2017

Twitter Streams Sentiment Analysis

A sentiment analysis web application of Real-Time Twitter Stream.

CAB432 Assignment 2 Report
YeeChen Fong - n9602976
PengYuan Ge - n8936854

Table of Content

Introduction

Twitter Streams Sentiment Analysis aims to provide a resource for users to perform sentiment analysis on Tweets coming in from all around the world in real-time. The service computationally identifying and categorizing opinions expressed in a tweets, to determine whether the tweet writer's attitude towards a particular topic is positive, negative, or neutral. Moreover, the service provides the visualisation of the sentiment of tweets based on a live topic filter which the user can modify it by entering or removing "keywords". These visualisation comes in the form of a Sentiment Activity (Real-time line chart) and a Sentiment Topic (Topic bar chart).

The architecture of the the application is divided into 3 sections, *Tweet Stream*, *Sentiment Analysis Processing*, and *Application Client*. The *Tweet Stream* section consist of a M4 EC2 instance which retrieves Tweets from a Tweet Stream. The *Sentiment Analysis Processing* section consist of an Elastic Load Balancer which manages an Auto-Scaling group consisting of T2 EC2 instance/instances. The Load Balancer receives tweets from the M4 instance and passes the request to the auto-scaling group instance with the lowest load, the instance processes the tweet and stores it in a DynamoDB. The *Application Client* section consist of a T2 EC2 instance which retrieves the tweet object from the DynamoDB and displays it to the front-end.

Services used by the to develop this application includes Twit (a Twitter API client for Node.js), Sentiment (a Sentiment module for Node.js), Socket-IO (a JavaScript socket library for realtime bi-directional communication between web clients and servers), AWS-SDK Js (a JavaScript library that enables developers to use AWS services in node.js), Chart.js (a JavaScript charting tool), Request Js (a HTTP/HTTPS request library for Node.js) and the usual HTML5, CSS, Bootstrap, Javascript, JQuery, EJS, Express and Node.js.

The development process is divided into 3 phases. It begins with developing and testing of the Stream server, Sentiment Analysis server, and the Application client. The next phase is deploying the packages into 3 AWS instances followed by creating the AMIs respectively. The last phase is the create a Launch Configuration using the Sentiment Analysis server AMI, followed by creating an Auto-Scaling group, Load Balancer, and attach the Load Balancer to the Auto-Scaling group.

Use Cases & Dependent Services

Use Case 1

As an **user**, I want the application to **filter tweets based on topics** so that **I can see the sentiment of tweets on topics that I am interested**.

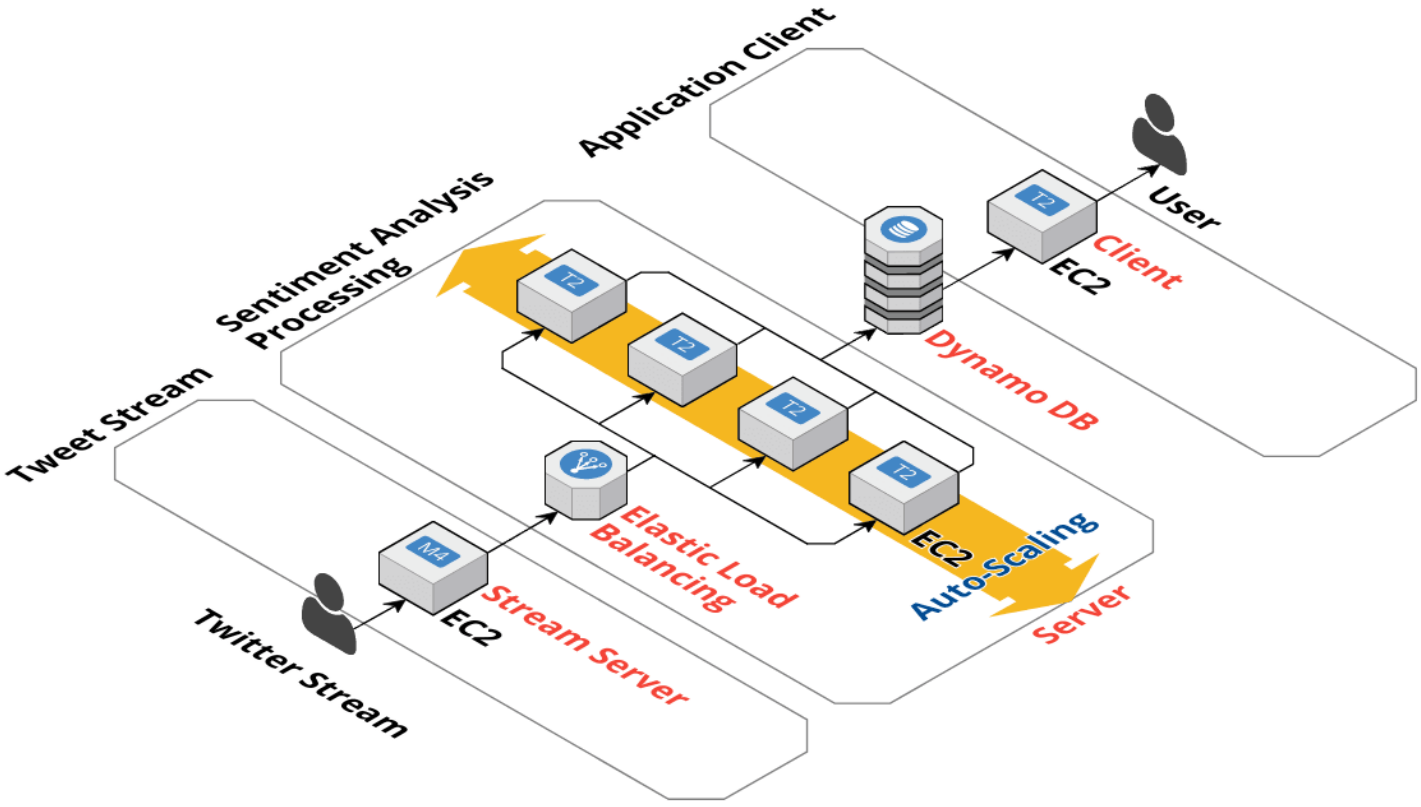
Use Case 2

As a **user**, I want the application to **visualise the the overall sentiment, the number of positive, negative and neutral tweets, the number of tweets related to the keywords in real-time** so that **I can envision the public's general attitude towards a particular topic when tweeting**.

Technical Description

Architecture

The architecture of the the application is divided into 3 sections, ***Tweet Stream***, ***Sentiment Analysis Processing***, and ***Application Client***.



***Tweet Stream* section (only server-side)**

The *Tweet Stream* section consist of a M4 EC2 instance which retrieves Tweets from a Tweet Stream from the public Twitter Endpoint. It sends a POST request to the elastic load balancer domain name on port 3000 for each tweet coming in.

***Sentiment Analysis Processing* section (only server-side)**

The *Sentiment Analysis Processing* section consist of an Elastic Load Balancer which manages an Auto-Scaling group consisting of T2 EC2 instance/instances. Every incoming POST request from the Stream Server in the *Tweet Stream* section is handled by the Elastic Load Balancer. The Load Balancer check for any server in the Auto-Scaling group which has the lowest load and passes the POST request to it. The server accepts the POST request with the tweet in JSON and performs the Sentiment Analysis on the text of the tweet. A sentiment rating for the tweet is returned from the Sentiment library which is then processed to categorise the "emotion" of the tweet in either one of the three category, "Positive", "Neutral", and "Negative". The sentiment ratings and the "emotion" is then combined with the tweet to create a *Tweet Object* which is then passes into a query to be stored in a DynamoDB.

***Application Client* section (client and server-side)**

The *Application Client* section consist of a T2 EC2 instance which receives "keywords" from the user and retrieves the recently stored *Tweet Object* related to the keyword from the DynamoDB and displays it to the front-end every 1 second. The "emotion" of the *Tweet Object* is counted and added to a real-time line chart and a topic bar chart. The real-time line chart refreshes every 5 seconds, show the number of "Positive", "Neutral", and "Negative" and total tweets coming in, whilst the topic bar chart shows the number of "Positive", "Neutral", and "Negative" tweets for each topic/keyword the tweets are related to. An overall counter is also displayed which shows the overall sentiment of all the tweets coming in.

Technology

Server-side

- **Node JS [1]:**
  Open-source, cross-platform, asynchronous javascript runtime framework for building event driven network applications. The node.js server is used to handle serving files as well as send requests and receive response from APIs, and Databases. Used in the *Tweet Stream*, *Sentiment Analysis Processing*, and *Application Client* section.
- **Express JS [2]:**
  A simple web framework that sets the routes and views easily, allowing the client and server-side to be integrated easily, which allows for rapid development. Used in the *Sentiment Analysis Processing*, and *Application Client* section.

- **Request JS [3]:**
  A simple HTTP/HTTPS request library that makes HTTP/HTTPS calls the simplest way possible. GET and POST request are made and data is returned back in JSON format which can be easily converted to Javascript objects. Used in the *Tweet Stream* section.

- **Twit [4]:**
  A Twitter API client for Node.js, supports both the REST and Streaming API. In this application it is used to receive tweet streams. Used in the *Tweet Stream* section.

- **Sentiment [5]:**
  A Sentiment module for Node.js that uses the AFINN-165 wordlist and Emoji Sentiment Ranking to perform sentiment analysis on arbitrary blocks of input text. Returns sentiment rating of tweets. Used in the *Sentiment Analysis Processing* section.

- **Socket-IO [6]:**
  A JavaScript library for realtime web applications. It enables realtime, bi-directional communication between web clients and servers via sockets. Used in the *Application Client* section.

- **AWS-SDK Js [7]:**
  A JavaScript library that enables developers to use AWS services in node.js. It is

used to send and retrieve *Tweet objects* to and from the DynamoDB. Used in the *Sentiment Analysis Processing* section.

- **AWS Elastic Load Balancer:**
  An AWS service function that automatically distributes incoming application traffic across multiple targets, such as Amazon EC2 instances, containers, and IP addresses. It is able to manage application traffic in a single Availability Zone or across multiple Availability Zones. Used for the auto-scaling group in the *Sentiment Analysis Processing* section.

- **AWS Auto-Scaling Group:**
  An AWS service function that allows dynamically scaled EC2 capacity up or down automatically according to conditions define. Auto Scaling allows automatic increase in the number of Amazon EC2 instances during demand spikes to maintain performance and decrease capacity during lulls to reduce costs. Used in the *Sentiment Analysis Processing* section.

- **AWS DynamoDB:**
  A NoSQL database service that stores and retrieves any amount of data, and serve any level of request traffic. Unfortunately, the scaling feature of the database is not available for AWS student accounts.

Client-side

- **HTML5 & CSS3:**
  Markup and stylesheet language used to build web pages. HTML provides the structure of the page and CSS styles the pages, setting the colors, layout and fonts.

