Assignment 2 Report

CAB432 Cloud Computing – semester 2 2019

John Santias N9983244

Ka long Lee n9845097

2019

Contents

[Introduction 2](#_Toc22649821)

[Development 2](#_Toc22649822)

[Stage 1: 2](#_Toc22649823)

[Stage 2: 2](#_Toc22649824)

[Stage 3: 2](#_Toc22649825)

[Stage 4: 2](#_Toc22649826)

[API and Packages 3](#_Toc22649827)

[Twitter API 3](#_Toc22649828)

[IBM Watson Natural Language Understanding API 3](#_Toc22649829)

[MongoDB 3](#_Toc22649830)

[ChartJS 3](#_Toc22649831)

[Use Cases 3](#_Toc22649832)

[Technical Description 4](#_Toc22649833)

[Cloud Architecture 4](#_Toc22649834)

[Server Architecture 5](#_Toc22649835)

[Client Side 5](#_Toc22649836)

[Server Side 5](#_Toc22649837)

[AWS Elastic Load Balance 6](#_Toc22649838)

[AWS Auto-Scaling Group 6](#_Toc22649839)

[AWS EC2 Server 6](#_Toc22649840)

[Docker 6](#_Toc22649841)

[MongoDB 6](#_Toc22649842)

[Scaling and Performance 7](#_Toc22649843)

[Scaling Up 7](#_Toc22649844)

[Scaling Down 7](#_Toc22649845)

[Testing and Limitations 7](#_Toc22649846)

[Test Plan 7](#_Toc22649847)

[Possible Extensions 8](#_Toc22649848)

[Appendix 8](#_Toc22649849)

[Appendix A –Cloud Architecture design Iterations 8](#_Toc22649850)

# Introduction

Tweetery aims to provide users with a visual representation of the emotional value of tweets on a search or trending topic. Utilising the Twitter API for getting tweets into the web server, cleaning the tweet body and performing emotional analysis with IBM Watson API. The average emotion scores of the chosen topic is displayed on a graph rendered on the client side.

When the user searches for a topic, the query is sent to the node server for further processing. The server uses the search query to get tweets from Twitter API. The received tweets are then combined and sent to IBM’s Watson Natural Language Understanding API to analyse the emotions. These results are stored on a remote global Redis Cache server which can be returned to users who searches for the same topic. Results are also stored on MongoDB for long term storage and can be copied to the cache upon user request. The node server will repeat this processing for each of the last seven days.

Between the client and node server, and the node server and Redis cache server, is a load balancer to assist in handling high volume of traffic that can be generated by popular trending topics and queries. The load balancer assists in the scaling of our servers.

# Development

Tweetery will be developed in stages in order to have tight grip on the scope of the application and reduce the number of potential issues towards the end of development.

## Stage 1:

Stage 1 involved setting up the skeleton of the project and generating the keys for the Twitter and IBM API. The skeleton for the node application consisted of two folders, one dedicated for the client, the other for the server. Dependencies were also installed for the node application to process API calls. As expected by the end of this stage, the application was able to call the Twitter API for trending topics and tweets based on single/multiple queries for emotional analysis using IBM’s API.

## Stage 2:

Stage 2 involved creating the front-end UI to display components such as the trending topics list, a navigation bar, search bar and a graph for the processed emotional analysis. It was ensured that client side was calling the correct server endpoints for heavy processing.

## Stage 3:

Stage 3 involved setting up two storage services for the node application to store and retrieve data. This included setting up a Redis cache server and a mongoDB database. The node application was connected to both services, then stored and retrieved data based on a defined Schema. Afterwards, testing and bug fixing was conducted followed by dockerising the application.

## Stage 4:

Stage 4 involved deploying to the cloud with further testing and bug fixing. Monitoring the application was an additional step to this stage 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.

During this monitoring process it was discovered that the application was not producing enough CPU load for scaling, so we tried to move the redis server from being a global cache to a container image linked to the container running our node server. We tried to increase the processing load by adding more libraries such as …..

## Stage 5 (Additional feature):

We attempted to add a new feature that would emotionally analyse real time tweets and plot new emotion results on the graph every 3 three seconds.

