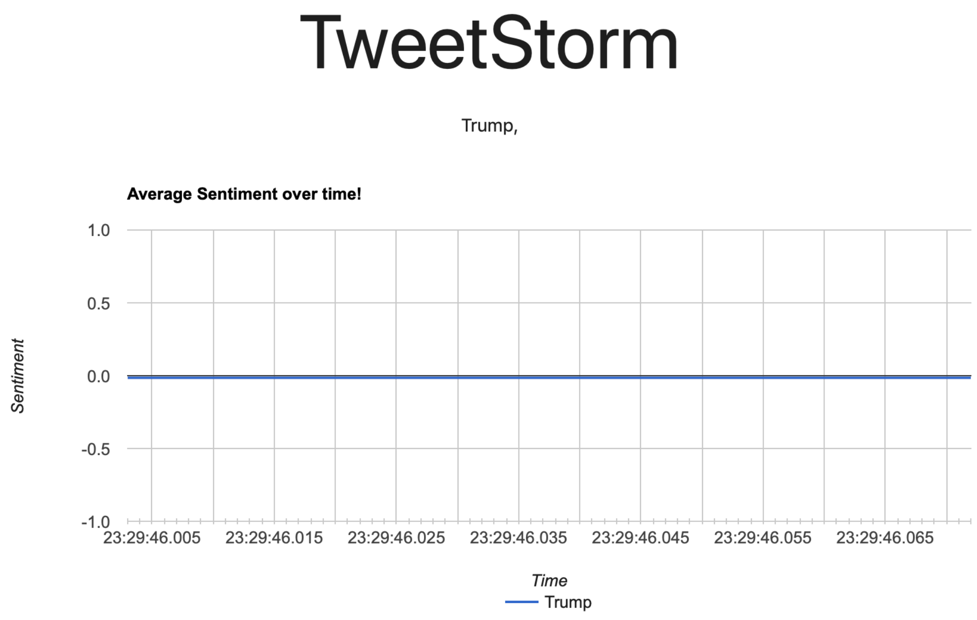
Web Application

1.0



The User Can type in a search query or select a trending topic from the list below the search button. Once the user has selected the query/queries they press search to begin seeing the results.

2.0



The results are then displayed in a graph on the main page and update automatically. The user can view the results change in real time or initiate another search query.

B - Deployment Instructions

Pre-Requisites

Security Groups

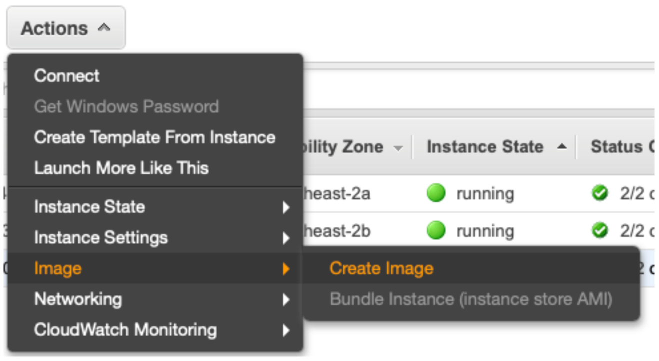
Three Security Groups are required, this can be configured when building EC2 instances of prior using AWS management console.

1. TS-Client-SG
   a. Inbound & Outbound Port 22 SSH - TCP Protocol – Any Source
   b. Inbound & Outbound Port 3000 - TCP Rule – Any Source
   c. Outbound - Open All Traffic
2. TS-Stream-SG
   a. Inbound & Outbound Port 22 SSH - TCP Protocol – Any Source
   b. Inbound Port 3002 - TCP Rule – Any Source
   c. Outbound - Open All Traffic
3. TS-Server-SG
   a. Inbound & Outbound Port 22 SSH - TCP Protocol – Any Source
   b. Inbound Port 3002 - TCP Rule – Any Source
   c. Outbound - Open All Traffic

AMI

The auto-scaling group will launch of an AMI this should be configured before the rest of the configuration is performed.

