2020

Benjamin Semple N10533885 &

Joshua Paterson n10193197

9/30/2020

<your application name>

CAB432

Assignment 2

Contents

[Introduction 2](#_Toc52357527)

[Purpose & description 2](#_Toc52357528)

[Services used 2](#_Toc52357529)

[Twitter Standard Search API (v.1.1) 2](#_Toc52357530)

[API 2 2](#_Toc52357531)

[API n 2](#_Toc52357532)

[Use cases 2](#_Toc52357533)

[US 1 2](#_Toc52357534)

[US 2 2](#_Toc52357535)

[US n 3](#_Toc52357536)

[Technical breakdown 4](#_Toc52357537)

[Architecture 4](#_Toc52357538)

[Context diagram 4](#_Toc52357539)

[Sequence Diagram 4](#_Toc52357540)

[Process flow Diagram 5](#_Toc52357541)

[Network diagrams (Cloud specific) 5](#_Toc52357542)

[Client / server demarcation of responsibilities 5](#_Toc52357543)

[Response filtering / data object correlation 5](#_Toc52357544)

[Test plan 7](#_Toc52357545)

[Difficulties / Exclusions / unresolved & persistent errors 8](#_Toc52357546)

[Extensions (Optional) 8](#_Toc52357547)

[User guide 8](#_Toc52357548)

[References 8](#_Toc52357549)

[Appendices 8](#_Toc52357550)

## 

## Introduction

### Purpose & description

The application this report detail is Scenario 1 provided in the Assessment 2 brief. The purpose of the app is to filter live twitter messages by user provided tags/inputs and use those messages to perform live and in real time sentiment analysis to determine the overall sentiment of twitter messages with the provided tags at any given moment. This app provides value as it provides the ability to quickly get a general idea of the publics sentiment of any topic.

### Services used

#### Twitter Standard Search API (v.1.1)

Returns a collection of relevant Tweets matching a specified query – may also be filtered based on popularity or geocoding however for this project filtering was based on tags.

Endpoint: https://api.twitter.com/2/tweets/search/recent

Docs: <https://developer.twitter.com/en/docs/twitter-api/v1/tweets/search/api-reference/get-search-tweets>

#### Python

To perform sentiment analysis on the tweets provided by the Twitter Standard Search API a python script is used. Note that the application is primarily a web application written in JavaScript. To perform sentiment analysis on the tweets within the python script two libraries are imported being textblob and vaderSentiment.

**vaderSentiment**

VADER (Valence Aware Dictionary and sEntiment Reasoner) is a lexicon and rule-based sentiment analysis tool that is specifically attuned to sentiments expressed in social media, and works well on texts from other domains.

**textblob**

Textblob is an open-source python library for processing textual data. Used to convert messages into usable objects for vadarSentiment.

## Use cases

#### US 1

|  |  |
| --- | --- |
| As a | academic data scientist |
| I want | to get data from the site including a pre-processed opinion of the tweet |
| So that | allow for an easier analysis of twitter |

#### US 2

|  |  |
| --- | --- |
| As a | regular twitter user |
| I want | to see a general statement of opinions of a topic |
| So that | I can get that without having to read every single tweet |

*Both the above US cases have implemented these services by creating a query to the twitter api using user provided tags then using that data in a python program that performs sentiment analysis on individual messages then the application stores the results. These results are used to take an average of the polarity of sentiment on all messages provided to determine and then display results to users on screen.*

## Technical breakdown

### Architecture

Diagram

Description automatically generated

Figure 1: Application Data Flow Diagram

The architecture of the application is as seen above in figure 1. This application can receive user input at the URL level an example of this is below in figures 2 and 3. The user will insert the tags they wish to filter twitter messages by going to the applications URL address then inserting there search tags in after as seen in figure 2 then the application will access this information using the req variable.



Figure 2: URL user input example



Figure 3: in code user input example

Once the application has been given an input from a user it will then check an AWS s3 bucket for both twitter messages and the sentiment of those messages. If the requested data is available, then that data will be imported and used to display the results to user’s screens.