- **JavaScript:**
  This client-side of Javascript enables interactive features and dynamic functions. It is used to count the emotions of tweets and displays the overall sentiment.

- **Bootstrap:**
  Most popular HTML, CSS and JavaScript framework used to develop simple and consistent UI for web page. It contains the HTML and CSS based design template for typography, forms, buttons, navigation and other interface components. The web pages in this services are styled using Bootstrap components.

- **JQuery:**
  A Javascript library used for animating effect, event listening and event handling. It is used to handle the real-time and tweet topics visualisation.

- **EJS, Embedded JavaScript templates [8]:**
  A template of Express framework. Javascript can be written directly with the HTML markup in EJS template. It also allows Javascript objects to be rendered from the server side (Node.js) to the client side(EJS) with ease.

- **Chart.js [9]:**
  A simple JavaScript charting tool. Used to display the real-time tweet counts and topic counts.

Issues & Difficulties

1. **One tweet stream per IP address (Twitter Rate Limits)**
   This prevents user from changing the track words filter by adding new tweet streams if an existing tweet stream exist.

2. **Tweet stream word tracklist cannot be modified during connection (Twitter Rate Limits)**
   This prevents user from modifying the word tracklist without disconnecting the existing stream and connecting a new tweet stream.

3. **Excessive connection attempts causes tweet stream to be throttled (Twitter Rate Limits)**
   This prevents user from changing the track words filter on the stream as it is not possible to change the filter directly on the Twitter public endpoints without doing a disconnect and reconnect. The problem is, doing a disconnect and reconnect too many times causes throttling and even IP address ban for a period of time. Thus Twitter stream filter endpoints has to be pre-defined “A” - “Z” beforehand.

4. **Predefined Tweet Stream filter**
   Keywords “A” - “Z” are overly represented in the tweet stream. Any other keywords has significantly less occurrence to be displayed in the front-end.

5. **Generating Load**
   The predefined filter makes it impossible for the application to generate/reduce load via incrementing/decrementing filter keywords. Thus auto-scaling could not be done on the client.

6. **Non-scalable DynamoDB**
   The AWS student account do not allow the creation of IAM (Identity and Access Management) roles. The IAM entity is needed to allow the DynamoDB to scale to stores and retrieves any amount of data, and serving any level of request traffic. The DynamoDB’s write capacity units and read capacity units has to be manually set (in this application its 25 write capacity units and 5 read capacity units).

Overall Implementation

Tweet Stream **section (only server-side)**

An app.js file create a tweet stream connection to Twitter using a pre-defined track of characters “A” - “Z”. Every single Incoming tweets passing the stream filter is then POST requested to the domain name of the Elastic Load Balancer. The server files are stored in an AMI (Amazon Machine Image) which can deployed at any time.

Sentiment Analysis Processing **section (only server-side)**

For the Sentiment Analysis server, apart from the www.js and app.js file which creates the node server, an index.js file is created which allows the incoming POST request with the tweet to be processed by the sentiment library and to be stored in the DynamoDB. The Sentiment Analysis server files are stored in an AMI.

The AMI is used to create the Launch Configuration, followed by an Auto-Scaling group with 2 policies to be trigger after breaching the alarm threshold:

- Increase by an instance when a Scaling group instance’s Maximum CPU Utilisation >= 40% for 60 seconds.
- Decrease an instance when a Scaling group instance’s Maximum CPU Utilisation <= 10% for 60 seconds.

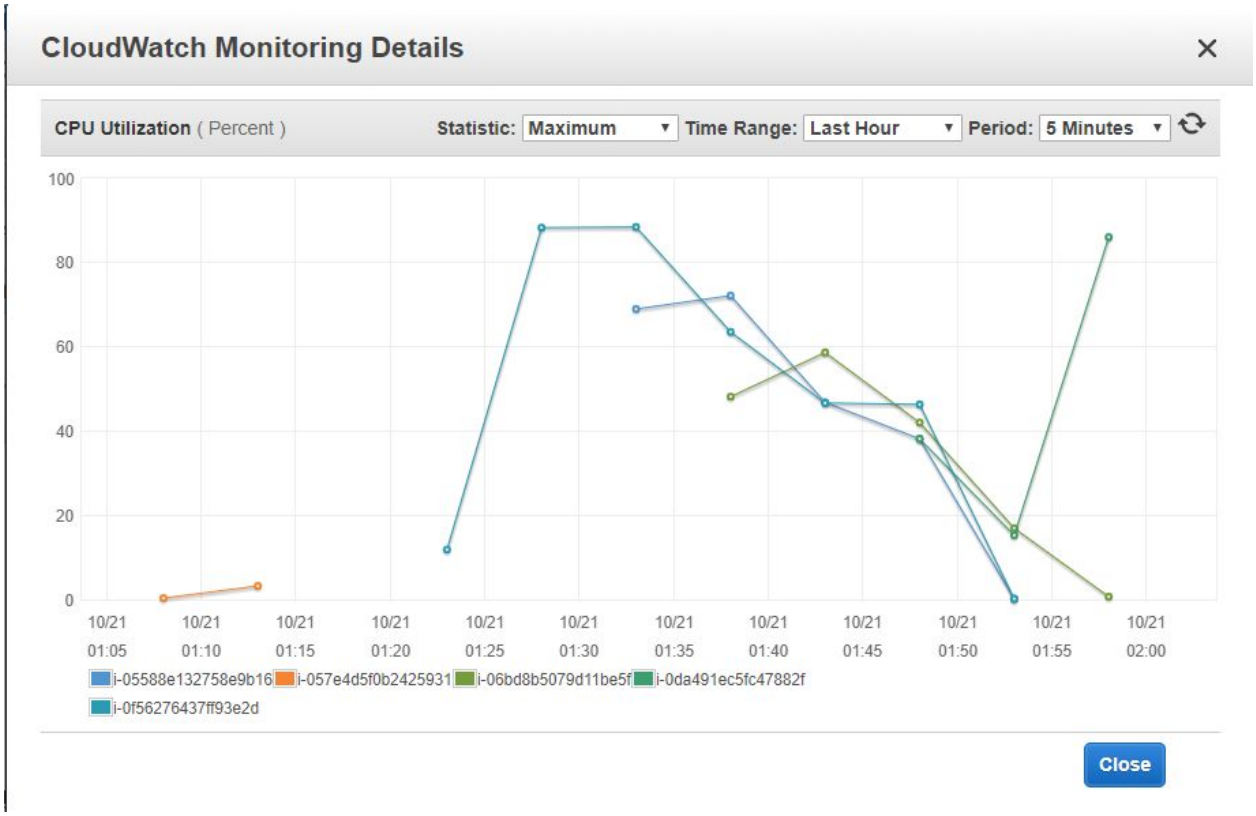
An Elastic Load Balancer is created with the Load Balancer Protocol set to HTTP and port 3000, the Instance Protocol set to HTTP and port 3000. The health check is set to ping at the instances via TCP at port 3000 with response timeout at 10 seconds, health check intervals at 60 seconds, unhealthy threshold at 2 consecutive health check failures, and healthy threshold at 5 consecutive health check successes.

Application Client **section (client and server-side)**

Apart from the www.js and app.js file which creates the node server, a socket.js file is created to constantly receive one second old stored tweet from the DynamoDB. The socket.js receives keywords from the user and whenever a tweet with text related to the keywords matches, the tweet is then send via socket to the client index.js which then processes the “emotion” counts and visualise the results in a real-time chart and a topic chart.

Scaling & Performance

As mentioned in the Architecture and Overall Implementation sections, the scaling of the instances occurs in the comes from the *Sentiment Analysis Processing* section. The scaling is possible through the load generated via processing the tweets and storing the tweets in the DynamoDB. To be safe, a retrieving function to get the exact tweet that was just stored is also implemented in the Sentiment Analysis server to generate more load.



In the diagram shown above, when the the stream server in the *Tweet Stream* section starts (at 01.23 am PST; servers are located in Oregon, US), the first server in the auto-scaling group takes roughly 5 minutes to generate huge load before peaking at around 87% maximum CPU utilisation. The auto-scaling group then keeps on adding more instances until all instances in the scaling group has less than 40% maximum CPU utilisation (not shown in diagram due to the stream server stops early).

When the stream server stops, the maximum CPU utilisation in some instances drops below 10% and the scaling policy starts to terminate some instances. This continues until one instance remains (not shown in the diagram, but it works).

Testing & Limitation

Test Plan & Results

ID	Purpose	Expected/Actual	Pass?	ScreenLink
1	Insert keywords	E: Display tweets with the matching topic A: Display tweets with the matching topic	**Pass**	1.0 - 2.0
2	Clear Real-Time chart	E: Real-Time line chart is cleared A: Real-Time line chart is cleared	**Pass**	3.0
3	Clear Tweet Topic chart	E: Tweet topic bar chart is cleared A: Tweet topic bar chart is cleared	**Pass**	4.0
4	Clear tweet counts and overall sentiment	E: Tweet counts and overall sentiment are cleared A: Tweet counts and overall sentiment are cleared	**Pass**	5.0, 8.0, 9.0
5	CPU exceeds 40% in an auto-scaling instance	E: Increase an instance in the auto-scaling group A: Increase an instance in the auto-scaling group	**Pass**	N/A
6	CPU fails below 10% in an auto-scaling instance	E: Decrease an instance in the auto-scaling group A: Decrease an instance in the auto-scaling group	**Pass**	N/A
7	Load Balancer	E: Balances load well among the auto-scaling instances A: Balances load well among the auto-scaling instances	**Pass**	6.0

Limitations & Compromises

1. **Predefined Tweet Stream filter:**
   Keywords “A” - “Z” are overly represented in the tweet stream. Any other keywords has significantly less occurrence to be displayed in the front-end. Unfortunately, this has to be compromised as Twitter does not allow too many disconnect and reconnection of tweet stream. Twitter would also throttled the tweet stream and even IP ban the streaming client due to excessive disconnect and reconnect.

2. **Generating Load:**
   The predefined filter stream a lot of tweet, which allows the Sentiment Analysis servers to process and store the Tweet object in the DynamoDB to generate load. But load wasn’t generated quick and it wasn’t enough to demonstrate the scaling policies, thus the Sentiment Analysis servers is coded to retrieves the same Tweet object that has been stored immediately to generate more load.

Possible Extensions

A feature that could be added is to display the total number of positive and negative words from all the tweet on the web application. This allows the user to view most common to least common positive/negative words. This can be done by getting the positive and negative words of each tweet via the sentiment library and perform a count on each of them. The total count of these positive and negative words is then displayed on a word cloud where the most common positive/negative words is the largest, whilst the least common positive/negative words is the smallest.

Reference

[1] N. Foundation, "Node.js", *Node.js*, 2017. [Online]. Available: https://nodejs.org/en/. [Accessed: 16- Sep- 2017].

[2] Express, "Express - Node.js web application framework", *Expressjs.com*, 2017. [Online]. Available: https://expressjs.com. [Accessed: 16- Sep- 2017].

