WaCC Report: Twitter Sentiment Analysis

Alex Cuello Ortiz (S3540197)
Bogdan Petre (S3480941)
Zhuoyun Kan (S3346935)

Group 16

November 3, 2017

1 Introduction

1.1 Project Description

Sentiment Analysis is the process of determining whether a message on social media is positive, negative or neutral in order to get a general idea of the public opinion on a particular topic. In this project, a web application was developed to collect tweets from Twitter and show the results of certain topics over time-based on the user's interest. Once the user is asked to enter a topic as the keyword in a text field, the application shows all the tweets containing the keyword as well as the analysis results of each tweet. Besides, it also allows the user to get a global view of the topic in real-time by using an interactive interface with chart lines and tables.

analysis results are represented by integer numbers in this project. While +1 indicates a positive attitude, 0 means neutral and -1 represents a negative one. For a certain topic, the sum of all the results will be evaluated as the public opinion. Besides, this application is designed to be scalable, fault tolerant, easy to deploy and user-friendly.

2 Architecture

In this section, the architecture of the project is introduced, which involves the technology stack used in the project and each component with its role as well.

As seen in Figure 1, basically a three-tier architecture was applied during the development of this web application. It is a client-server architecture which is composed of a presentation tier, a domain logic tier, and a data storage tier.

In the presentation tier, a user interface was built to send requests and receive responses from the domain logic tier with the help of REST APIs and WebSockets. The frontend was developed in Angular, which will be discussed in more detail in the next part. The domain logic tier contains sentiment analysis, calculations of results, logic decisions, tweets data processing and so forth. It is represents the backend and it is written in Scala. The data storage tier refers to two databases, namely Cassandra and MongoDB in this project. In order to achieve better scalability of the system, replications and fault tolerance are also implemented with Docker and Google Cloud Platform. This will be introduced in the next sections.