# API and Packages

## Twitter API

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

Twitter is a social network platform for users to share messages, images and videos on a feed. Messages can be categorised by hashtags (#) and searches can be conducted on those hashtags or keywords. Tweetery uses this API to retrieve tweets based on a hashtag or keyword.

## IBM Watson Natural Language Understanding API

<https://www.ibm.com/watson/service/natural-language-understanding>

The IBM Watson Natural Language Understanding API is one of IBM’s Watsons machine learning API. It is used to process advanced test analysis and extract metadata from content such as concepts, entities, keywords, categories, sentiment, emotions, relations, and semantic roles. Some of which can provide a score for the tone or a list of words that have great meaning for the user.

Tweetery uses this API to analyse the emotion of the tweets gathered and return a score between zero and one. Given the highest score is the emotion analysed in the content.

## MongoDB

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

MongoDB is a cross-platform document-oriented database program. Classified as a NoSQL database program and uses JSON-like documents with schema. mLab was used to host MongoDB databases on a cloud provider (AWS, GCP, Azure). Tweetery uses two tables to store emotion analysis, .

## ChartJS

<https://www.chartjs.org>

ChartJS is a community-maintained project for developers to visualise data in eight different ways. Each of them animated and customisable. Tweetery uses this library to visualise the emotional scores of the given topic.

# Use Cases

*As a user, I want to see a list of trending topics and select one to see the average emotional stats.*

The user navigates to the home page of tweetery. User selects a trending topic from the right-hand side and sees the results.

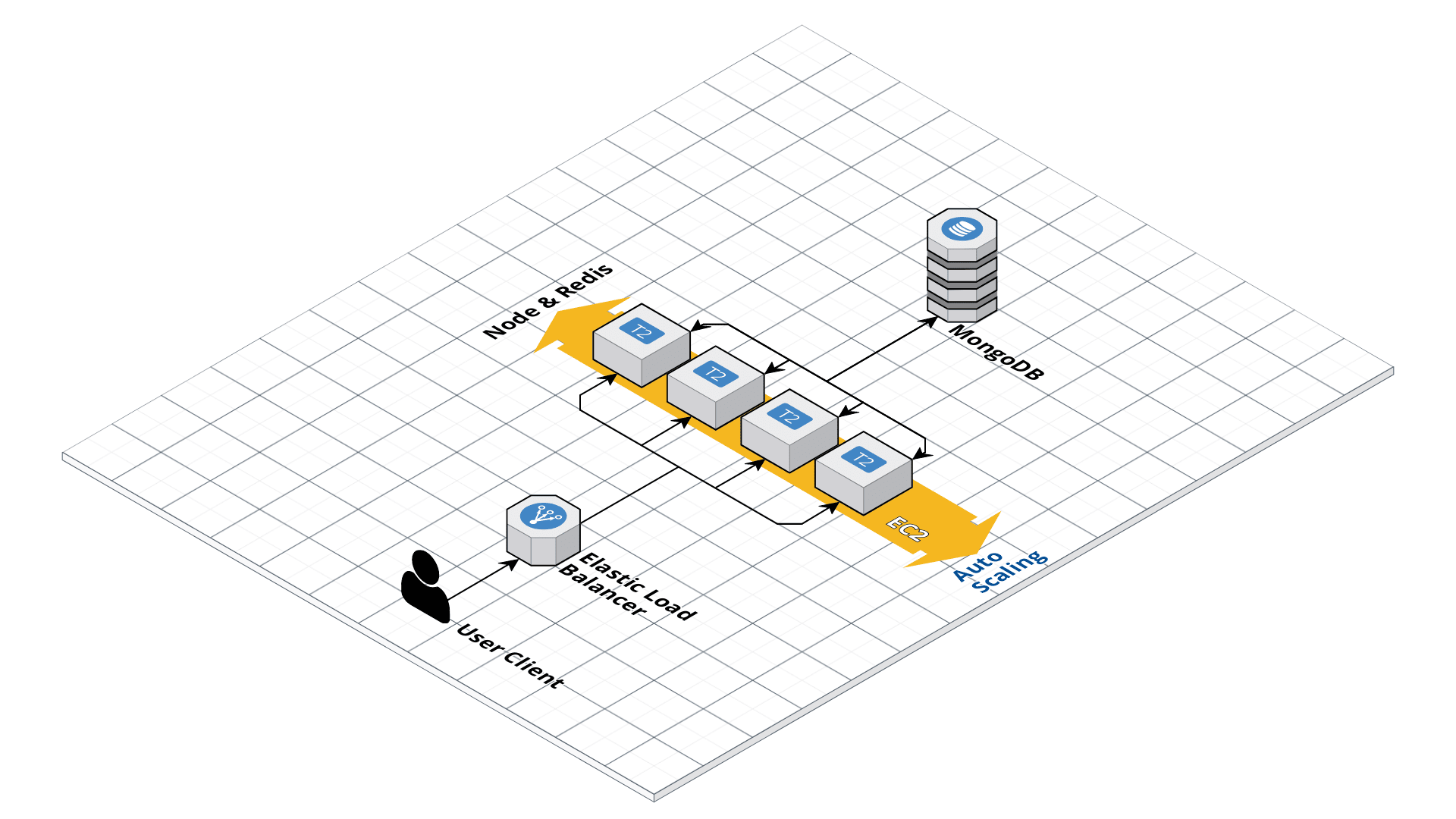
*As a user, I want to select or search multiple trending topics to compare the emotions of those topics.*

