

**COMP90024 Cluster and Cloud Computing**

**Semester 1 – 2019**

Project 2

**Team 53**

**Team Members:**

|  |  |  |
| --- | --- | --- |
| Student Name | Student Number | Email |
| LINGNA CHEN | 893754 | lingnac@student.unimelb.edu.au |
| YINXUAN LI | 874449 | [yinxuanl@student.unimelb.edu.au](mailto:yinxuanl@student.unimelb.edu.au) |
| DINGHAO YONG | 868878 | [dinghaoy@student.unimelb.edu.au](mailto:dinghaoy@student.unimelb.edu.au) |
| SHUMAO XU | 851442 | [shumaox@student.unimelb.edu.au](mailto:shumaox@student.unimelb.edu.au) |
| SIYI LIU | 710301 | [siyil6@student.unimelb.edu.au](mailto:siyil6@student.unimelb.edu.au) |

# Abstract

This report presents all the works have been done by Team 53 for COMP90024 Cluster and Cloud Computing Assignment 2 building a Sydney Twitter Data Analysis System with Nectar (National eResearch Collaboration Tools and Resources).

**KEY WORDS**: Twitter, Cloud, Nectar, CouchDB, marvel, game of thrones, sentiment analysis, lust, greedy

**External Links:**

GitHub: <https://github.com/yinxuanl/COMP90024>

YouTube:

[Abstract 1](#_Toc8763997)

[1. Introduction 2](#_Toc8763998)

[2. System Functionality 3](#_Toc8763999)

[3. User Guide 3](#_Toc8764000)

[4. System Architecture and Design 5](#_Toc8764001)

[4.1. System Components 5](#_Toc8764002)

[4.2. System Architecture 6](#_Toc8764003)

[4.3. Resource Allocation 7](#_Toc8764004)

[5. Scalability and Deployment Automation Detail 8](#_Toc8764005)

[5.1. Scalability 8](#_Toc8764006)

[5.2. Instance Creation with Ansible 8](#_Toc8764007)

[5.2.1. Issue discussion in interacting with NeCTAR cloud 9](#_Toc8764008)

[5.3. Environment Configuration 10](#_Toc8764009)

[6. Data Processing and Analysis 12](#_Toc8764010)

[6.1. Twitter Harvester 12](#_Toc8764011)

[6.1.1. Data and Processes 12](#_Toc8764012)

[6.1.2. Parameters 12](#_Toc8764013)

[6.1.3. Used Twitter APIs 13](#_Toc8764014)

[6.1.4. Running Scripts 13](#_Toc8764015)

[6.2. Interact with CouchDB: data cleaning and data storage 13](#_Toc8764016)

[6.3. Data Analysis with MapReduce (techniques and examples) 15](#_Toc8764017)

[6.3.1. User perspective: The sentiment changes when users mention Marvel 15](#_Toc8764018)

[6.3.2. User perspective: The sentiment changes when users mention Game of Thrones 17](#_Toc8764019)

[6.3.3. Time perspective: The average sentiment related to Marvel from 2015 to 2019 18](#_Toc8764020)

[6.3.4. Time perspective: The average sentiment related to GOT from January to May 2019 19](#_Toc8764021)

[7. Web Implementation 20](#_Toc8764022)

[8. Security, Fault Tolerance and Error Handling 22](#_Toc8764023)

[8.1. Security 22](#_Toc8764024)

[8.2. Fault Tolerance and Error Handling 23](#_Toc8764025)

[9. Conclusion & Future Improvement 25](#_Toc8764026)

[Reference 26](#_Toc8764027)

# Introduction

This report deploys a cloud-based analytic method to analyze large-volume data which is retrieved from Twitter. Twitter is a large social media platform which has a substantial amount of daily active users. Much useful information can be retrieved from tweets to help with analysis. However, the features of big data such as large-volume, high-velocity, and low-veracity require a well-designed system to collect, process, store, analyze and finally visualize the outcomes. The system with four instances are built on NECTAR Cloud platform, which contains multiple harvesters to harvesting related tweets, clustered CouchDB to store data and a front-end website with RESTful API to visualize analysis outcomes. Dynamic deployment of this system is also explained detailed in this report.

Entertainment in Sydney is mainly focused and citizens' lust and greedy at this aspect is discussed in this project. Movie and television products are one of the main types of entertainment in people's daily life and this report narrow the topic into Marvels and Game of Thrones since these two are hot topics recently. Over 1 million tweets have been gathered to help with analyzing several scenarios under sentiment analysis. Besides, the system is collecting real-time tweets continuously to show up-to-date results. Moreover, this system is scalable by adding more volume in storage and more instances in harvesting so that it has the capability to handle an increasing substantial amount of data.

The team members’ roles are interactive and the main contributions of each member are shown in the table below:

Table ‑ Contributions of each team member.

|  |  |
| --- | --- |
| **Team Member** | **Responsibility** |
| Yinxuan Li | Tweet Harvester, Data Processing |
| Lingna Chen | Architecture Designing, Configuration with Ansible |
| Dinghao Yong | Data Analysis, Web Implementation |
| Siyi Liu | Data Analysis |
| Shumao Xu | Instance Creation with Ansible |

# System Functionality

The system was created to realize MapReduce based implementations for analysing different scenarios existing in the city of Sydney. The system was created to achieve the following functionalities:

* Ansible is used to create instances and install all settings and dependencies automatically.
* The CouchDB database is automatically set up in multiple nodes and connected into a cluster.
* The harvester programs based on Twitter APIs are created to regularly harvest tweets.
* Tweet data is selectively collected from the city of Sydney according to various scenarios and several services are used to run these jobs and to clean and analyse the data.
* Through the python program, the MapReduce functionality is integrated into the system from inside of CouchDB.
* System visualization functionalities are implemented to present research results to users.

# User Guide

* The main repository tree of the project is shown as Figure 3-1.



Figure ‑ The repository tree of the project.

* Download the OpenStack RC file: Login to your UniMelb research Cloud website, download the OpenStack RC file, rename it as “openrc.sh” and put it in the repository “Cloud” (Shown in Figure 3-2).

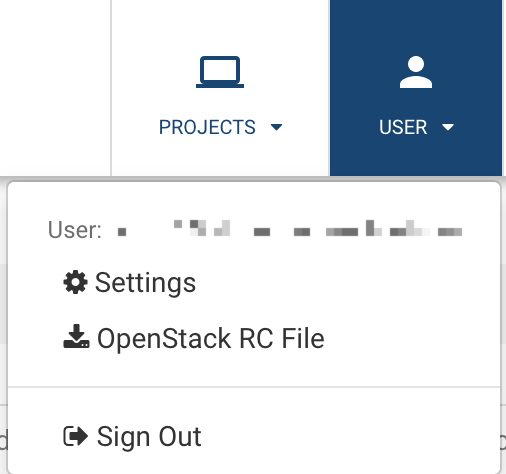


Figure ‑ Download OpenStack RC File from the Cloud website.