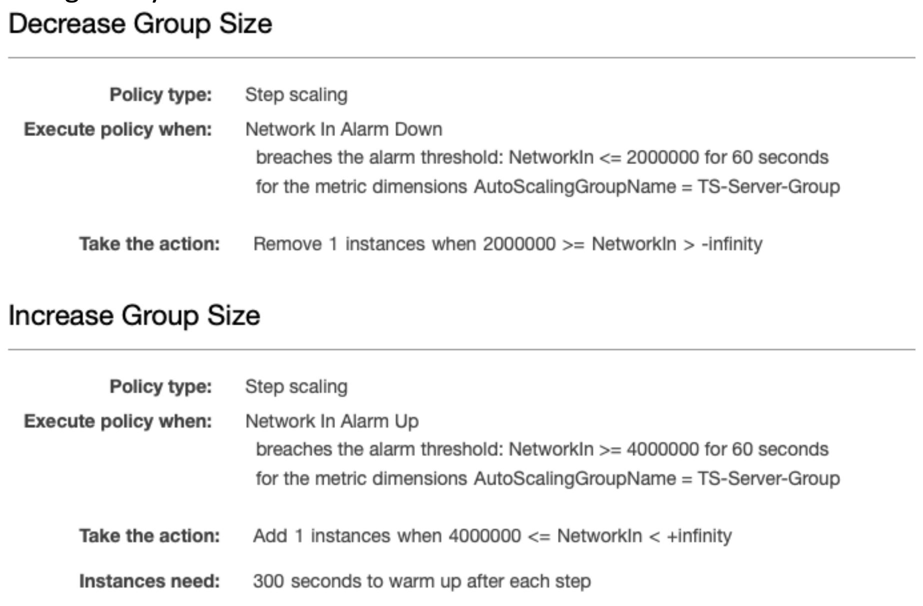
1. Create an EC2 instance, apply the TS-Server-SG security group
2. SSH in using appropriate credentials and run the following commands:
   a. sudo apt-get update
   b. sudo apt-get install docker.io
   c. nano startup.sh
      i. <u>Inside the bash file enter the following</u>
      sudo docker system prune -a -f
      sudo docker rm $(docker ps -aq)
      sudo docker pull greyden/tsserver:latest
      sudo docker run –name client -p 3001:3001 greyden/tsserver &
   d. crontab -e
      i. inside the cronjob file enter the following
      ii. @reboot /home/ubuntu/startup.sh
   e. sudo nano reboot – once reboot is complete reconnect and ensure docker container is running and that all scripts worked as intended
3. From AWS management console creating an image of the running EC2 instance



4. The instance can now be terminated.

Auto-Scaling Group

Create an Auto Scaling Group with the following configuration
Instances: Min 1 Max 4
Image: (Image created as per Prerequisites – AMI)
Security Group: TS-ServerSG
Scaling Policy:



Decrease Group Size

Policy type:	Step scaling
Execute policy when:	Network In Alarm Down breaches the alarm threshold: NetworkIn <= 2000000 for 60 seconds for the metric dimensions AutoScalingGroupName = TS-Server-Group
Take the action:	Remove 1 instances when 2000000 >= NetworkIn > -infinity

Increase Group Size

Policy type:	Step scaling
Execute policy when:	Network In Alarm Up breaches the alarm threshold: NetworkIn >= 4000000 for 60 seconds for the metric dimensions AutoScalingGroupName = TS-Server-Group
Take the action:	Add 1 instances when 4000000 <= NetworkIn < +infinity
Instances need:	300 seconds to warm up after each step

Health Check Grace Period: 30
Default Cooldown: 60
Availability Zones: ap-southeast-2b, ap-southeast-2a, ap-southeast-2c

Load Balancer

Create A Classic Loader Balancer with the following config
Port Configurations: 80 (HTTP) forwarding 80 (HTTP)
3001 (TCP) forward 3001 (TCP)
Security: TS-Server-SG
Idle timeout: 60 seconds
Health Check:

1. Ping Target: HTTP:3001/
2. Timeout: 2 seconds
3. Interval: 10 seconds
4. Unhealthy Threshold: 2
5. Healthy Threshold: 5

Once this is complete modify the Auto Scaling Group to use the Created Load Balancer