[3] M. Rogers, "request", npm, 2017. [Online]. Available: https://www.npmjs.com/package/request. [Accessed: 22- Oct- 2017].

[4] A. Sliwinski, "sentiment", npm, 2017. [Online]. Available: https://www.npmjs.com/package/sentiment. [Accessed: 22- Oct- 2017].

[5] T. Tezel, "twit", npm, 2017. [Online]. Available: https://www.npmjs.com/package/twit. [Accessed: 22- Oct- 2017].

[6] Socket IO, "Socket.IO", Socket.io, 2017. [Online]. Available: https://socket.io/. [Accessed: 22- Oct- 2017].

[7] Amazon Web Services, "aws-sdk", npm, 2017. [Online]. Available: https://www.npmjs.com/package/aws-sdk. [Accessed: 22- Oct- 2017].

[8] EJS, "EJS -- Embedded JavaScript templates", *Ejs.co*, 2017. [Online]. Available: http://ejs.co/. [Accessed: 16- Sep- 2017].

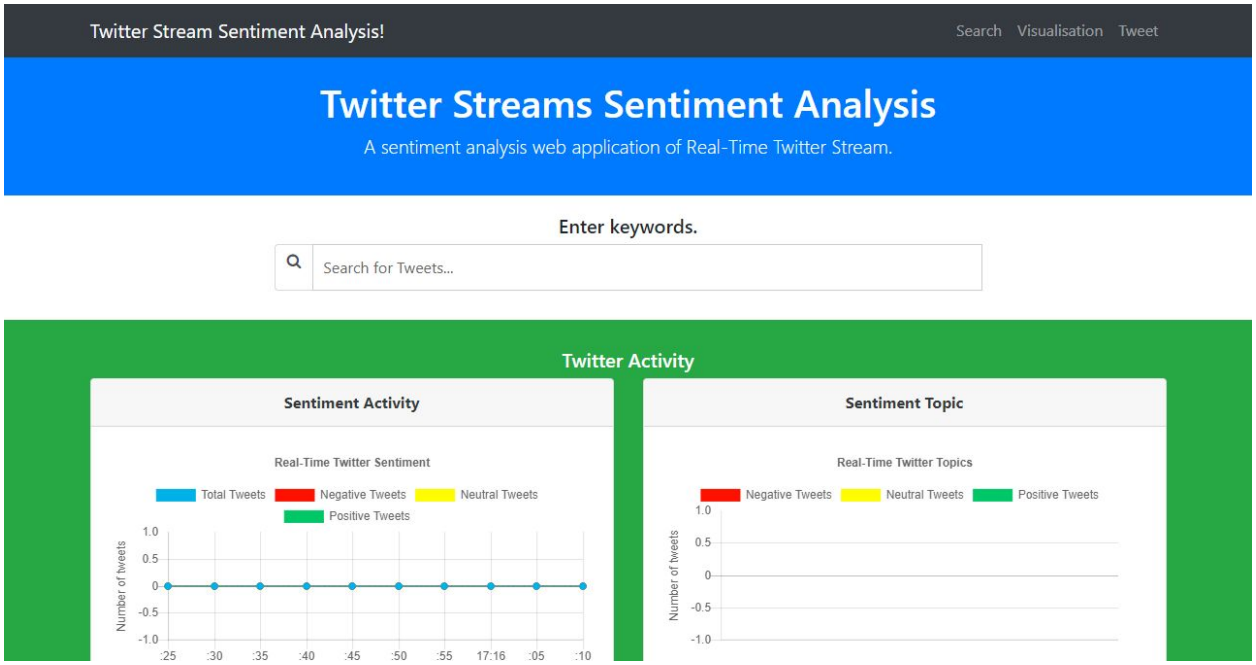
[9] Chart JS, "Chart.js | Open source HTML5 Charts for your website", Chartjs.org, 2017. [Online]. Available: http://www.chartjs.org/. [Accessed: 22- Oct- 2017].

Appendix

Brief User Guide

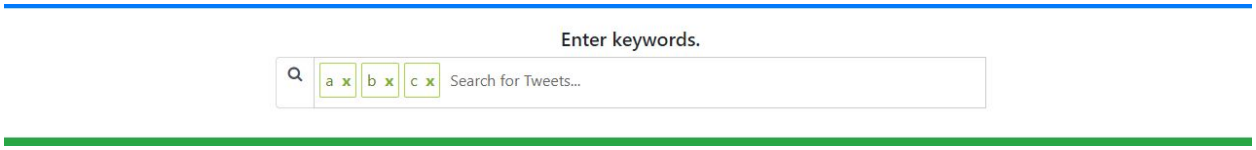
Web Application

1.0



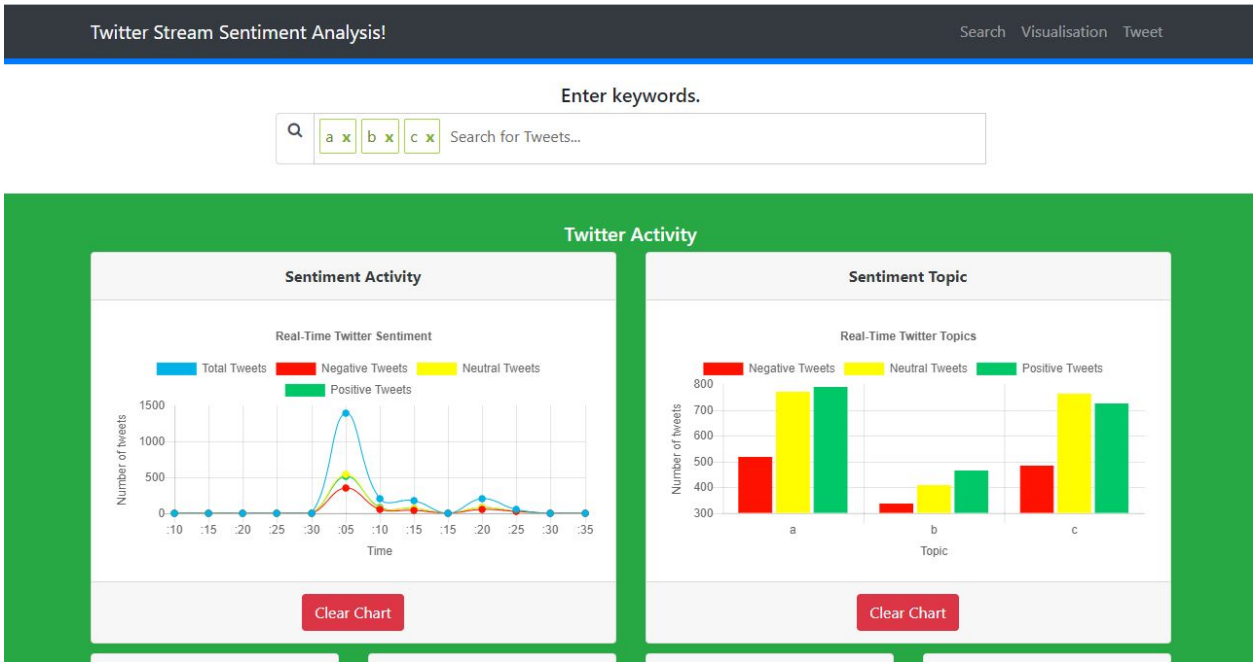
1. This is the web application page. Enter some “keywords” to start searching for tweets with sentiment result.

2.0



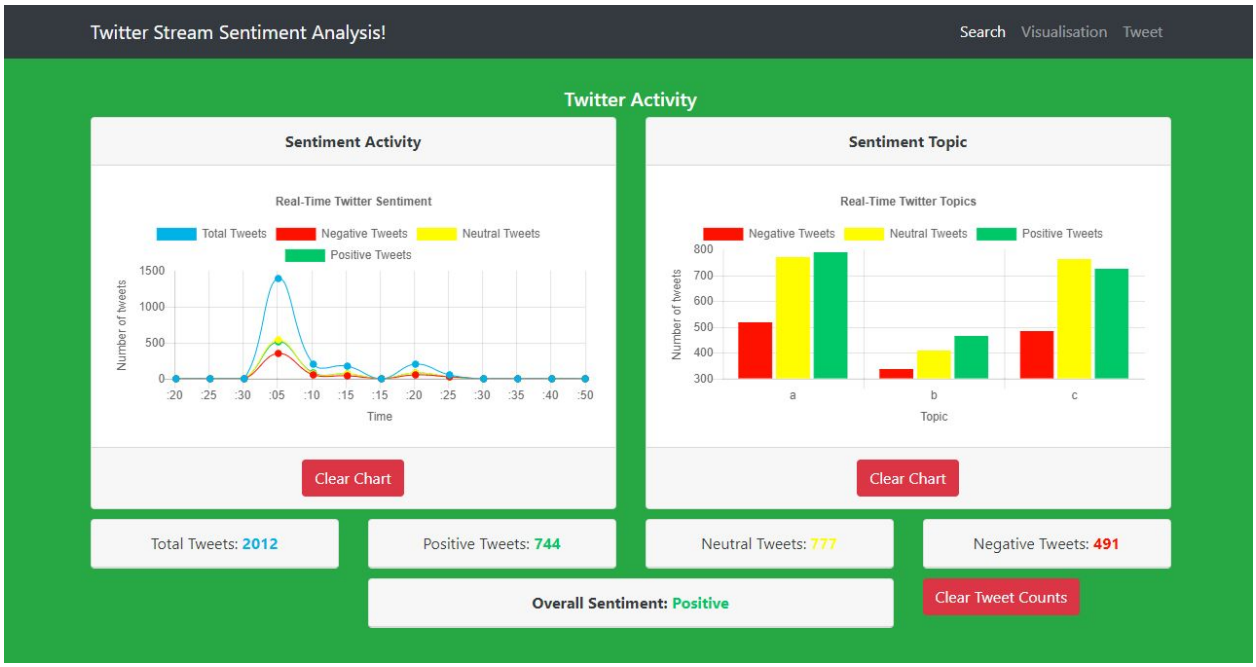
2. You can enter any “keyword” you desire. Let’s go with a, b, and c (since these letter would display lots of tweet.