* Use Ansible for automation: As Figure 3-3 shown, run the shell script “run-nectar.sh” on the localhost in the “Cloud” repository with the command line “./run-nectar.sh”. Then the Ansible will automatically create instances on the Cloud and finish the environment configuration.

https://lh5.googleusercontent.com/0CwyOQyIAA5JFxZ2NqWBre2WXpdgqSVYADL4wNPi_sB4y7lDDJb0MfQBIr38K_0iW22DG-GJMNaq1Mt2dWN-JZyXbnqeGzsfuTp2zcD2OSwMlRBtnVh4r-Cri_VU9xUM0a2mAkFr

Figure ‑ Command line to use shell script to run Ansible script on localhost.

* SSH into the harvester server and run the Harvester to harvest twitter data: Run the shell script “harvester.sh” in the root with the command line “nohup bash marvel.sh >marvel.out 2>&1 &” (Shown as Figure 3-4).

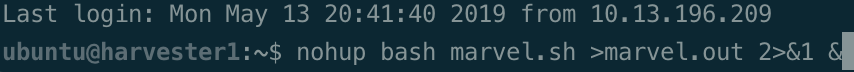


Figure ‑ Run the shell script on one harvester instance.

* SSH into the webserver and Run the shell script “harvester.sh” in the root with the command line shown as Figure 3-5 and Figure 3-6.

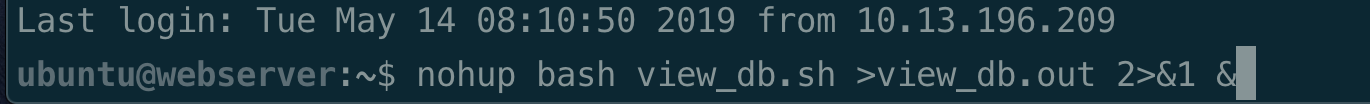


Figure ‑ The first command line to run on Webserver.

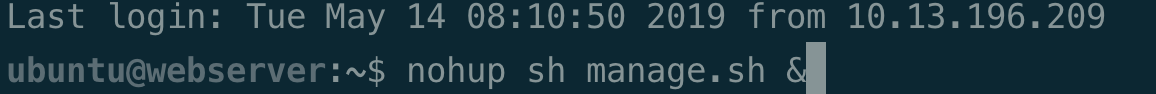


Figure ‑ The second command line to run on Webserver.

* Data visualisation: Open the web browser and go to 172.26.38.202:9100 (Shown as Figure 3-7).

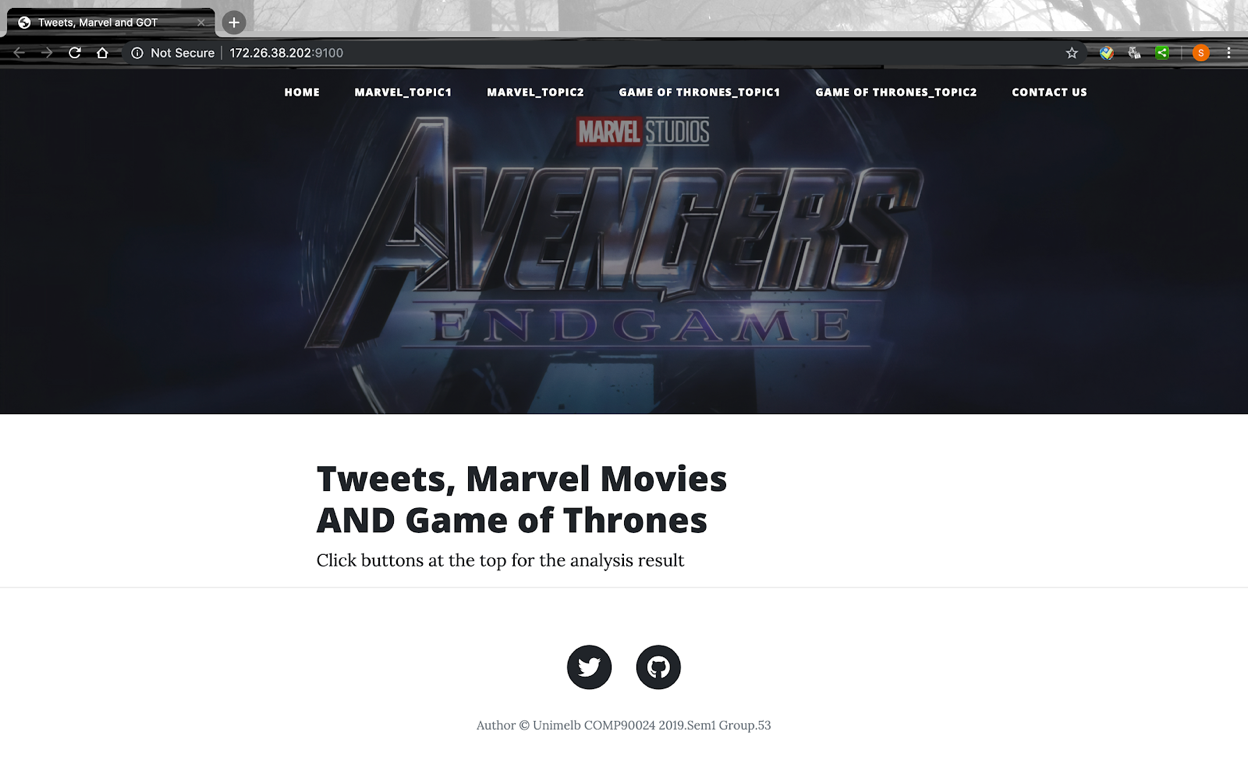


Figure ‑ The website for data visualisation.

* Click buttons at the top for the analysis result.

# System Architecture and Design

## System Components

* Twitter Harvester

In order to produce a convincing data analysis outcome, a substantial amount of data is required. Since Twitter with millions of active users is chosen as a raw data provider, our system should deploy Twitter harvesting function to collect data. Besides, this harvester should pre-process original data and only data related to our topic should be retained. Sentiment analysis is also performed at this step. Moreover, more than one harvester should be considered in handling such a large amount of data to increase processing capacity.

* Database

Considering that data is obtained in a long and continuous time, the database should be equipped with our system to store it firstly. NoSQL-database CouchDB is used to achieve high availability of data, partition-tolerance and eventual consistency. As data is harvested with multiple nodes, a cluster is constructed to manage distributed data. Moreover, the database should respond to requests from the web server to retrieve data at any time.

* Web Server

Web Server should accept requests from multiple clients simultaneously, get requested data from the database and then visualize it to clients. Such processes demand very low response time and thus we should improve the system’s performance with a delicate design. Taking big data into consideration, ad-hoc queries for visualized data is inefficient and a regular update mechanism is applied: Views in DB Server are set to refresh at 12 am every day; Only after views are fully updated, the visualized charts shown to end users that stored in Web Server will be updated. Such process guarantees an acceptable response time at the expense of real-time reflection of data collected.