Stream Server

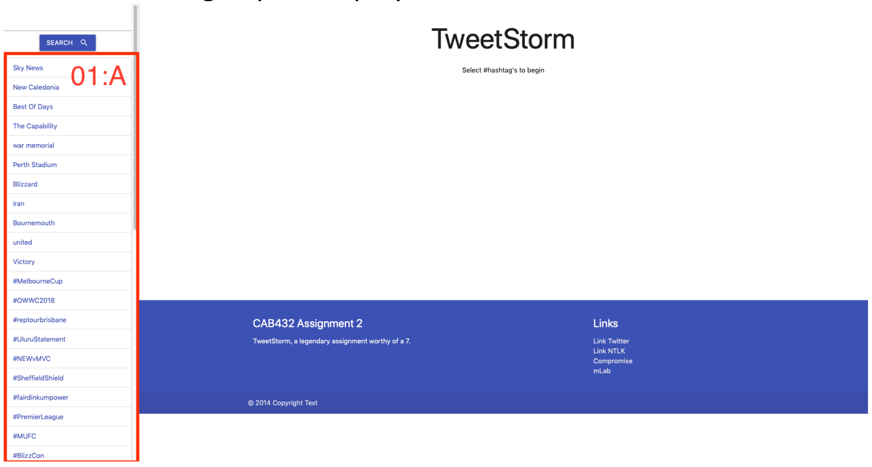
1. Create an EC2 instance, apply the TS-Stream-SG security group
2. SSH in using appropriate credentials and run the following commands:
    a. sudo apt-get update
    b. sudo apt-get install docker.io
    c. nano startup.sh
        i. <u>Inside the bash file enter the following</u>
        sudo docker system prune -a -f
        sudo docker rm $(docker ps -aq)
        sudo docker pull greyden/tsstream:latest
        sudo docker run –name client -p 3001:3001 greyden/tsstream &
    d. crontab -e
        i. inside the cronjob file enter the following
        ii. @reboot /home/ubuntu/startup.sh
    e. sudo nano reboot – once reboot is complete reconnect and ensure docker container is running and that all scripts worked as intended

Client Server

1. Create an EC2 instance, apply the TS-Client-SG security group
2. SSH in using appropriate credentials and run the following commands:
    a. sudo apt-get update
    b. sudo apt-get install docker.io
    c. nano startup.sh
        i. <u>Inside the bash file enter the following</u>
        sudo docker system prune -a -f
        sudo docker rm $(docker ps -aq)
        sudo docker pull greyden/tsclient:latest
        sudo docker run –name client -p 3000:3000 greyden/tsclient &
    d. crontab -e
        i. inside the cronjob file enter the following
        ii. @reboot /home/ubuntu/startup.sh
    e. sudo nano reboot – once reboot is complete reconnect and ensure docker container is running and that all scripts worked as intended

C -Test Cases

01: A – Trending Topics Display


SEARCH

01:A

Sky News
New Caledonia
Best Of Days
The Capability
war memorial
Perth Stadium
Blizzard
iran
Bournemouth
united
Victory
#MelbourneCup
#OWWC2018
#reptourbrisbane
#UluruStatement
#NEWvMVC
#SheffieldShield
#fairdinkumpower
#PremierLeague
#MUFC
#BlizzCon

TweetStorm

Select #hashtag's to begin

CAB432 Assignment 2

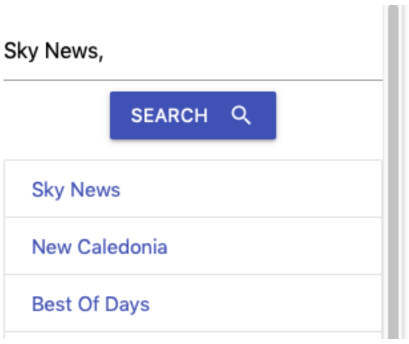
TweetStorm, a legendary assignment worthy of a 7.