3.0



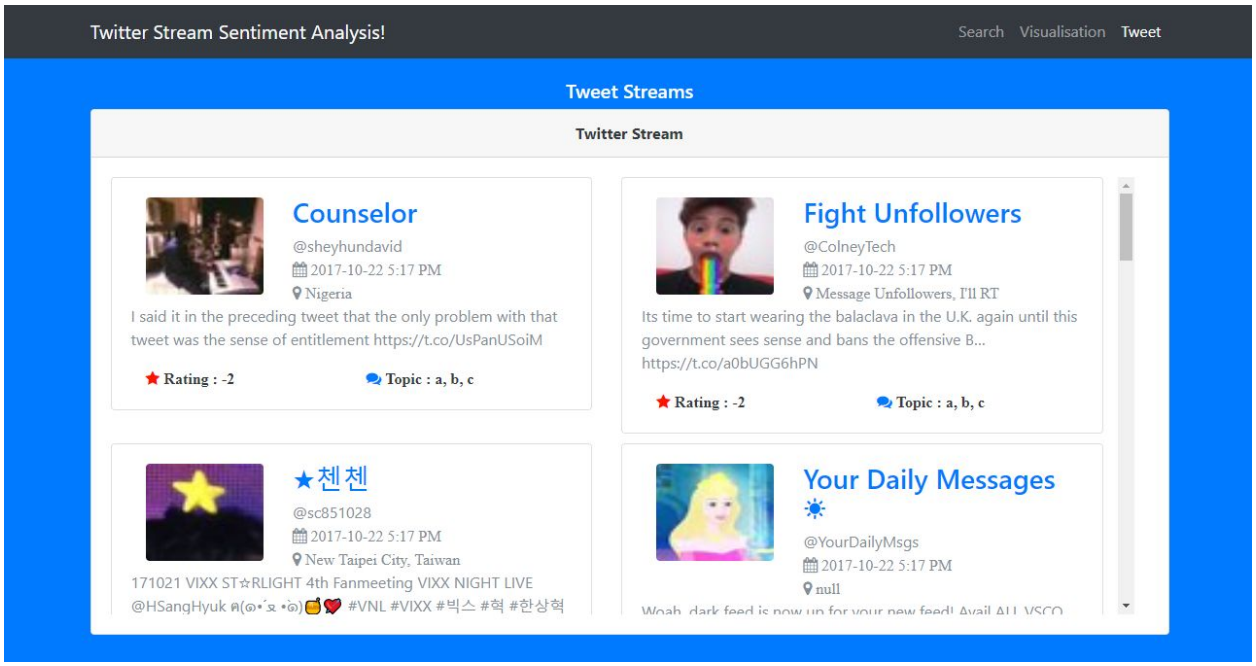
3. Watch as the Real-time line chart (known as Sentiment Activity) and the Topic bar chart (known as Sentiment Topic) starts to populate .

4.0



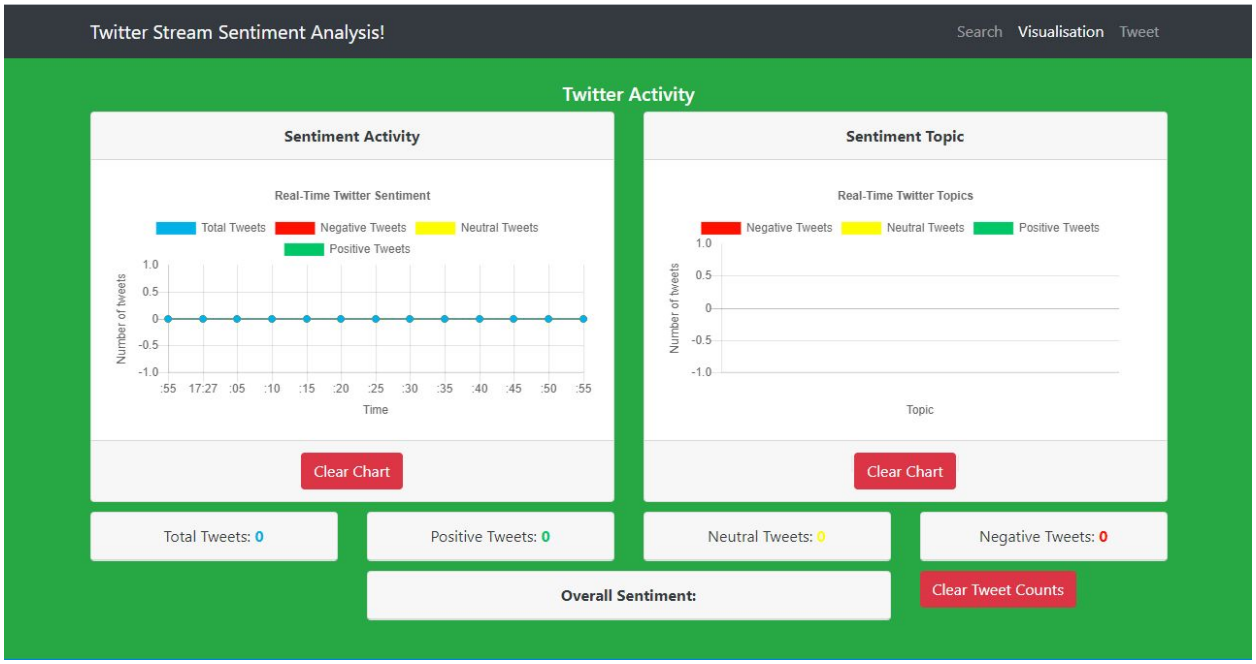
4. The total, positive, neutral, and negative tweet count is also displayed.

5.0



5. Here you can look at the tweets displayed.

6.0



6. Clear the charts and tweet counts by pressing on the “Clear Chart” and “Clear Tweet Count” buttons.

Necessary downloads

***Tweet Stream* section - Stream server**

- Request - https://www.npmjs.com/package/request, npm install request
- Twit - https://www.npmjs.com/package/twit, sudo npm install twit

***Sentiment Analysis Processing* section - Sentiment Analysis server**

- AWS SDK - https://www.npmjs.com/package/aws-sdk, sudo npm install aws-sdk
- Body parser - https://www.npmjs.com/package/body-parser, sudo npm install body-parser
- Cookie parser - https://www.npmjs.com/package/cookie-parser, sudo npm install cookie-parser
- Dataformat - https://www.npmjs.com/package/dateformat, sudo npm install dateformat
- Express - https://www.npmjs.com/package/express, sudo npm install express
- Sentiment - https://www.npmjs.com/package/sentiment, sudo npm install sentiment
- Request - https://www.npmjs.com/package/request, sudo npm install request

***Application Client* section - Client**

- AWS SDK - https://www.npmjs.com/package/aws-sdk, sudo npm install aws-sdk
- Body parser - https://www.npmjs.com/package/body-parser, sudo npm install body-parser
- ChartJS - https://www.npmjs.com/package/chartjs, sudo npm install chartjs
- Cookie parser - https://www.npmjs.com/package/cookie-parser, sudo npm install cookie-parser
- EJS - https://www.npmjs.com/package/ejs, sudo npm install ejs
- Express - https://www.npmjs.com/package/express, sudo npm install express
- Socket IO - https://www.npmjs.com/package/socket.io, sudo npm install socket.io

Instruction for API Keys

None!!

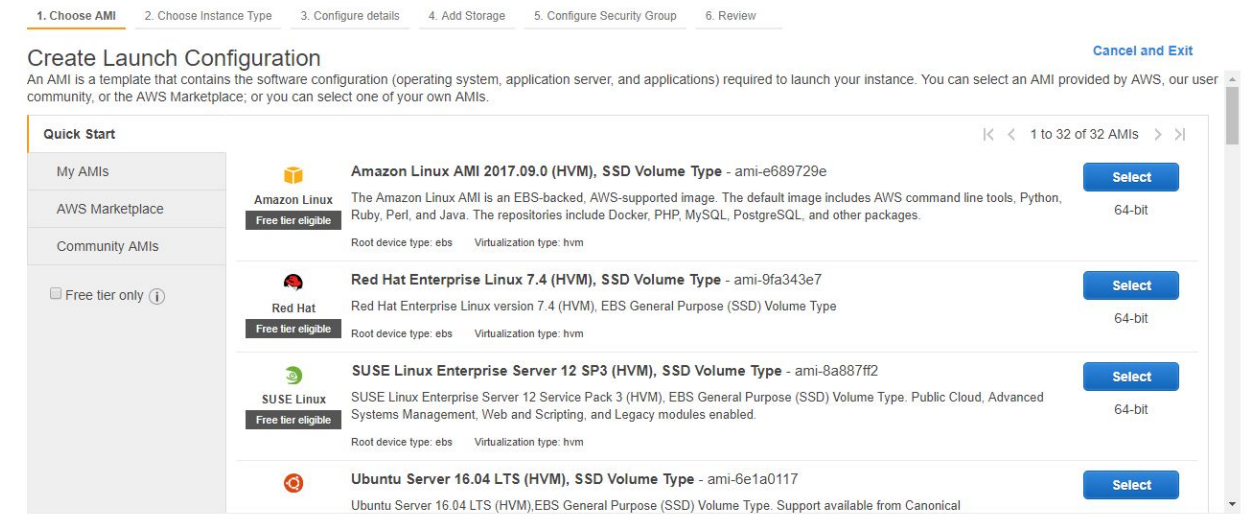
Deployment Instructions

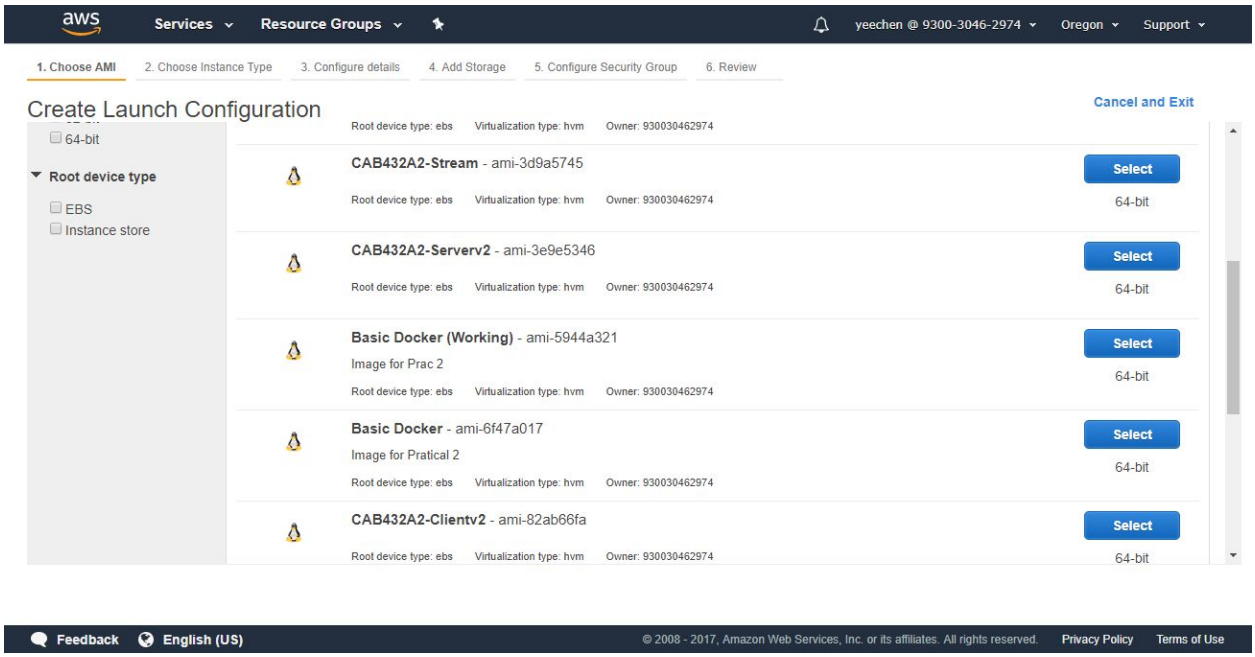
1. Create a new T2 EC2 instance for Stream server, Sentiment Analysis server, and Application Client. Allow port 22 (SSH), 80 (HTTP), and 3000 (Node.js) to be accessible.