## System Architecture

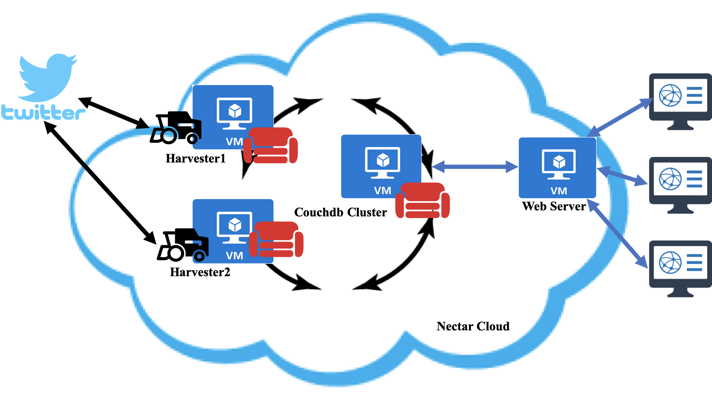


Figure ‑ System Architecture.

With 4 instances provided by Nectar service, these instances are named as Harvester 1, Harvester 2, DB Server and Web Server.

* Harvester 1 and Harvester 2 are same conducting Twitter Harvester and is part of the database cluster.
* DB Server conducts the main role of the database.
* Web Server conducts the role of Web Server.

Three instances (Harvester 1, Harvester 2, DB Server) are connected into a cluster to provide CouchDB service, where one is chosen as the main database server. In addition to the database’s storage function, this DB Server is designed to build views for processed data with MapReduce and respond to requests from Web Server.

The other two instances are also equipped with harvester function, which retrieving data from Twitter, pre-processing it and storing useful one into the system’s database.

The fourth instance is designed to act as Web Server, which is separate to the whole cluster and is the only one to communicate with end users.

## Resource Allocation

Considering the limitation of resources, i.e. 8 compute cores and 250GB of storage capacity, the detailed allocation is shown in Table 1 and the reason for such allocation will be explained in the following part.

Table ‑ Allocation of resources.

|  |  |  |  |
| --- | --- | --- | --- |
| Instance name | VCPUs | Volume size | RAM |
| Harvester 1 | 2 | 60GiB | 9GB |
| Harvester 2 | 2 | 60GiB | 9GB |
| DB Server | 2 | 80GiB | 9GB |
| Web Server | 2 | 50GiB | 9GB |

Considering that the workload in each instance is all heavy as stated in the precious parts, each of them is equally allocated with 2 CPUs and 9GB RAM. Volume with 60GB is attached to harvesters for data storage. A larger volume (i.e. 80GB) is attached to DB Server. The volume attached in Web Server is used to store caches for ready-to-visualized data.

# Scalability and Deployment Automation Detail

## Scalability

Ansible is an IT automation tool that is used to deploy our system. Our system is scalable that more instances acting as harvester can be added when the workload is heavier. Through little changes in Ansible Playbook, a new instance will be opened in Nectar and automatically configured with the ready-to-run environment. Additionally, more volumes provided by Nectar can be attached to our system in case of storage shortage.

## Instance Creation with Ansible

Ansible is an emerging automated operation and maintenance tool. Based on Python development, it combines the advantages of many operation and maintenance tools (puppet, cfengine, chef, func, fabric), and implements functions such as batch system configuration, batch program deployment, and batch run commands.

In this project, NeCTAR, Australian public cloud services are used to implement our system. NeCTAR is funded by the Australian government to provide free and flexible computing power to all Australian researchers, which include computing infrastructure, software, and services that allow the research community to store, access and run data, remotely, rapidly and autonomously. Before interacting with the NeCTAR, the OpenStack-RC file need to be acquired which could be considered as the configuration file for the NeCTAR API. In the Instances Creation step, the script run-nectar.sh is the only file needs to be run with the following command line: sh run-nectar.sh where 8 Ansible roles are specified as Figure 5-1.

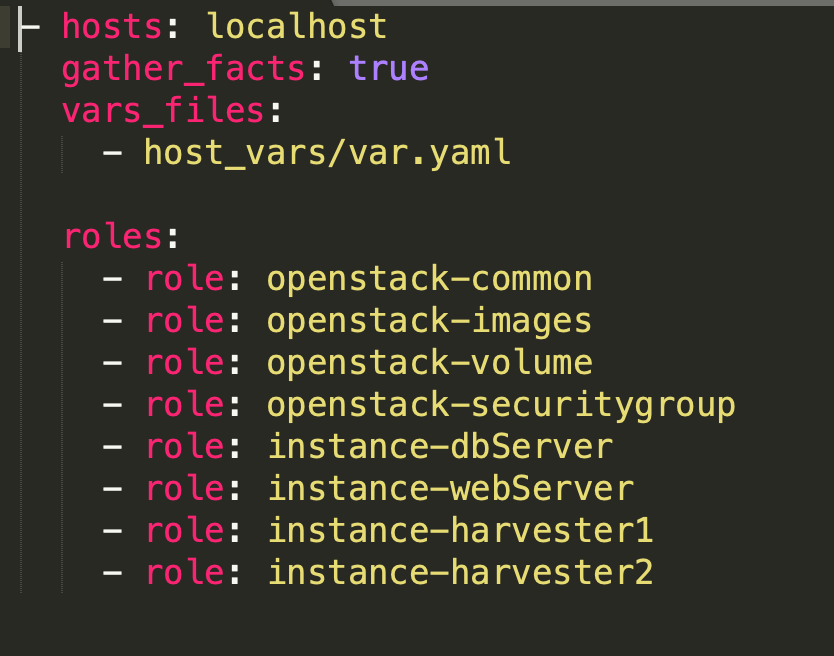


Figure ‑ Eight roles for creating four instances on NeCTAR.

* *OpenStack-Common*: Install pip and OpenStack into the instances;
* *OpenStack-images*: Show all available OpenStack images;
* *OpenStack-securitygroup*: Create security group rules for some ports, including port 80, 443, 20, 5986, 4369, 5984, 9100-9200. It also creates special security group rules for ICMP protocol, internal-ICMP protocol, internal-TCP protocol, internal-UDP protocol;
* *OpenStack-volume*: Create volumes. The volumes’ names and sizes are defined in the host\_vars/var.yaml.
* *Instance-dbServer*: Create an instance for CouchDB Server which will provide the primary CouchDB.
* *Instance-harvester1*: Create an instance for the first harvester server. A part of the CouchDB cluster will also be on deployed on it.
* *Instance-harverster2*: Create an instance for the second harvester server. A part of the CouchDB cluster will also be deployed on it.
* *Instance-webServer*: Create an instance for the web server.