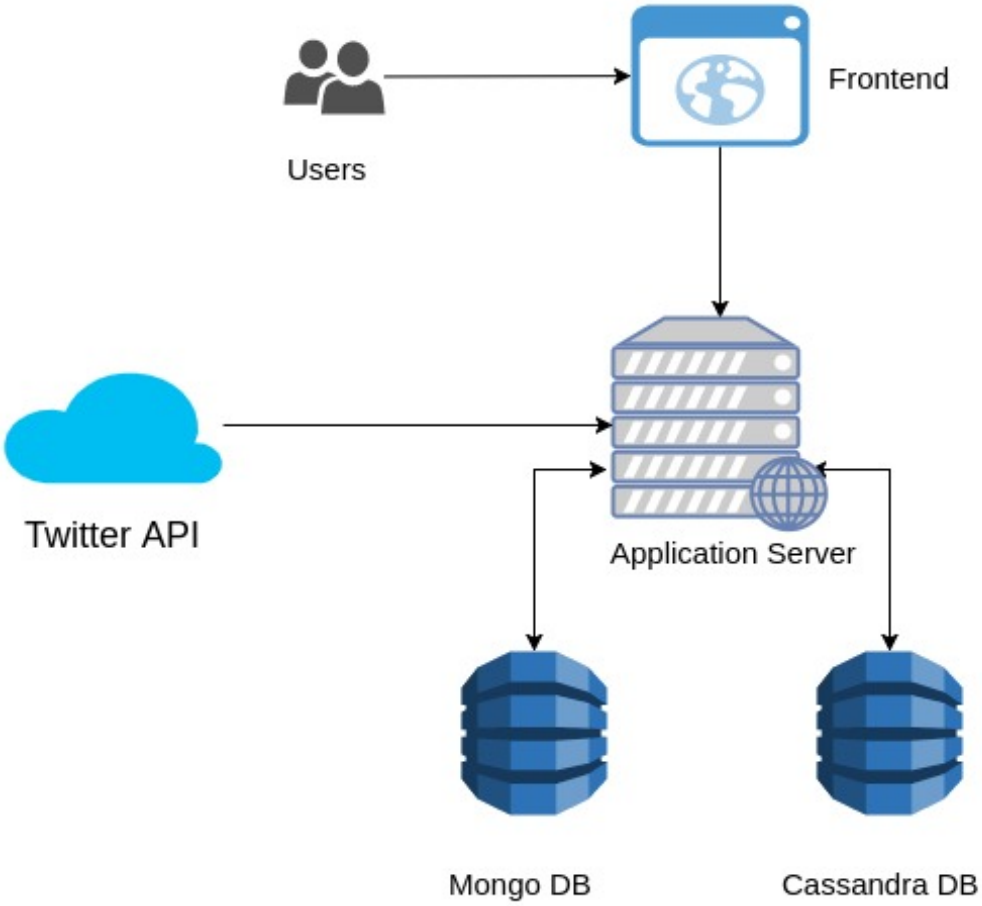

Figure 1: Overview of the application architecture

3 Components

3.1 Frontend

The frontend is built as a single page application written in Typescript, CSS and HTML using Angular 4. The user interface is constructed by sets of different modules with separate components inside, for example, chart-line, tweets-table and tweets-values-counter, which are then group them into a complete page. What should be noticed is that Angular4 has improved updates and new features compared with AngularJS.

In the frontend, we defined a search bar where the user should write down a topic as the keyword. Once the search button is pressed, a REST GET request is sent to the backend. After processing in the backend and dealing with the databases, the results are sent to the frontend and displayed in the tables, counters and the chart-line.

3.2 Backend

The backend was written in Scala using Play Framework. It uses tweets from a CSV file containing all the tweets posted in the US during 2 days, which is treated as real-time tweets received from the Twitter API.

Once the user searches a word in the UI, a request is sent to the backend for the sentiment analysis. After data processing, tweets with several properties are stored in the MongoDB and

the sentiment results are stored in the Cassandra regarding the keyword, timestamps etc. Finally, a response is made and returned to the frontend. The operations performed by the backend are done asynchronously, for the purpose of performance and scalability.

The first time a word is searched, The backend gets the tweets containing that word from the CSV, analyzes them, stores them in Mongo and the results of the analysis in Cassandra and then sends them, together with the results of the analysis to the frontend. Subsequently, when that same word is searched again, the backend gets the results and the tweets from Cassandra and Mongo respectively, and it sends them to the frontend.

3.3 Database

In this project, two different databases are used namely MongoDB and Cassandra. Both of them are NoSQL databases, so they have different ways to organize data compared with traditional relational databases. Generally, NoSQL databases are built to be more scalable and provide high performance with several advantages such as no complex table schema or constraints, automatic index, easier migration and so forth.

Although both of MongoDB and Cassandra are NoSql databases, they provide different data structures, which makes them arranged in a different use. In the following sections, more details are discussed based on the choice of these two databases.

3.3.1 Mongo DB

MongoDB is a document-oriented database with the scalability and flexibility. It stores data in flexible, JSON-like documents, which indicates that fields can vary from document to document and data structure can be changed over time. In this project, MongoDB is used to store the information about the tweets (content, url, user etc.). As tweet data may become detailed and unorganized with more sub-columns and tweets may have replies, which will make the layout of the table change, MongoDB is a good choice to deal with such kind of problems due to its high flexibility.

3.3.2 Cassandra DB

Cassandra is a column-oriented database, which is designed to have peer-to-peer symmetric nodes instead of master or named nodes. This mechanism can ensure that there would never be a single failed point in the database cluster. It is realized by a ring structure with n nodes in Cassandra, where each node is responsible only for 1/n of the data. Therefore, it guarantees the high availability in case of failure and becomes a good choice to manage large amounts of data.

In this project, Cassandra is used to store the analysis results of each topic. The topics (the same as keywords) are set to be the partition key, which means different topics are stored in different partitions. It makes the table easy to manage and the partitions will not grow too much when it comes to some hot topics.

3.4 Nginx

Nginx is a web server that can also be used as a reverse proxy, load balancer or HTTP cache.

In this project, it is used as a reverse proxy. It redirects all the calls to the website's IP address to the frontend. Also, calls to the /api/ are redirected by nginx to port 9000 of the backend, which is exposed.

4 Deployment

4.1 Docker

We used Docker to "containerize" the app. For the purpose of separating responsibilities, we deployed each of the 5 components (Nginx, Frontend, Backend, Mongo, Cassandra) of the app inside its own Docker container. For Mongo and Cassandra, we used the latest Docker images from Docker Hub and left them unchanged. For Nginx, we also used the latest image, but we changed the Dockerfile to use a custom default.conf file.

4.2 Docker-Compose

Docker-Compose is a tool for defining and running apps formed by multiple containers. Inside the docker-compose.yml file, all the services are declared, together with the Docker images, volumes, networks and links between services.

4.3 Google Cloud Platform

We choose Google Cloud Platform (GCP) as the platform to deploy the web application. The platform provides a range of services for compute, storage and application development that run on Google hardware. With high availability and convenience, it is easy to set up and run the web application.