Links

Link Twitter
Link NTLK
Compromise
mLab

© 2014 Copyright Text

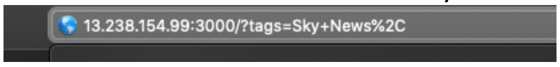
02: A – Search Button Sends Query


Sky News,

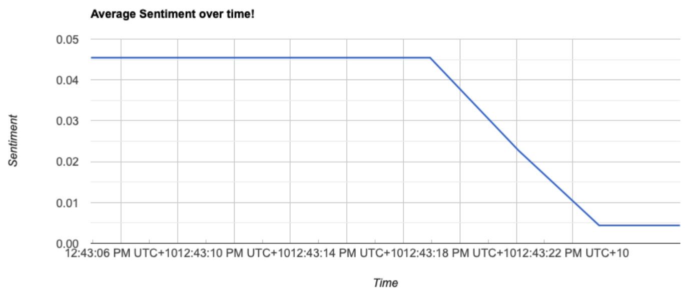
SEARCH

Sky News
New Caledonia
Best Of Days

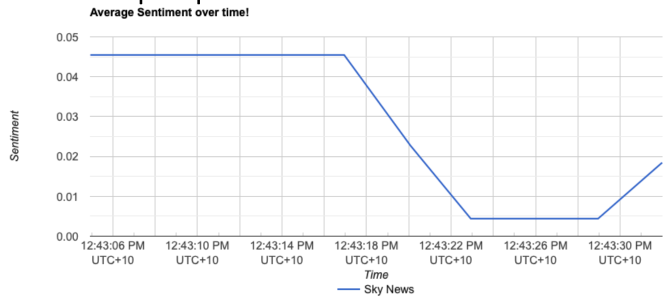
02: B – Search Button Sends Query


13.238.154.99:3000/?tags=Sky+News%2C

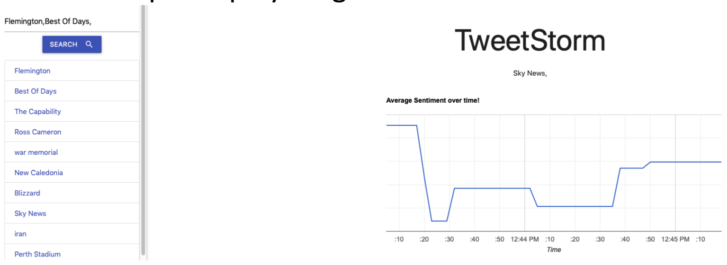
03: A – Graph Displays



03: B – Graph updates based on value



04: A – Graph Displays Legend & Labels

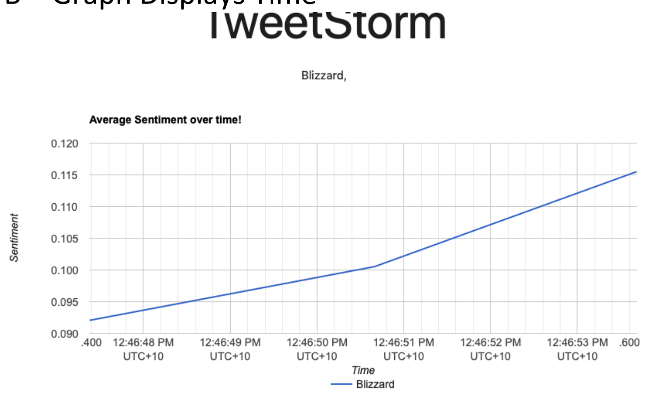


Flemington,Best Of Days,

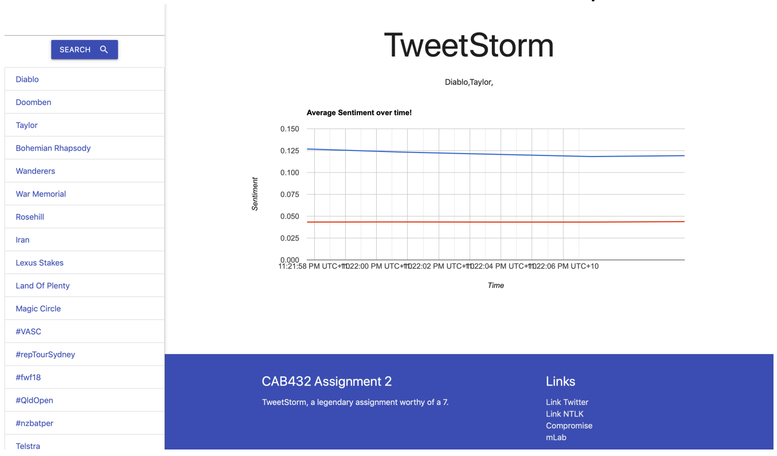
SEARCH

- Flemington
- Best Of Days
- The Capability
- Ross Cameron
- war memorial
- New Caledonia
- Blizzard
- Sky News
- iran
- Perth Stadium