### Issue discussion in interacting with NeCTAR cloud

There were several issues and challenges when interacting with the NeCTAR.

* Image ID changes: When using ansible to automatically create the instance, error may occur with the image ID. It seems that the image ID of a certain image could be periodically changed due to the security reason. In this case, before running the ansible script, it needs to be confirmed that the image ID matches a certain image at that moment. If the image ID was changed, the easiest solution to this is to manually create an instance at that moment, choose the expected image and copy the image ID to the code.
* Mount the volume: After creating the instances and attach the corresponding volumes to those instances, it is important to point out that those volumes need to be formatted and be mounted. The mount command of the Unix command line tells the operating system that the corresponding file system is ready to be used, and the file system will correspond to a specific point (called a mount point). Mounted files, directories, devices, and special files are available to users.
* Newly created instances are unreachable: After creating the instances, those instances may be temporarily unreachable for a short time which means the users could not log in to the instances via SSH command. The simple solution to this situation is to wait for a while before trying to login to the instances. Therefore, as shown in Figure 5-2, a *sleep* command is added in the script in order to avoid the unreachable problem.



Figure ‑ Using sleep command to avoid the unreachable problem for newly created instances.

## Environment Configuration

After instances are created, environments of these systems can be easily configured by running the configuration playbook ***config.yml*** with host IP addresses and unique accessed private key stated in ***hosts.ini***.

Command to run configuration playbook:

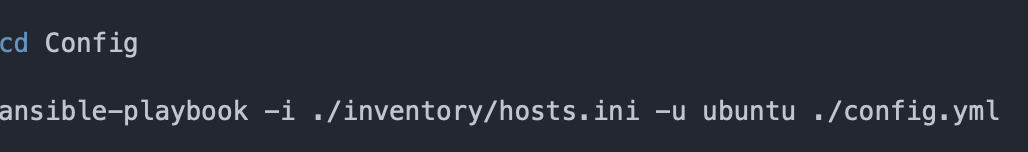
****

Figure ‑ Command to run configuration playbook.

Hosts and the related private key statement:



Figure ‑ Hosts and the related private key statement.

Besides, the configuration of host\_key\_checking in ansible is changed to False (Figure 5-5) to avoid warning at first connection to hosts or we should enter “yes” in ssh connection manually.



Figure ‑ Change the configuration of host\_key\_checking in Ansible to False.

Although there are 3 different types of roles among 4 instances, some allocations are all required such as volume attachment, http\_proxy settings and common packages installation including python.

Then, Docker and a CouchDB docker container will be installed in three database servers. Besides, Docker is mounted to settled volume to avoid the following problems:

* limited storage volume in the instance itself;
* in case of any unexpected shuttle down of instance.

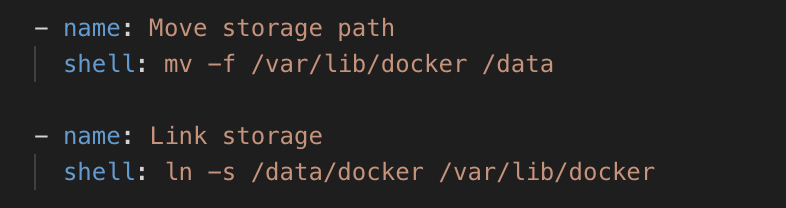


Figure ‑ Mount Docker storage path.

CouchDB can be deployed quickly by pulling the image from Docker and setting related parameters in creating the container (Figure 5-7).

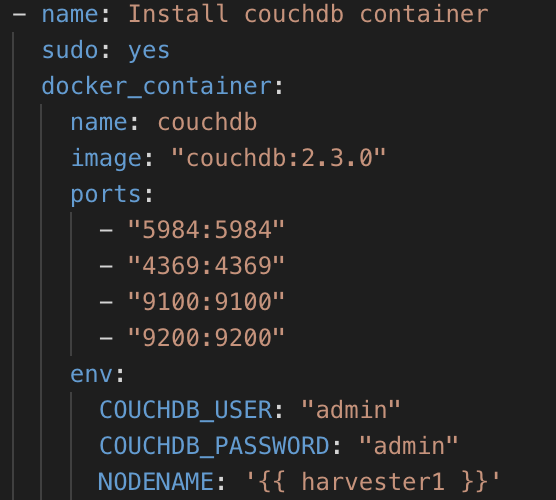


Figure ‑ Related parameters in creating the container.

After that, the cluster configuration code will be executed in dbServer to set up a cluster.

For 2 harvesters, some other special packages are installed to run our harvesting python code: tweepy, textblob, and CouchDB.

For Web Server, a high-level Python web framework Django is installed to support visualization of web application. The package pyecharts is needed to produce summarized charts. Besides, couchdb package is also used to perform MapReduce requests to DB Server.

In completion of these steps, all packages are installed in these instances and ready for all applications to run.

# Data Processing and Analysis

Our system collects data from the Twitter platform by accessing the Twitter APIs and uses the CouchDB for data storage and further analysis. In this section, we will first introduce the design and architecture of the Twitter harvester. Then, we will discuss the interaction with CouchDB. Finally, we will discuss the analytic tools we used and provide some example analysis results.

## Twitter Harvester

### Data and Processes

The data we collected from Twitter is consists of three parts, the tweet object that contains given keywords and is posted within a giving bounding box, the user object that has posted target tweets and the status object posted by the particular user with a given restriction.

Our harvester can continuously collect real-time post tweets while running at the cloud server. When a tweet is crawled, firstly, we will use the data cleaning steps to filter target attributes and store it to a related database; then, we will store the information of the tweet’s author and use his/her ‘screen\_name’ as the parameter to collect the status objects. While a status object is collected, we will use the regular expression to check whether it contains any of the given keywords and store it to the tweet objects collections if it does.

### Parameters

Below are the parameters we used while running a Twitter harvester:

Table ‑ Parameters for a Twitter Harvester.

|  |  |
| --- | --- |
| Location Bounding Box (Sydney) | [150.8, -34, 151.3, -33.6] |
| The total limit for a user’s statuses | 2000 |
| Example queries for topic ‘marvel’ | ‘marvel’, ‘marvelcomincs’, ‘avengers’ |
| Example queries for topic ‘Game of Thrones’ | ‘game of thrones’ |

### Used Twitter APIs

Our system uses the ‘POST statuses/filter’ to collecting the real-time post tweets, and the ‘GET statuses/user\_timeline’ to collect the status objects post by a target user. The ‘cursoring’ technique provided by Twitter’s REST API is also used to paginate large result sets from the ‘GET statuses/user\_timeline’ (//todo: reference). Moreover, in order to collect more data for analysis, we used the standard search API to crawl the last 7 days’ historical data as well.

### Running Scripts

Our harvester could be invoked by a script file and run at the server back-end. An example script (‘marvel.sh’) we used to run our Twitter harvester is shown as below:



Figure ‑ Example script to run a Twitter harvester.