2. Deploy the code in the respective EC2 instance in this order:
    - $ sudo apt-get update
    - $ sudo apt-get install git
    - $ curl -sL https://deb.nodesource.com/setup_6.x | sudo -E bash -&& sudo apt-get install -y nodejs
    - $ sudo apt-get install npm
    - $ sudo npm install npm -g
    - $ git clone https://gitlab.com/markyeechen/cab432assign2.git (require username and password)
    - $ cd cab432assign2
    - $ git fetch --all (only needed if code has been updated)
    - $ git reset --hard origin/master (only needed if code has been updated)
    - Only In ***Tweet Stream*** **section - Stream server**:
        - $ cd cab432assign2/twitter/stream
        - $ npm install
    - Only In ***Sentiment Analysis Processing*** **section - Sentiment Analysis server**:
        - $ cd cab432assign2/twitter/server
        - $ npm install
    - Only In ***Application Client*** **section - Client**:
        - $ cd cab432assign2/twitter/app
        - $ npm install
    - $ sudo npm install pm2 -g
    - $ sudo pm2 start npm -- start
    - $ sudo pm2 startup (allow it to start the npm when an instance starts up)
    - $ sudo pm2 unstartup (when you need to change some files, enter this first to shut down npm start)

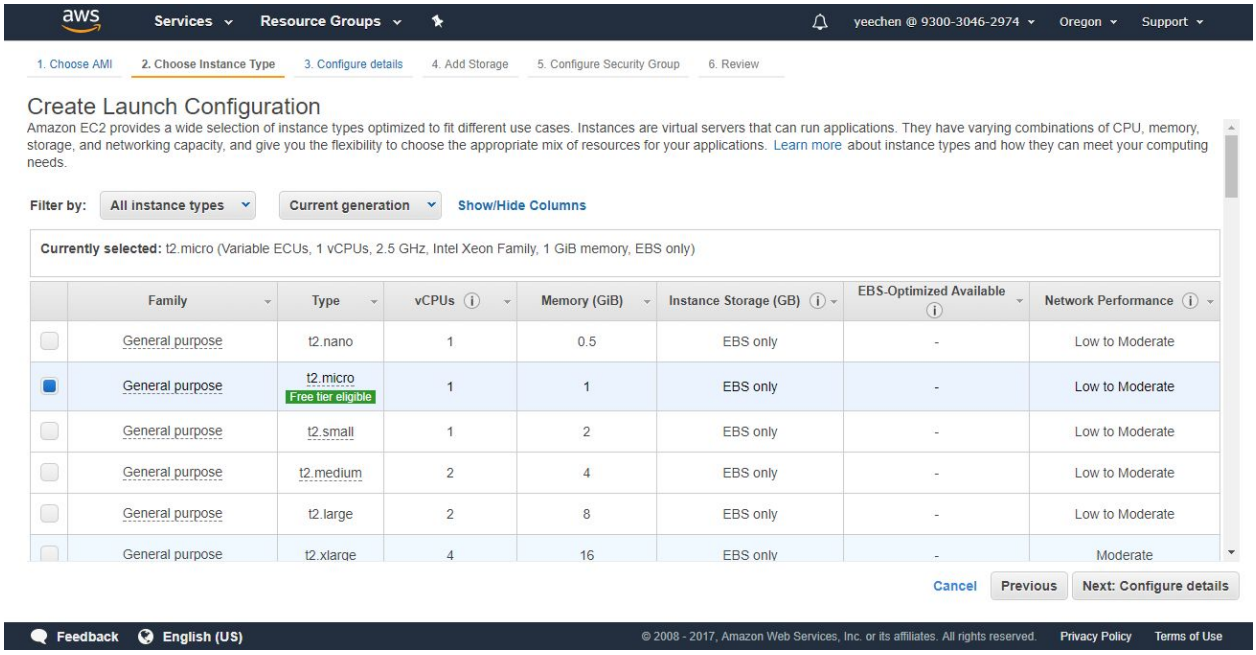
3. Save the instances into an AMI respectively.

4. Create a Launch Configuration as shown below.
    - Select the AMI for the Sentiment Analysis server, in this case it's CAB432A2-Serverv2.

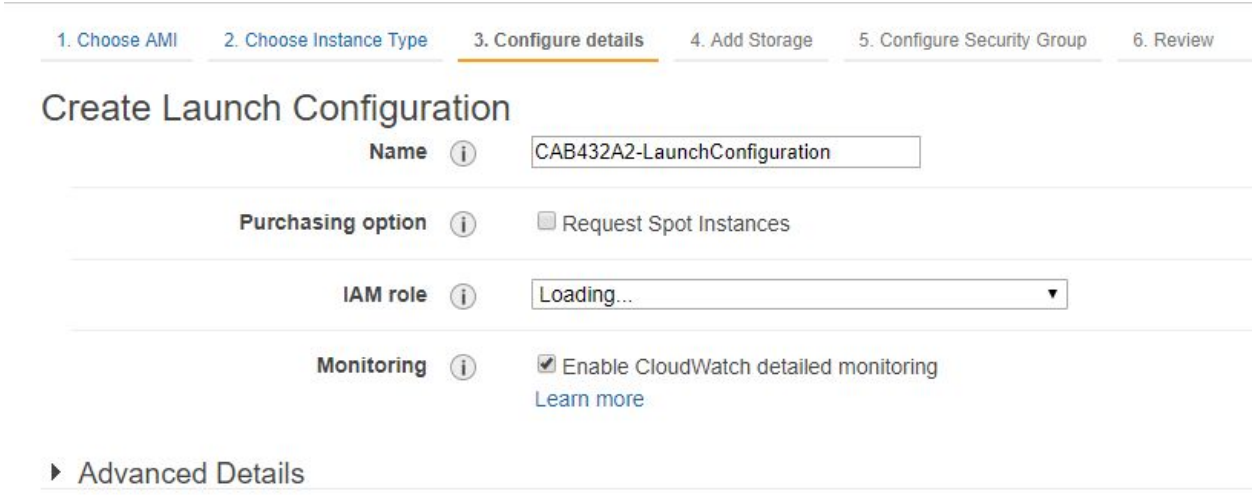




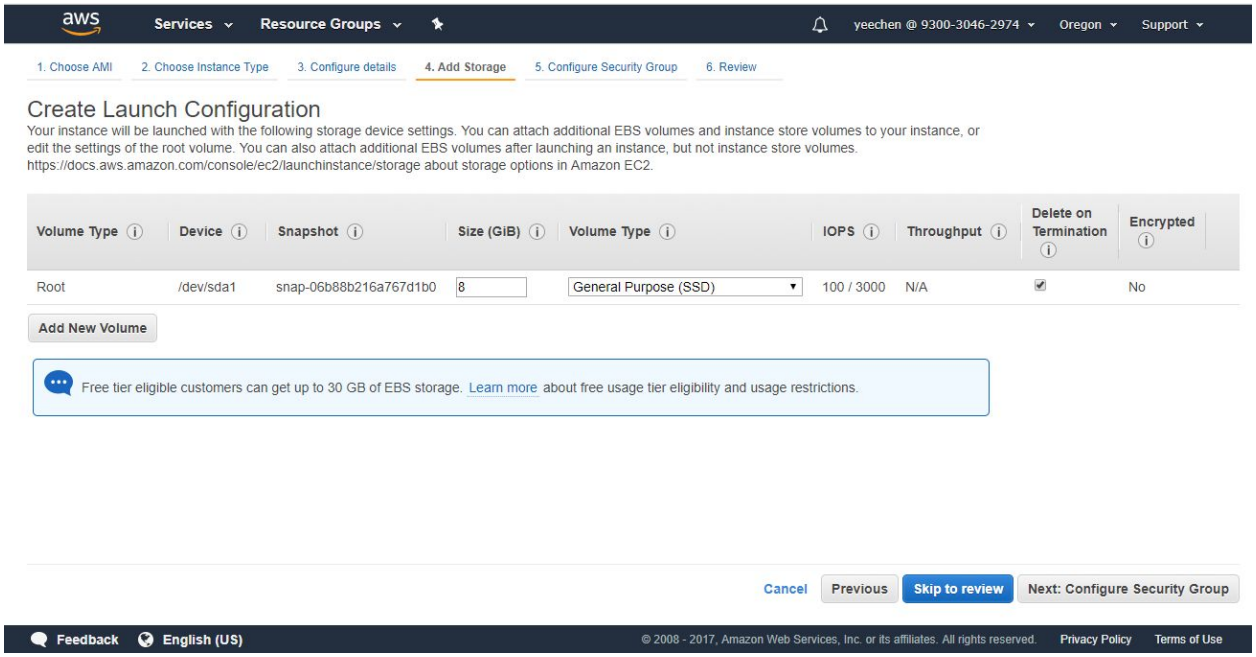
- Select T2 Micro.



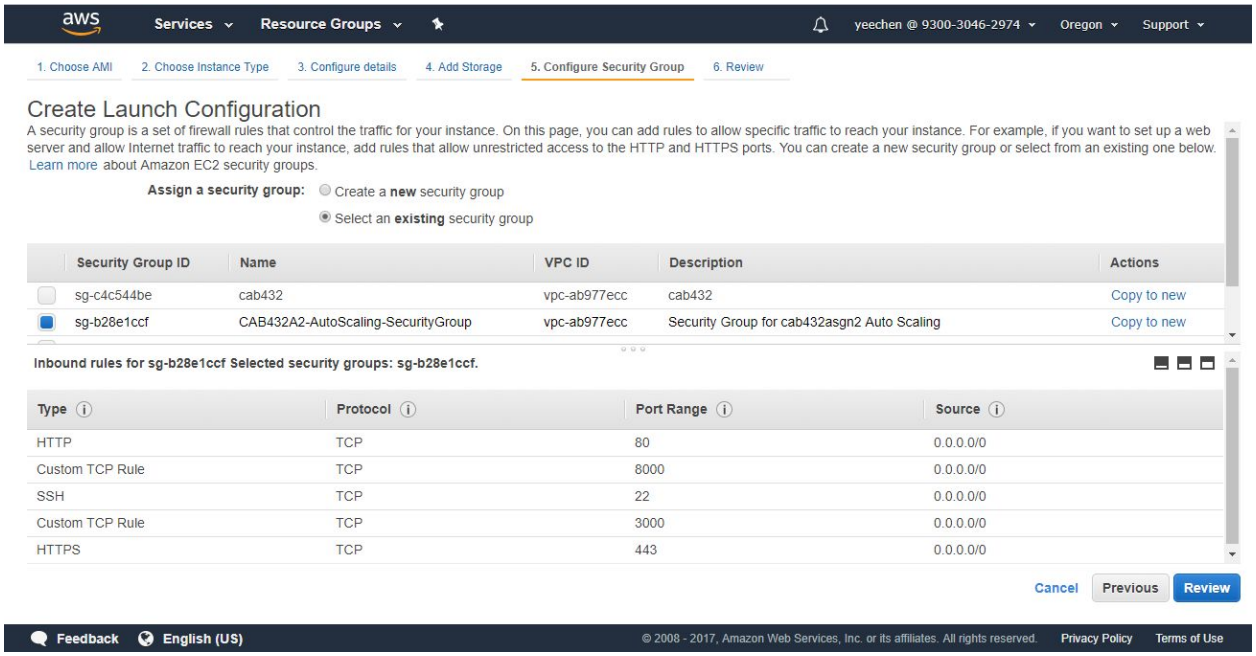
- Set the Configuration details, enable Cloudwatch monitoring.