04: B – Graph Displays Time



05: A – Initiate Search with more than one topic



SEARCH

- Diablo
- Doomben
- Taylor
- Bohemian Rhapsody
- Wanderers
- War Memorial
- Rosehill
- Iran
- Lexus Stakes
- Land Of Plenty
- Magic Circle
- #VASC
- #repTourSydney
- #fwf18
- #QldOpen
- #nzbatper
- Telstra

TweetStorm
Diablo,Taylor,
Average Sentiment over time!
Sentiment
0.150
0.125
0.100
0.075
0.050
0.025
0.000
11:21:58 PM UTC+1022:00 PM UTC+1022:02 PM UTC+1022:04 PM UTC+1022:06 PM UTC+10
Time

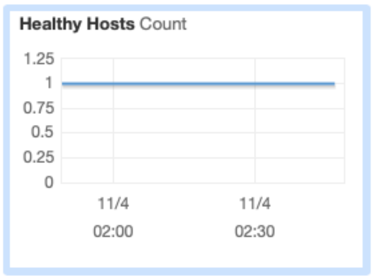
CAB432 Assignment 2

TweetStorm, a legendary assignment worthy of a 7.

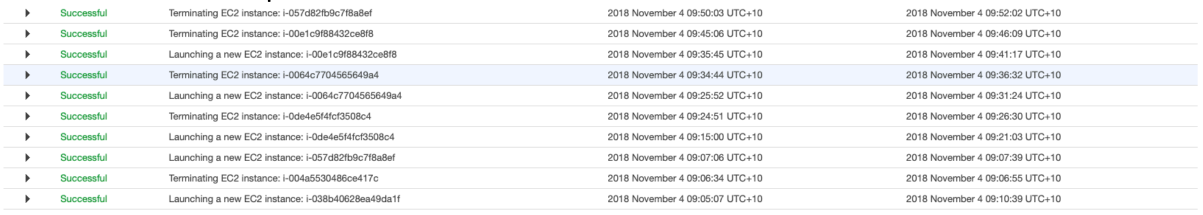
Links

- Link Twitter
- Link NTLK
- Compromise
- mLab

06: A – LB Health Check



07: A – Auto Scale Up – Auto Scale Down



▸	Successful	Terminating EC2 instance: i-057d82fb9c7f8a8ef	2018 November 4 09:50:03 UTC+10	2018 November 4 09:52:02 UTC+10
▸	Successful	Terminating EC2 instance: i-00e1c9f88432ce8f8	2018 November 4 09:45:06 UTC+10	2018 November 4 09:46:09 UTC+10
▸	Successful	Launching a new EC2 instance: i-00e1c9f88432ce8f8	2018 November 4 09:35:45 UTC+10	2018 November 4 09:41:17 UTC+10
▸	Successful	Terminating EC2 instance: i-0064c7704565649a4	2018 November 4 09:34:44 UTC+10	2018 November 4 09:36:32 UTC+10
▸	Successful	Launching a new EC2 instance: i-0064c7704565649a4	2018 November 4 09:25:52 UTC+10	2018 November 4 09:31:24 UTC+10
▸	Successful	Terminating EC2 instance: i-0de4e5f4fcf3508c4	2018 November 4 09:24:51 UTC+10	2018 November 4 09:26:30 UTC+10
▸	Successful	Launching a new EC2 instance: i-0de4e5f4fcf3508c4	2018 November 4 09:15:00 UTC+10	2018 November 4 09:21:03 UTC+10
▸	Successful	Launching a new EC2 instance: i-057d82fb9c7f8a8ef	2018 November 4 09:07:06 UTC+10	2018 November 4 09:07:39 UTC+10
▸	Successful	Terminating EC2 instance: i-004a5530486ce417c	2018 November 4 09:06:34 UTC+10	2018 November 4 09:06:55 UTC+10
▸	Successful	Launching a new EC2 instance: i-038b40628ea49da1f	2018 November 4 09:05:07 UTC+10	2018 November 4 09:10:39 UTC+10