* ‘-q’: indicates the queries to filter real-time tweets, a maximum of 400 keywords can be applied.
* ‘-d’: indicates the database name to store tweet objects.
* ‘-t’: indicates the database name to store status objects.
* ‘-u’: indicates the database name to store user objects.

## Interact with CouchDB: data cleaning and data storage

Twitter provides a range of rich information of tweet objects and user objects while it contains many redundant attributes which will not be used in our system as well. Therefore, we will pre-process the data before storing it into the database. The examples of cleaned data are shown as below:

* A tweet object:

In order to deal with recurring tweets, we set the ‘id’ of a tweet, which is generated by Twitter, as the ‘\_id’ parameter when storing into the database. Since the CouchDB uses the ‘\_id’ as the only identification for each item, this will avoid the problem of the conflict with two identical tweets.

As for the text attribute, since Twitter allows 140 - 280 characters per tweet, we stored the ‘full\_text’ value from the original tweet object in case of a tweet with text longer than 140 characters.

The sentiment score and subjectivity score are analysed by using python’s textblob module. The sentiment score is range from -1 to 1 and a negative score refers to negative sentiment polarity while a positive score refers to the opposite.

The created time information of a tweet is separated into ‘week’, ‘month’, ‘day’, ‘time’ and ‘year’ which could be helpful for further efficient analysis.

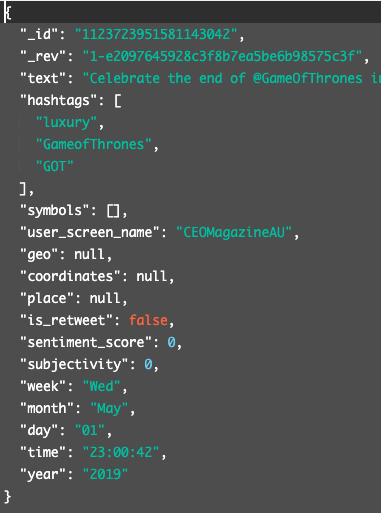


Figure ‑ Example tweet object.

* A user object:

In a user object, the user’s ‘screen\_name’ is selected as the ‘\_id’ in order to deal with the problem of conflicts. The user’s self-defined location and description are also stored for further analysis.

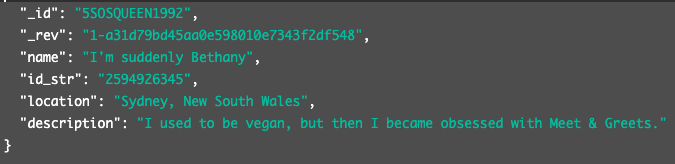


Figure ‑ Example user object.

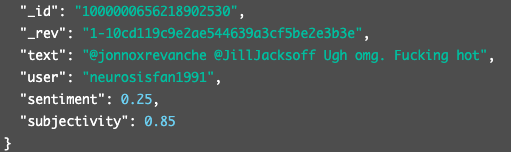
* A status object:

Figure ‑ Example status obejct.

## Data Analysis with MapReduce (techniques and examples)

Our system uses CouchDB's built-in MapReduce to perform the data analytics, and the analysis results are visualized at the web server.

### User perspective: The sentiment changes when users mention Marvel

This scenario is to analyze the sentiment changes in the tweet from the users’ perspective. Marvel movies are popular all over the world. According to the "Sydney Morning Herald", Marvel's latest superhero movie Endgame has become the highest-grossing movie in Australian history. More than 95% of the audience praised the film (Carmody, B., 2019). This has caused us curiosity, what is the relationship between the tweets of Sydney audience post and lust? Tweets related to lust include strong love for certain things and sentiment changes (Seven Deadly Sins, 2019). First, the collected tweet data are cleaned in order to eliminate meaningless (such as blank, irrelevant and etc.) and repeated tweets. At the same time, sentiment analysis is conducted on the tweet data after cleaning. The sentiment scores of users’ tweet related to Marvel in Sydney are compared with their previous tweets’ sentiment scores. The difference in statistics and visualization of these two scores are in figure 6.3.1.