If the data is unavailable from the s3 bucket it will then send a request to the twitter API for the required twitter messages for processing. Then to allow the web application to remain responsive if sentiment analysis requires a high amount of time to process the live (or most recent) twitter messages are displayed on screen.

Once messages have been displayed the JavaScript web application will call a python script which has the function of taking in a single message then perform a sentiment analysis of the individual message then returns that messages polarity (positive or negative on a scale from -1 to 1) and its subjectivity (confidence in polarity from 0 to 100%). The JavaScript application will loop through all messages provided calling the python script for all and storing the results in memory. The application will then take messages with a high level of subjectivity then average their polarity to find an answer for twitters publics sentiment for messages with the users inputted tags.

With the results from the sentiment analysis the application will then store the results on the previously mentioned s3 bucket. This data will be stored on the bucket for 60 seconds. This low amount of time is necessary to allow users to retrieve current data while also providing performance benefits for high interest subjects.

After the data is stored on the s3 bucket it will then be used to display the same information on screen for users.

*Diagram

Description automatically generated*

Figure 4: AWS Scaling Diagram

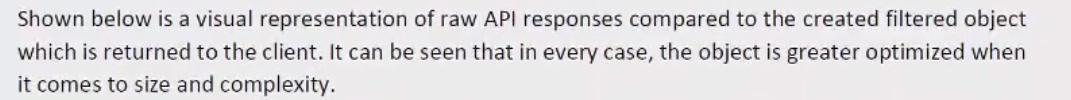
The above image or figure 4 details how the server will increase the computing capacity when specific client requires higher levels then initially provided. As seen in the diagram there will be some number of client web browsers making request to the server these requests will be filtered through a load balancer which will provided a EC2 instance to each individual client the AMI which can be seen in figure 1 will then start processing the request. When an AMI will require more processing power the Auto scalier will then assign more EC2 instances to a AMI when required.

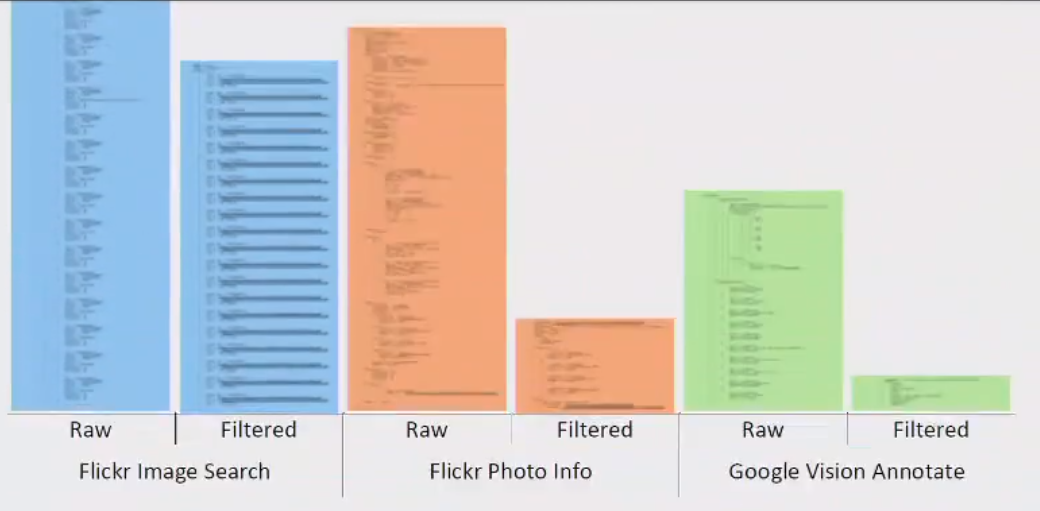
#### Client / server demarcation of responsibilities

The server in this application is performing most of the work on an AWS virtual machine that is created each time a user accesses the URL. This virtual machine will then perform the work required then send the display information or html and style to the user’s system for their browser to handle and then display to them.