- Proceed with default hard disk volume

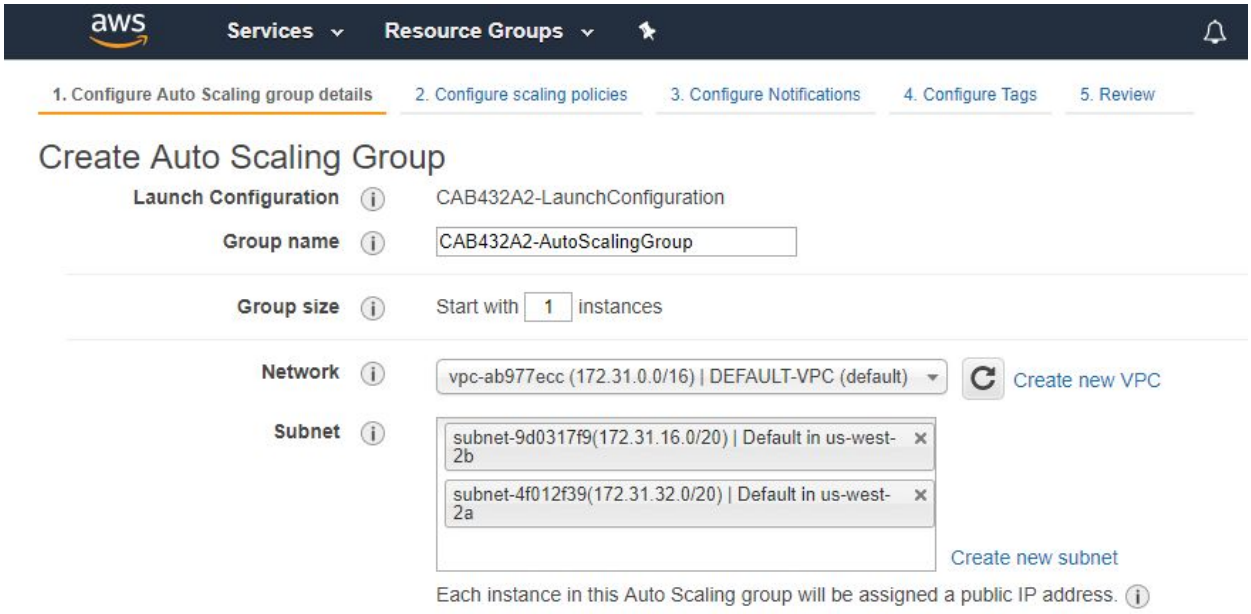


- Allow port 22 (SSH), 80 (HTTP), and 3000 (Node.js) to be accessible.

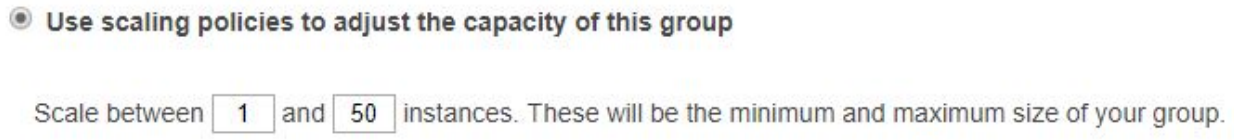


- Finalise the Launch Configuration .

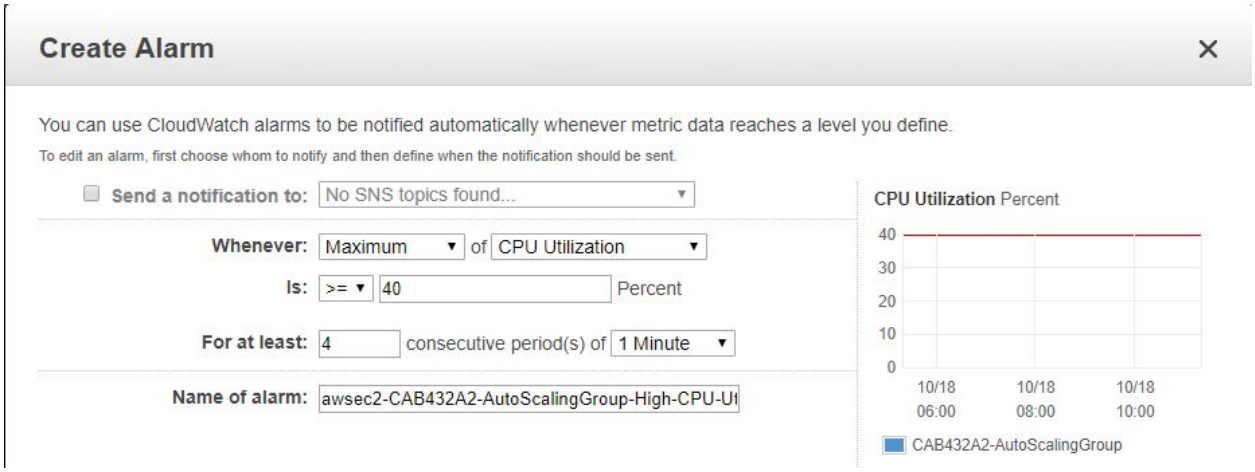
5. Create an Auto Scaling group as shown below.
    - Fill up the details, start the group size with 1 instance. Set the subnet to 2 zones (us-west-2a and us-west-2b).



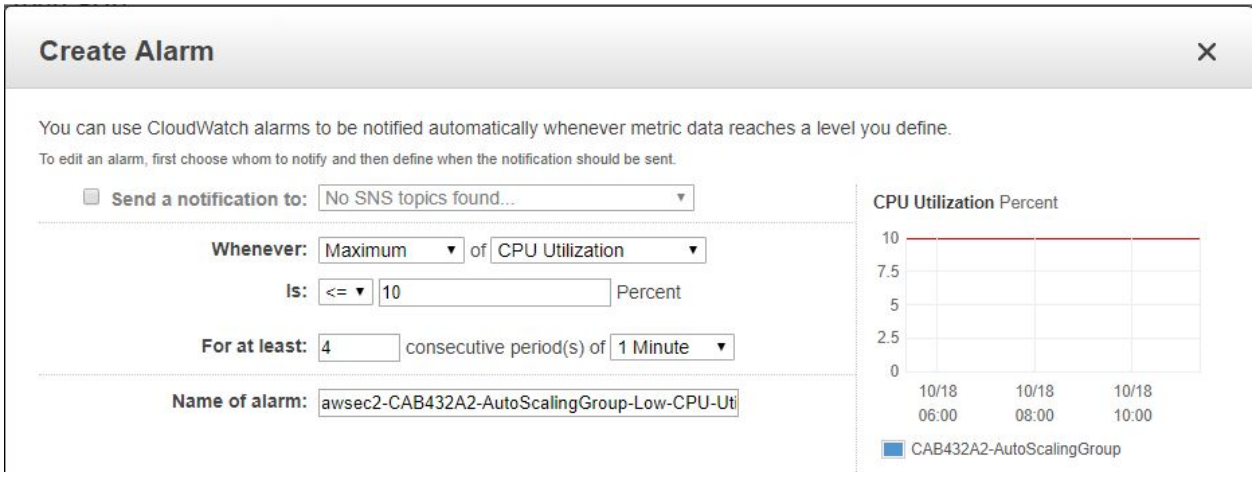
- Set the auto-scaling group to scale between 1 to 50 (or any really) .



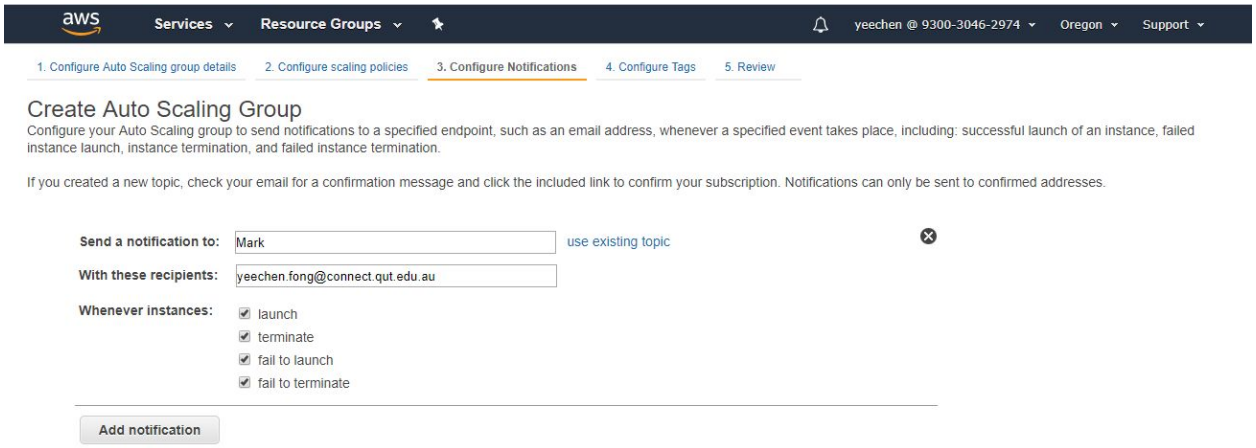
- Create an alarm which increases an instance when a Scaling group instance's Maximum CPU Utilisation >= 40% for 4 consecutive periods of 1 minutes (we changed it to 1 consecutive periods of 60 seconds later on).