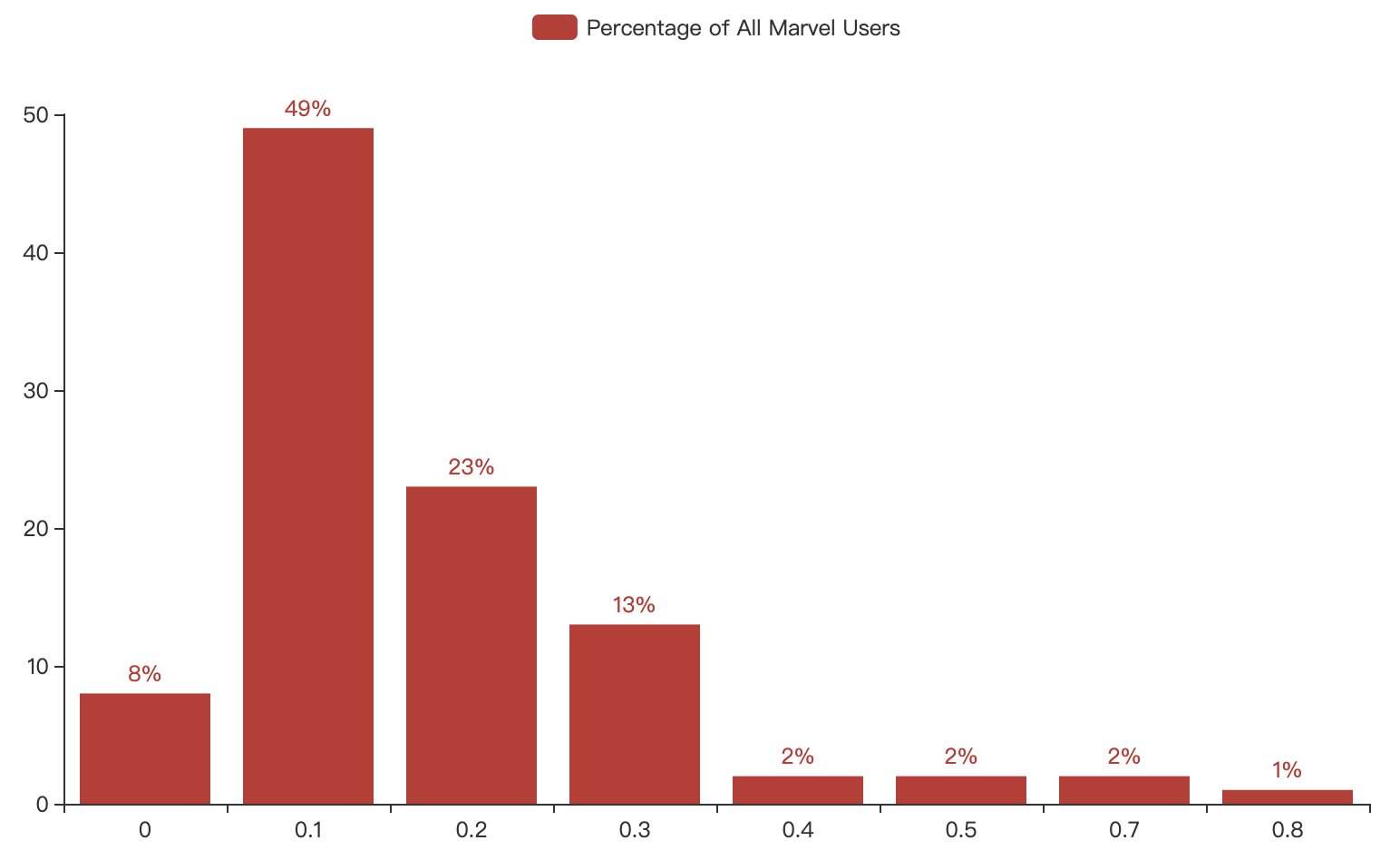


Figure ‑ Sentiment Changes When Mention Marvel Related Content.

According to the above figure, it can be observed that 49% of Marvel key users have changed the sentiment scores by 0.1 when sending Twitter about Marvel contents. Meanwhile, 23% and 13% of user sentiment scores changed by 0.2 and 0.3 respectively. However, the change from 0.4 to 0.8 is relatively tiny, usually around 1% to 2%. One of the interesting points is that 8% of the users’ sentiment scores have not changed. These data reflect the fact that most of Marvel's key users generate both positive and negative emotion fluctuations when sending tweets. This represents that more than 84% of users’ tweets have a low-level (0.1-0.2) emotional fluctuation, and the proportion of users with severe emotional changes is not high. Besides, there are about 8% of user sentiment scores that have not changed, which is beyond our expectation. The reason for this phenomenon is the limitation of sentiment analysis. More details will be discussed in the section *detail the limitations of mining twitter content and language processing*.

### User perspective: The sentiment changes when users mention Game of Thrones

This scenario is based on different content (GOT and Marvel) to compare the sentiment changes in the tweet from the perspective of the users. The Game of Throne related content is very enthusiastic on the internet. A survey of users showed that 48% of respondents watched “Game of Throne”, and even those who did not watch 100% of them knew that someone had done so (Numerator.com, 2019). Although the sentiment of the Marvel audience has been analysed, we also want to know the ‘*Lust*’ attitude of the audience to the Game of Throne. Thus, comparison of the sentiment changes in the Game of Throne and the Marvel movies can be conducted. The figure 6-6 shows results of sentiment difference when mention Game of Throne related Content.

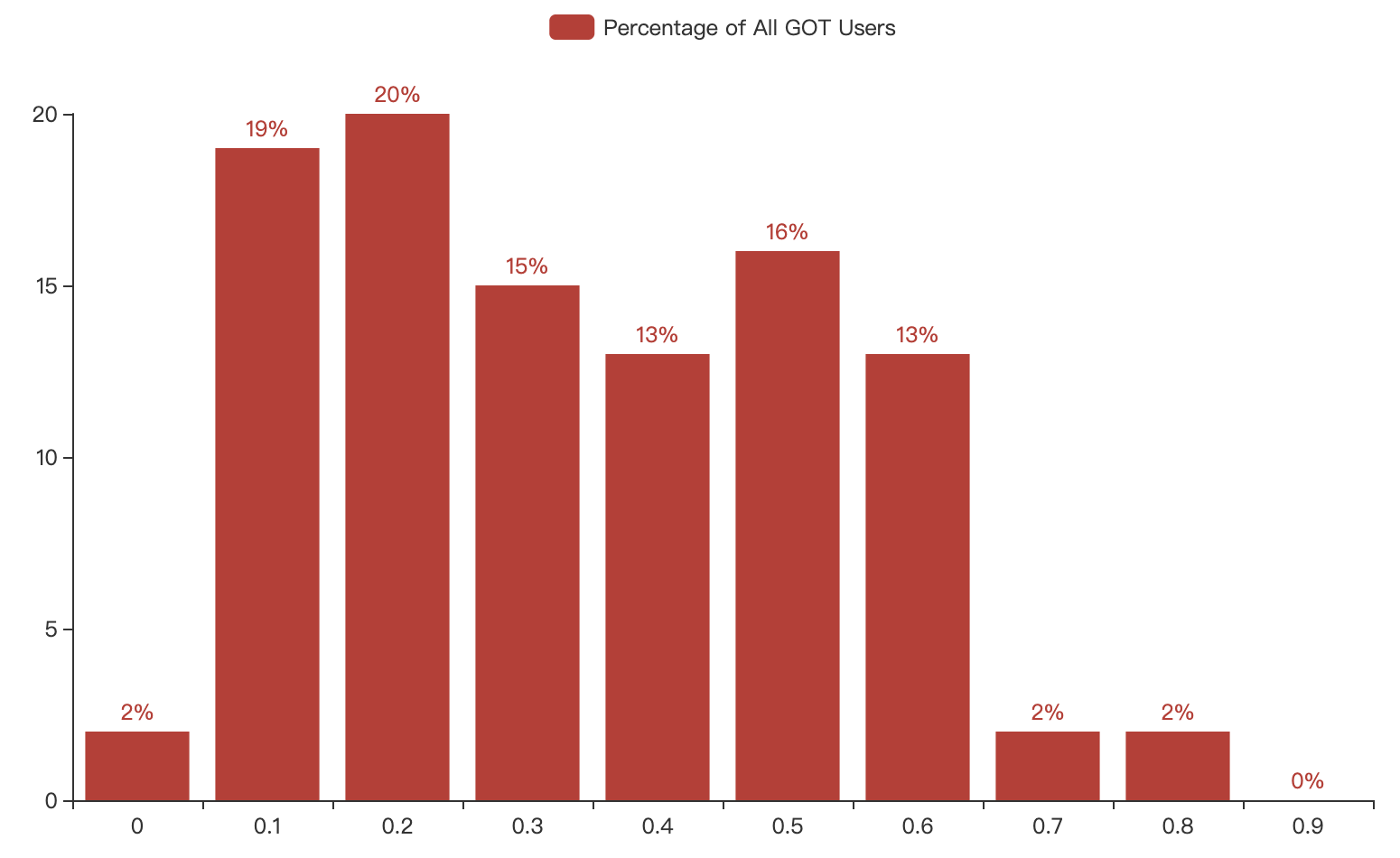


Figure ‑ Sentiment Changes When Mention GOT Related Content.