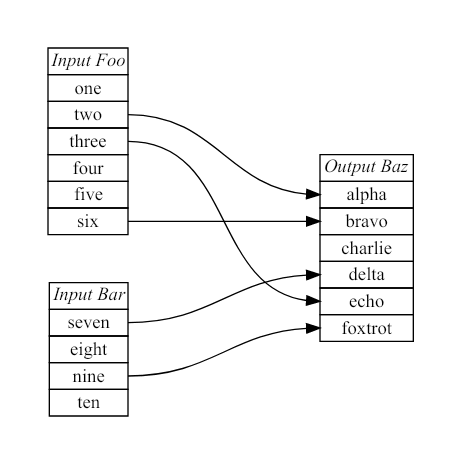
#### Response filtering / data object correlation

*Show us how you manipulated the data. The same comments apply about referring to the diagrams and supporting your work with code fragments as appropriate. We provide an example below of how to show this diagramatically. Please note that this example is quite specific to the system being explained. Yours might have an entirely different look, but do a similar job. This section will vary markedly according to the application and may not make much sense for some custom applications such as rendering.*





*You could instead use a data relationship diagram:*



### Scaling and Performance

*This is a crucial aspect of the report, and you should use this section to document the approach taken to scaling – the nature of the application load, how it was varied and how the scaling infrastructure responded. You should refer to the architectural diagram above or reproduce the relevant aspects here. You should include screenshots of CPU, network or queuing metrics as observed on the cloud services dashboard, together with screenshots of your settings and the scaling pool instance creation and destruction. We expect that your work here will demonstrate successful scale out and scale in as required in the assignment specification. The screen shots that you use here will also very likely be re-purposed as part of the slide deck for the demo.*

*An example scaling image is shown below, and we would normally expect to see this sort of image and some evidence of your group settings. Note the instance count on the left hand axis. As noted, many alternatives are possible.*



### Test plan

*Manual testing is fine and our expectations are in line with the example grid below. You can show the results through a screen shot and point us to these from the table.*

*Your tests should include*

* *Positive outcome cases*
* *Negative outcome cases (error scenarios)*
* *Edge cases*
* *Non-functional cases*

*Note that the grid below is unrelated to this application.*



*As they are common in industry you could define your Acceptance Criteria as GWT statements. This is not compulsory, but see:* [*https://www.agilealliance.org/glossary/gwt/*](https://www.agilealliance.org/glossary/gwt/)*. And here is an example:*



Difficulties / Exclusions / unresolved & persistent errors /

*In this section, you should explain anything that caused you problems and how you overcame those problems. Tell us if there was any issue that prevented you completing the assignment to specification. Tell us about any assumptions or compromises that you have made. Those who worked with an API like Spotify, which presented particular concerns, should discuss the compromises here, and this is also where you can tell us about problems with API keys and responses.*

*More generally, you might consider:*

* *Your major roadblocks and how you resolved them.*
* *Any functionality you didn’t or couldn’t finish*
* *Are there any differences between your brief and what you delivered? If so, explain why.*
* *Are there any outstanding bugs?*

## Extensions (Optional)

*In this section, you can tell us if you wish to how you might extend your app and make it better. This is an opportunity to tell us about good ideas that you had that you didn’t have time to tell us about.*

## User guide

This application is simple to use. For demonstrational purposes let say you want to find out if people are generally against or for vaccines.

To accomplish this you will need the URL of the web application which is as follows.

Insert the web address into a web browser of choices in this case chrome. then follow the URL with

/Vaccine

The full web address can be seen below. Note you may replace /vaccine with any tag or get information on that specific topic.

The URL will send you to the following webpage

## References

*Use a standard approach to referencing – see the guidance at* [*https://www.citewrite.qut.edu.au/cite/*](https://www.citewrite.qut.edu.au/cite/)*.*

## Appendices

*Stuff you want to include, but is too long or too complex to include in the main report text. The full Docker file, some longer excerpt from API docs. Whatever helps.*

*[Our thanks to those students who allowed us to use their work in the examples presented above.]*