The user selects/searching multiple topics and sees the results on the same page.

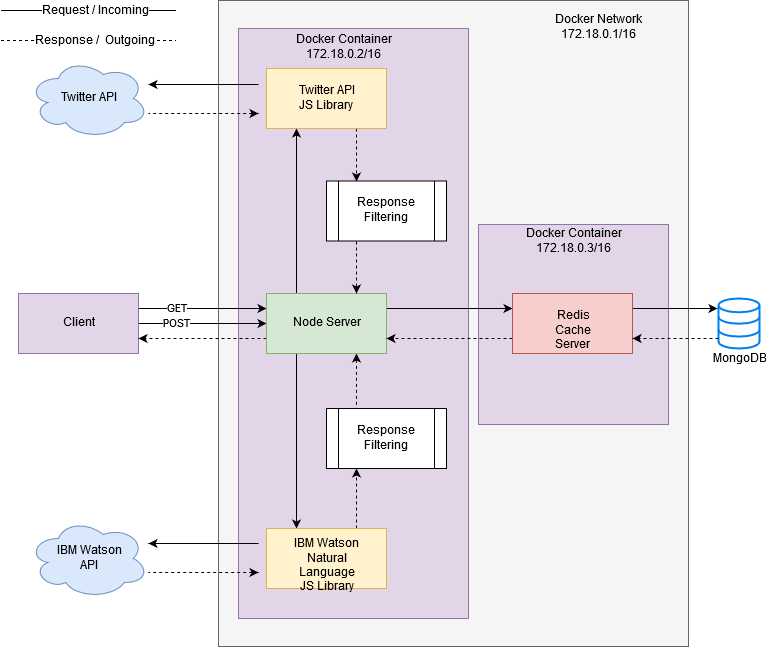
# Technical Description

Tweetery utilises one EC2 t2.micro server achieve its tasks in a stateless manner. Inside this server runs two docker containers inside of a bridged network. One running NodeJS application to process user requests, and the other running a redis server to cache data. In addition to this, MongoDB is utilised to provide persistence.

## Cloud Architecture



## Server Architecture



# Client Side

The client can initiate two types of request, a GET and a POST request. When a user requests to access the application, a GET request is sent to the server. The server returns the index page with a nav bar, a list of trending topics and a search bar. When the user initiates a search, the form body sends a POST request to the server for processing.

The server handles the query sent from the form, processes the API calls with the query and displays the results in a graph. The client uses the ReactJS framework to render the page and information being sent to the user.