According to the figure 6-6, it can be seen that only 2% of users have no change in sentiment. This means that the majority of Sydney users have emotional fluctuation for Game of Thrones. Meanwhile, most of the fluctuations are concentrated in the interval of 0.1-0.6, and the number of users whose sentiment score changes by 0.2 is highest accounting for 20% of total. Other scores change around 15%. Compared with the results of Marvel, the audiences of Game of Throne have a more even distribution of emotion changes, and the percentage of users in the 0.5-0.6 range is significantly higher than that of Marvel users. As a result, the GOT user's emotion is more volatile than that of Marvel users when they are discussing related content.

### Time perspective: The average sentiment related to Marvel from 2015 to 2019

This scenario is to analyze the average sentiment scores of users over the years from time perspective. Marvel movie film scores from 2015 to 2019 continue to rise (IMDb, 2019). We want to know if the *greedy* desire of the audience has been met as the quality and rating of the film continue to improve. The figure 6-7 shows that average sentiment related to Marvel.

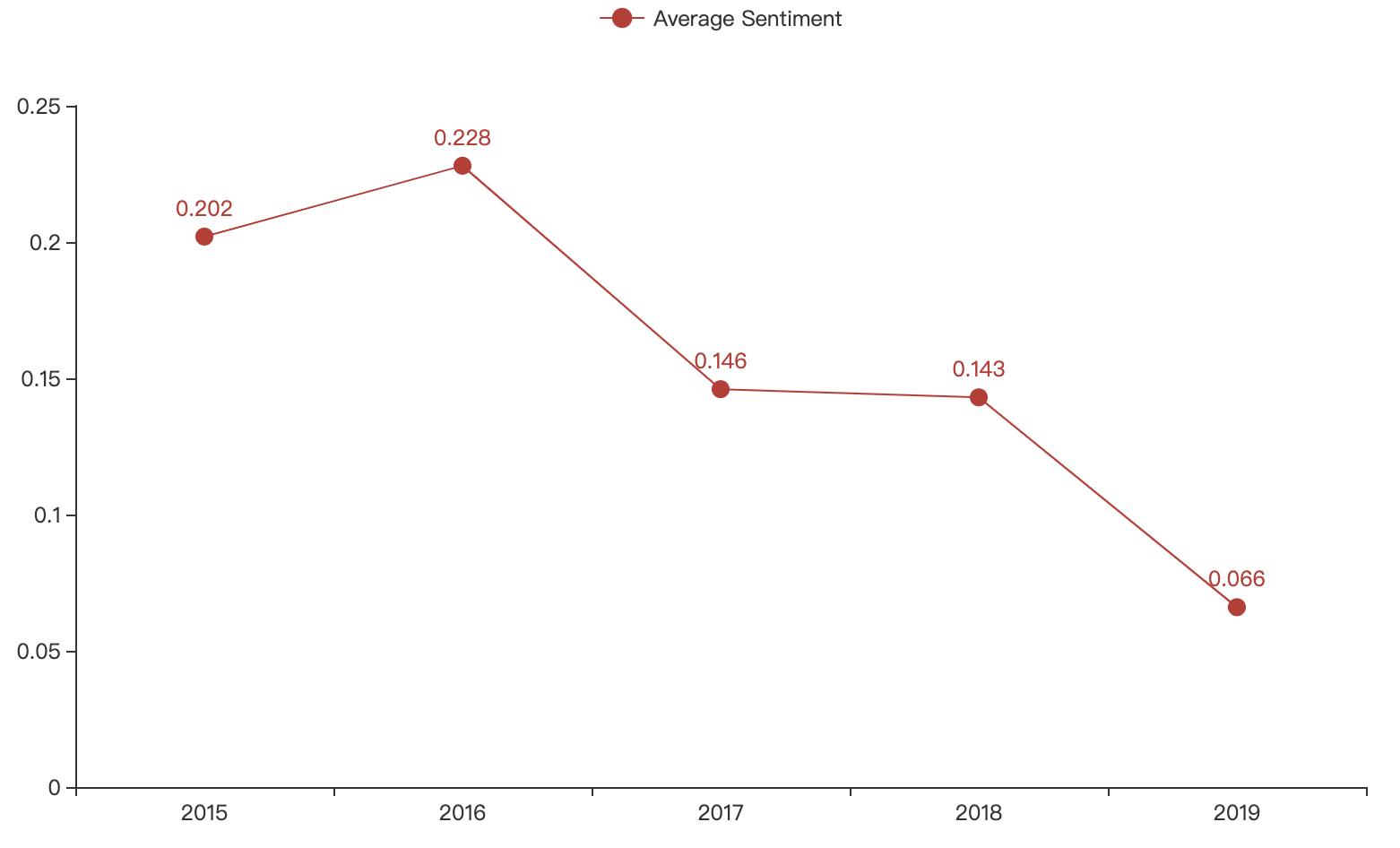


Figure ‑ Average Sentiment Related To Marvel Movies in Recent 5 Years.

Figure 6-7 shows an interesting trend, except for the average user sentiment scores from 2015 to 2016, the 2017 and 2018 scores have dropped significantly from the previous two years. Meanwhile, by 2019, the the lowest average sentiment score is 0.066. This trend indicates that the average average sentiment scores of users for Marvel content is decreasing year by year. In other words, Marvel users are increasingly pleasing over time. In connection with the seven sins, the user’s “greed” for Marvel’s content is getting bigger and bigger.

### Time perspective: The average sentiment related to GOT from January to May 2019

This scenario is to analyze the average sentiment scores of users over the months from time perspective. A survey showed that after the final season of Game of the Throne was released, 46% of the audience said they actually purchased the power of the episode brand products (Looker, 2019). Thus, with the release of the episode, is the monthly "*greedy*" sentiment of users on Twitter being met? How is the emotional fluctuation before and after the release? The figure 6-8 shows result of Average Sentiment Related to Game of Thrones monthly.