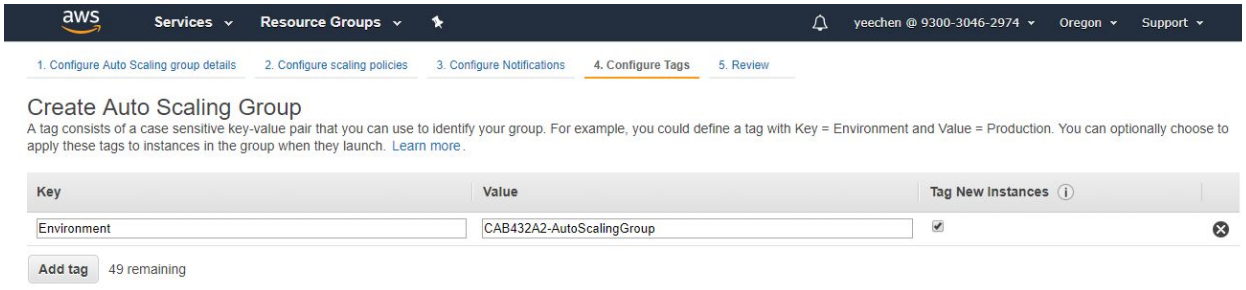
- Create an alarm which decreases an instance when a Scaling group instance's Maximum CPU Utilisation <= 10% for 4 consecutive periods of 1 minutes (we changed it to 1 consecutive periods of 60 seconds later on).



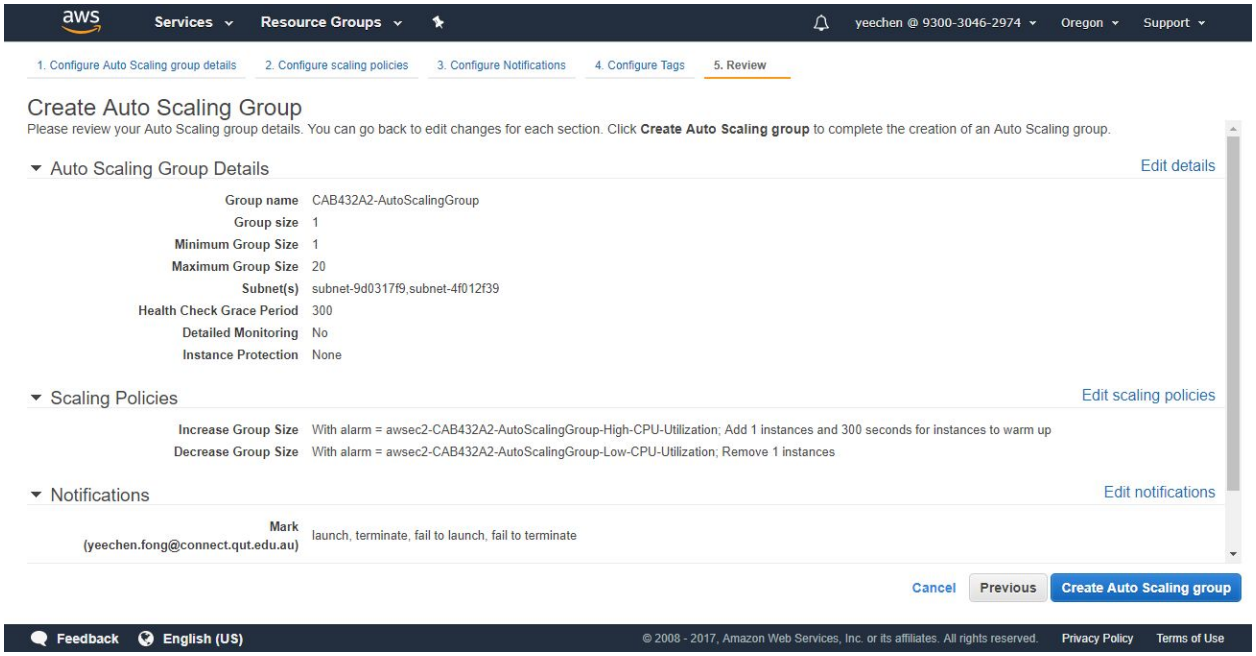
- Create notifications (optional) .



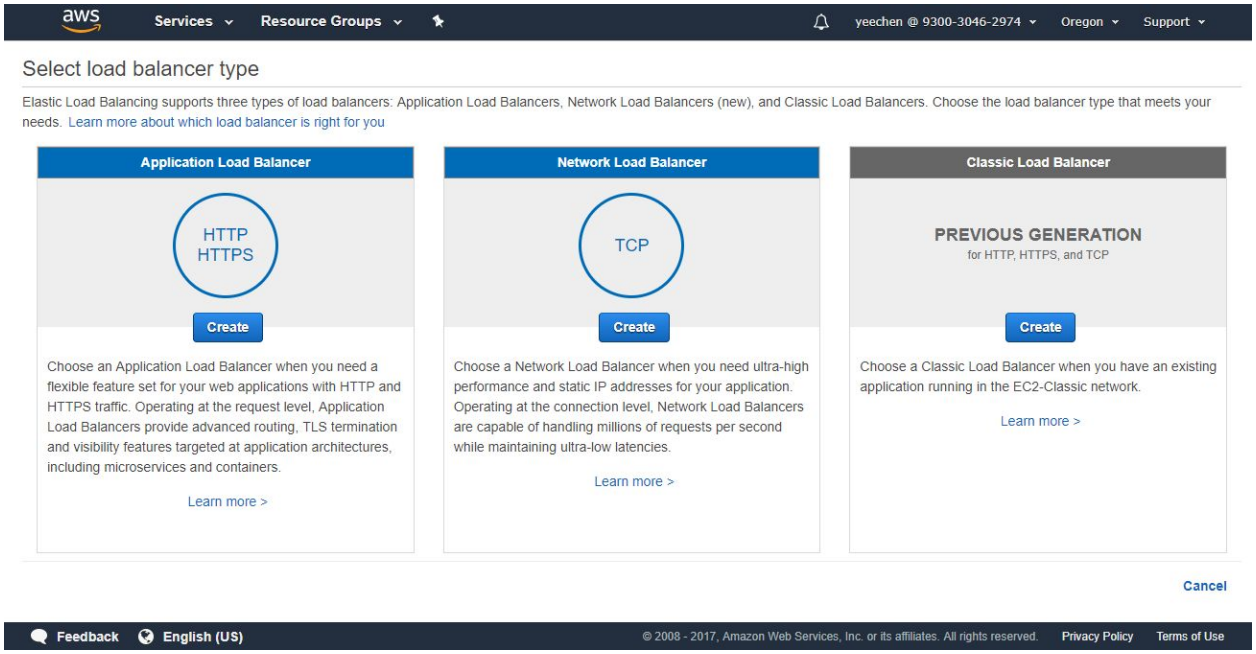
- Configure auto-scaling group tags (optional).



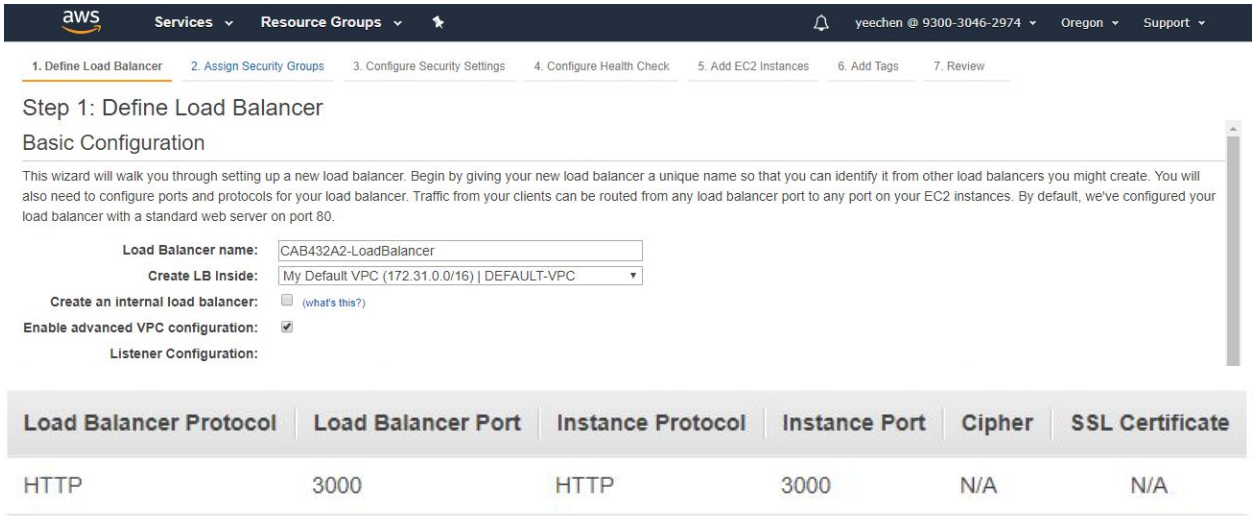
- Finalise auto-scaling group settings .



6. Create a Load Balancer as shown below.
    - Select the Classic Load Balancer.

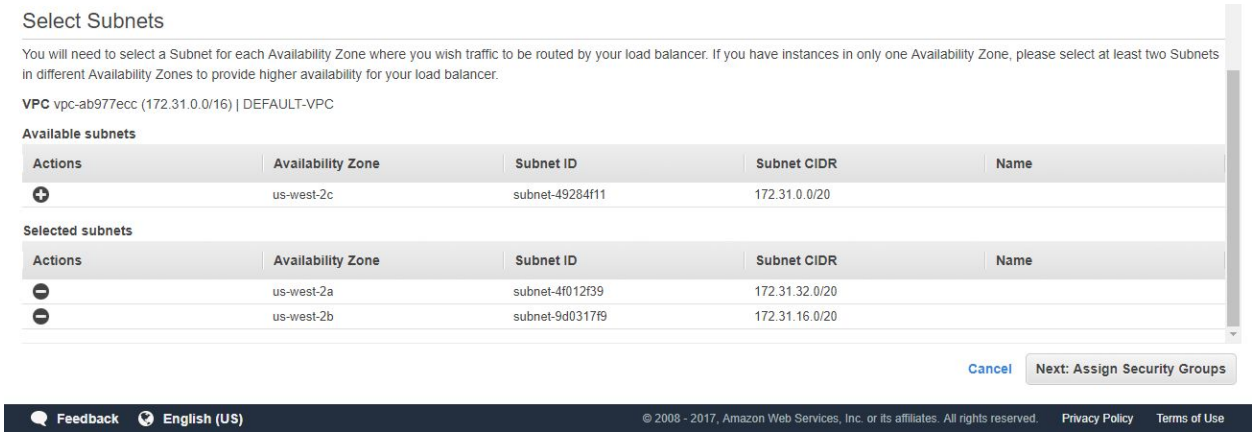


- Define the Load Balancer details, with the Load Balancer Protocol set to HTTP and port 3000, the Instance Protocol set to HTTP and port 3000.