In order to deploy the app on Google Cloud Platform, we used Kubernetes, which is their specific service orchestration tool. However, Kubernetes requires an yaml file in order to create resources. Fortunately, we used a tool called kompose to convert the docker-compose.yml file to yaml files that Kubernetes needed to deploy the app.

Each of the 5 components of the app represents a deployment resource, a service resource and at least one pod resource. Each service has an internal cluster ip, by which it can be accessed by members of the cluster. To be able to access the app from outside the cluster, we declared the nginx service of type LoadBalancer in order to receive an external ip.

Any component can be horizontally scaled by increasing (or decreasing) the number of pods.

5 Results

The final version of the UI with an example result is shown in Figures 2 and 3.

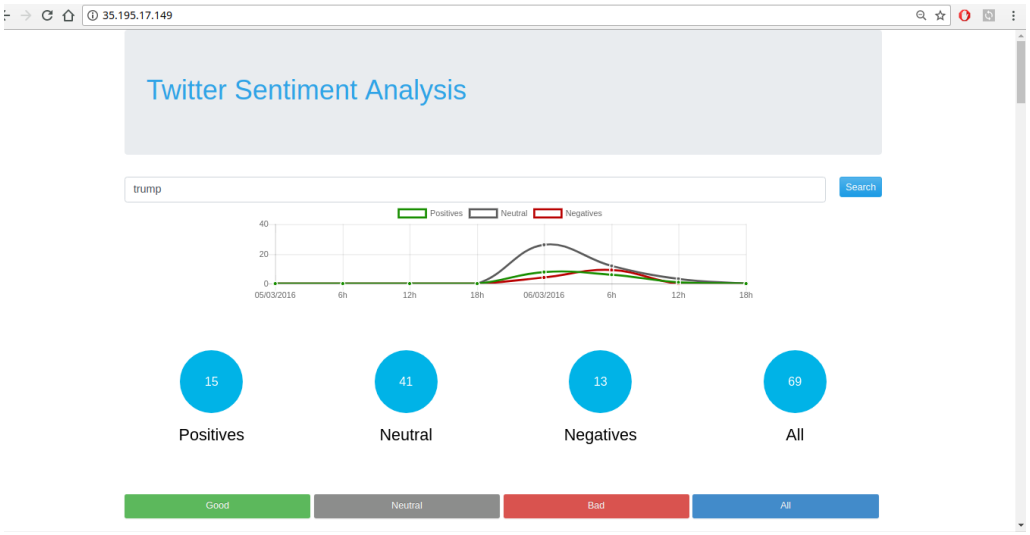

Figure 2: UI tweet analysis

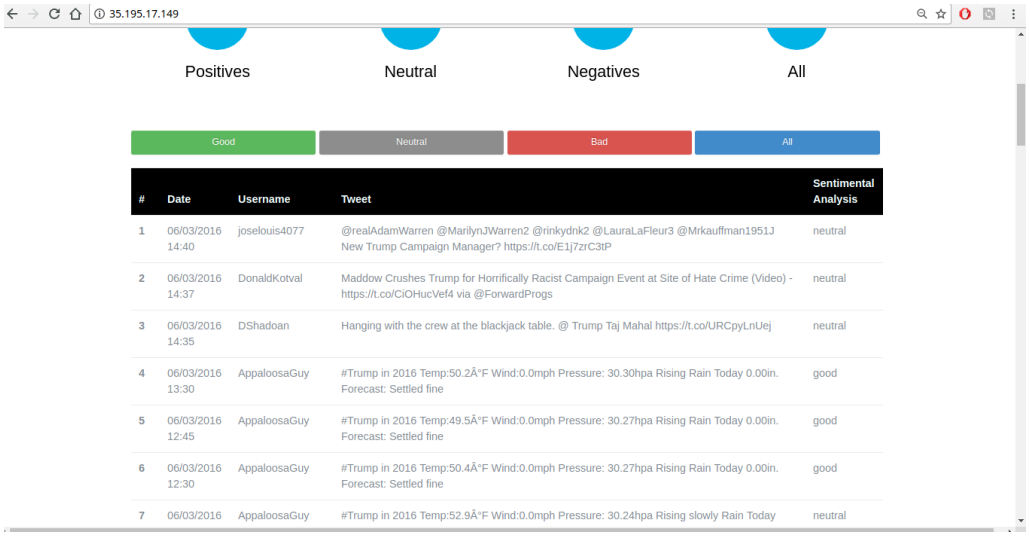

Figure 3: UI tweet list