# Server Side

The node server manages the incoming client request and outgoing responses. When it receives a GET request, the server renders and returns the home page with a nav bar, search bar and trending topics. There is no heavy processing in this request because the client hasn’t searched for anything. The client is only requesting the home page.

The heavy processing is initiated when the server receives a POST request containing the search queries. The server processes one query at a time. Using a query to search tweets for each of the last seven days. By hitting the twitter search endpoint, it returns a JSON object with tweet data stored in an array. Each element containing the tweet message, user information, time, date etc. At this point the server extracts the tweet messages, combines them and sends it to IBM’s API for emotional analysis. The result of this analysis is pushed to a JSON object with the date and query, then stored in the Redis cache for short-term storage and mongoDB for long-term storage. This information is also rendered and sent to the client for display. This process is repeated for each search query.

The server runs in a stateless manner, relying on storage services and APIs for storing and searching information. In every search initiated, the server always checks if the data exists on Redis and mongoDB before running a new search on the APIs.

## AWS Elastic Load Balance

An AWS Elastic Load Balancer automatically distributes incoming application traffic across multiple targets. Tweetery utilises a load balancer to balance the traffic between all the instances in the Auto-Scaling group.

## AWS Auto-Scaling Group

An AWS Auto-Scaling group creates EC2 instances dynamically based on the defined conditions. It maintains the number of instances by performing periodic health checks on the instances in the group. Tweetery utilises this to ensure consistency by scaling when server’s resources are overloaded.

## AWS EC2 Server

All of the servers are launched as a t2.micro server in the Asia Pacific (Sydney) region. Each of the server runs a custom bash script on boot to check for the latest version of the docker image. If there’s a new version the server will pull and run it. This was used throughout development, allowing me to make changes locally on my machine, create and push the docker image and reboot the server.

## Docker

The Node and Redis servers are separated into its own containers and linked to each other for storage. These containers are run inside of an independent network (172.18.00/16) connected to the default network bridge for ease of access and minimising multiple port forwarding rules. By default, public users can connect to the node application through port 5000 and the node application can access the cache through port 6379.

(INSERT image of docker network, and container)

## MongoDB

MongoDB was remotely hosted through mLab. Tweetery utilises a NoSQL database to write and retrieve records. Collections ‘Trends’ to store trending data, and ‘Emotions’ to store emotional data. MongoDB was chosen due to the ease of use and documented-oriented storage suitable for our JSON data.

# Scaling and Performance

## Scaling Up

## Scaling Down

# Testing and Limitations

## Test Plan

|  |  |  |  |
| --- | --- | --- | --- |
| **TASK** | **Expected Outcome** | **Result** | **Screenshot/s (Appendix)** |
| Retrieve Home | Display the logo |  |  |
| Display the search box |  |  |
| Display the trending topics |  |  |
| Search tweets | Search button sends query |  |  |
| Graph displays |  |  |
| Graph displays Legend |  |  |
| Graph displays Labels |  |  |
| Graph displays Date |  |  |
| Graph displays |  |  |
| Add row for search query |  |  |
| Delete search row |  |  |
| Initiate search with more than one topic |  |  |
| Search boxes filled on selection of trending topic |  |  |
| Handle Twitter API response error | Application continues running, display error |  |  |
| Handle IBM Natural Language Understanding API response Error | Application continues running, display error |  |  |
| Stream live tweets | Live tweets analysed and displayed on the graph |  |  |
| Stream doesn’t write after close |  |  |
| API request errors shown to the user without crashing the application |  |  |
| Data to and from Redis | Node server connected to Redis |  |  |
| Node server stores data in Redis by specified key |  |  |
| Node server retrieves data from Redis by key specified |  |  |
| Data to and from MongoDB | Node server connected to mongoDB |  |  |
| Node server stores data in mongoDB by specified key |  |  |
| Node server retrieves data from mongoDB by key specified |  |  |
| Load Balancer | Load balancer health check |  |  |
| Auto Scaling | Auto scale up |  |  |
| Auto scale down |  |  |

# Difficulties

Scaling AWS instances

Scaling twitter stream

# Possible Extensions

# Appendix

## Appendix A –Cloud Architecture design Iterations