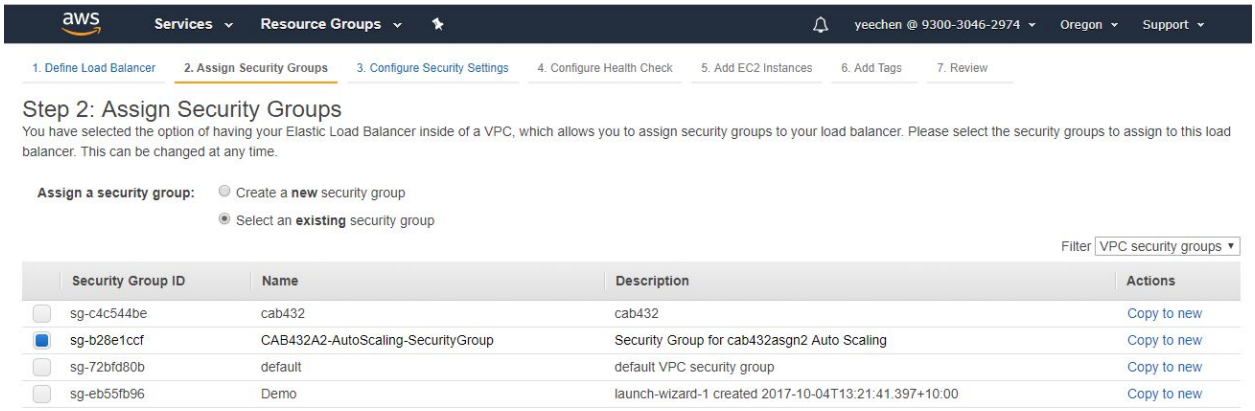


Load Balancer Protocol	Load Balancer Port	Instance Protocol	Instance Port	Cipher	SSL Certificate
HTTP	3000	HTTP	3000	N/A	N/A

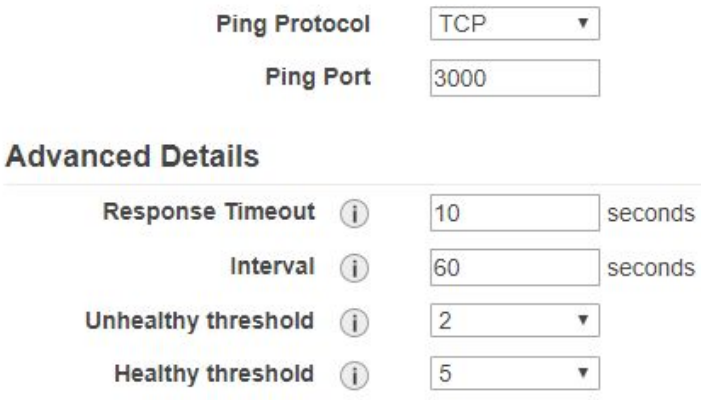
- Select 2 subnet zones (us-west-2a and us-west-2b).



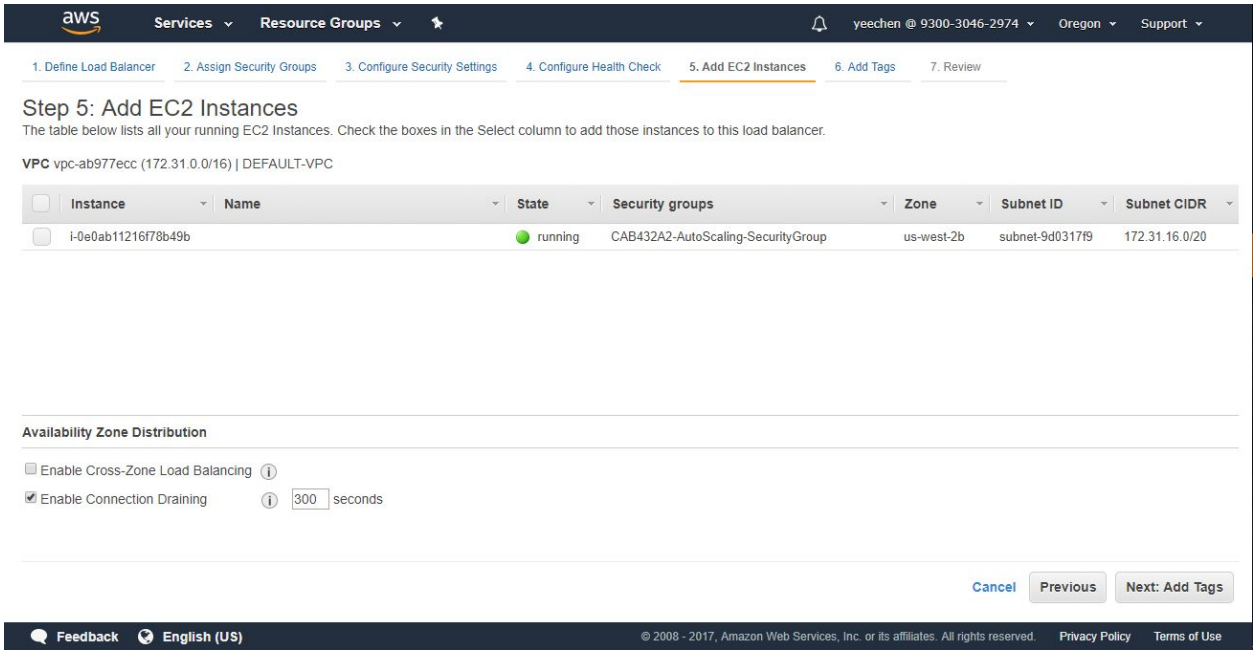
- Allow port 22 (SSH), 80 (HTTP), and 3000 (Node.js) to be accessible.



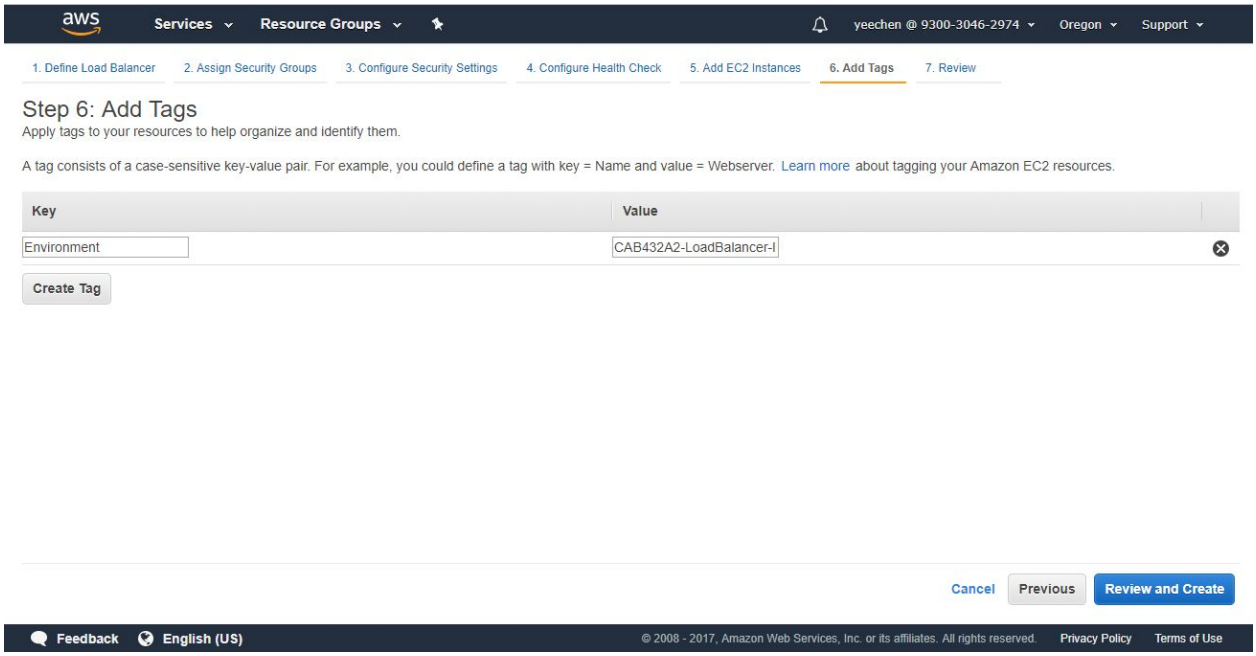
- The health check is set to ping at the instances via TCP at port 3000 with response timeout at 10 seconds, health check intervals at 60 seconds, unhealthy threshold at 2 consecutive health check failures, and healthy threshold at 5 consecutive health check successes.



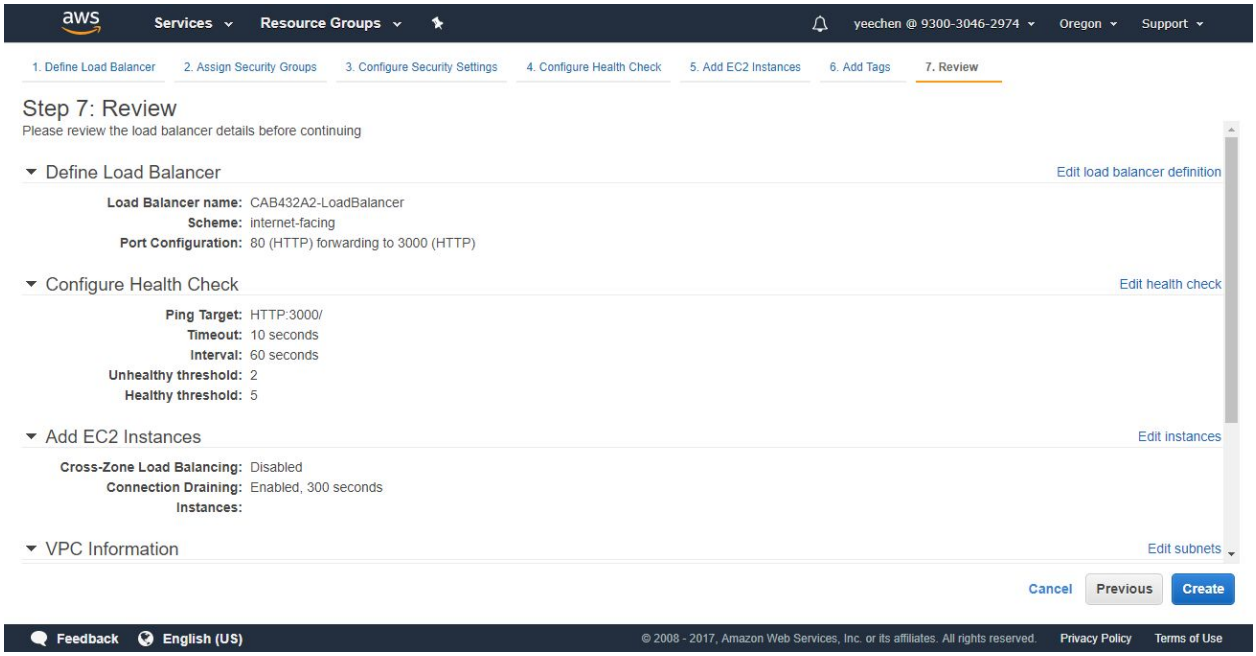
- Skip adding EC2 instances.



- Configure Load Balancer tags (optional).



- Finalise Load Balancer settings .



7. Add the newly configured Load Balancer to the Auto-Scaling group.