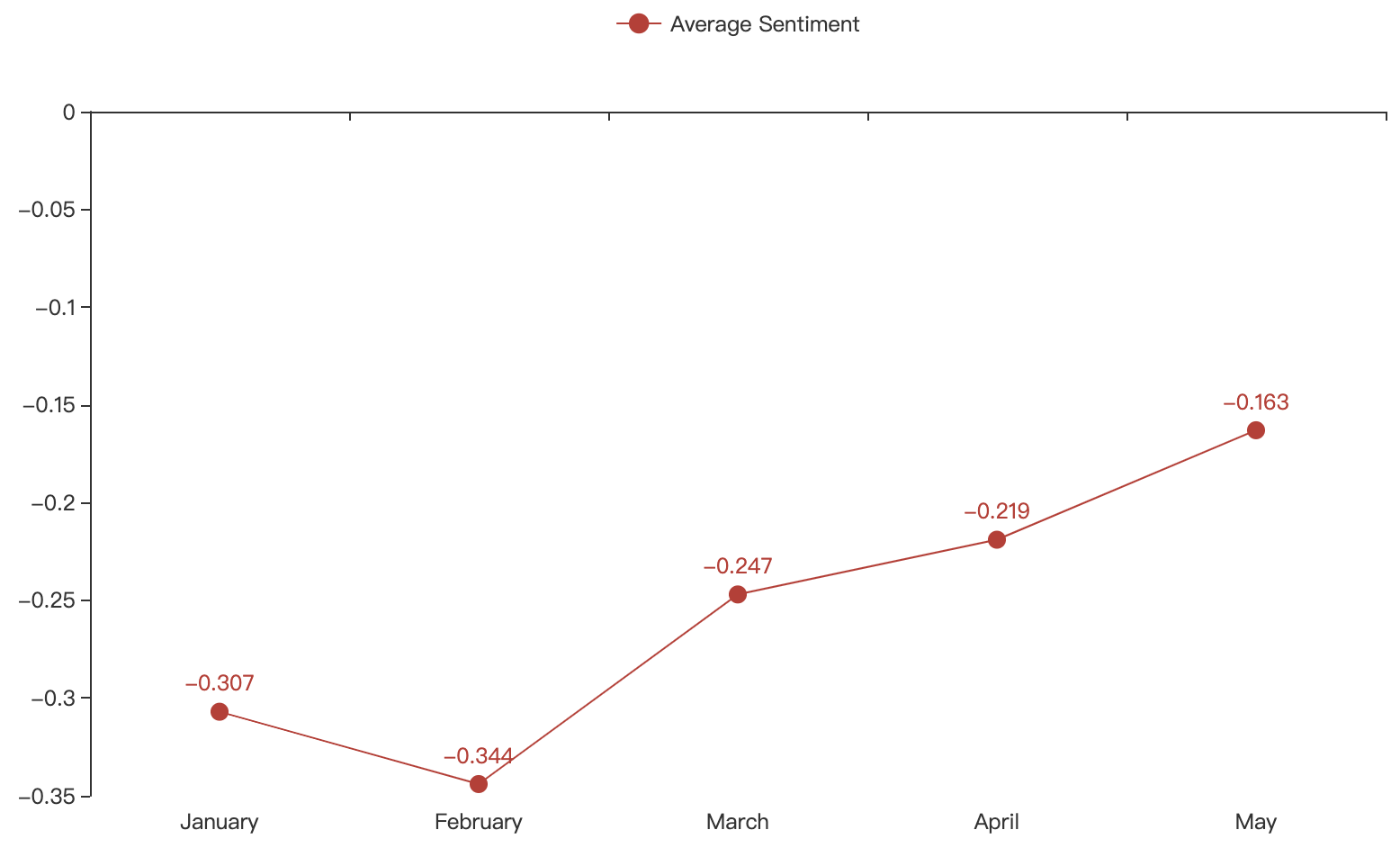


Figure ‑ Average Sentiment Related To Game of Thrones in Recent 5 Months.

From January to February 2019, the average sentiment score of Twitter users decreased from -0.307 to -0.344, and from February to May, the scores showed an upward trend. The highest score appeared in the second month of the episode, which was -0.163. However, one of the interesting phenomenons is that the overall GOT related users’ emotions are negative on Twitter. This may be because the quality of the episode is declining and the content of the episode is too tragic (Winter is Coming, 2016). Another possible reason is the large amount of dissatisfaction accumulated by the users during the waiting time, and the greedy negative emotions expand with the waiting time.

# Web Implementation

In this section, we will illustrate the web application design in the instance named webServer. In general, we use Django to build the web application in our implementation. Django is a free and open source web framework, which is suitable for rapid development and clean, pragmatic design. It takes care of much of the hassle of web development, so developers can focus on writing their applications without needing to reinvent the wheel (Django Software Foundation, 2019).

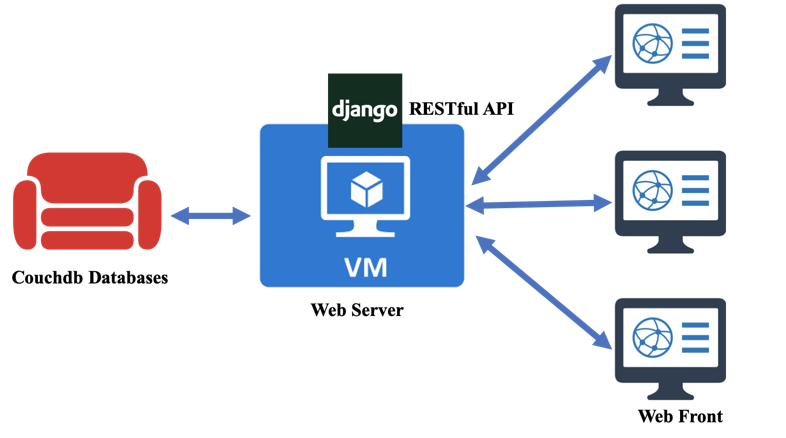


Figure ‑ Web Application Design.

In the Django web framework, what can be seen in a web page is the view of a template. These templates can extend some base templates. In our implementation, we use some same headers and footers in different pages with this operation.



Figure ‑ Base Header 1.



Figure ‑ Base Header 2.

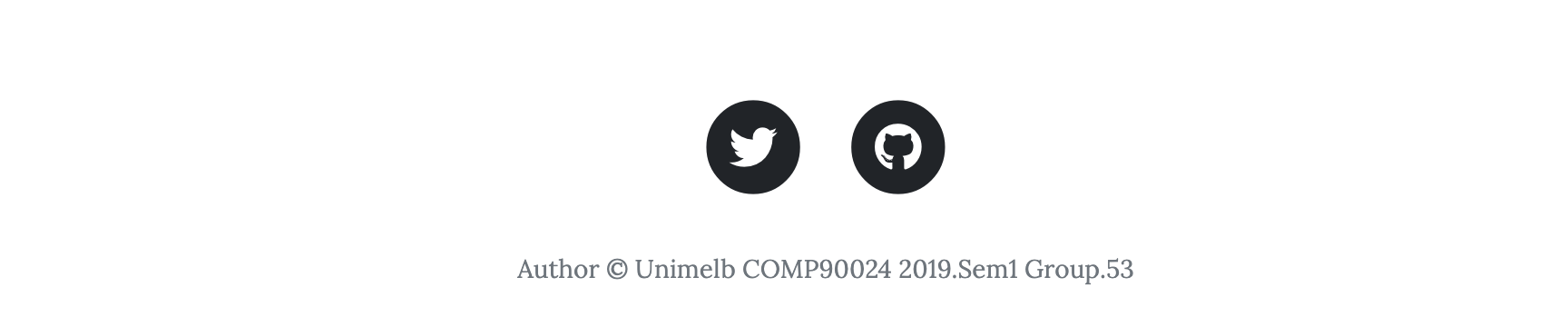


Figure ‑ Base Footer.

As required, we should build **RESTful API**s for the web pages in this assignment. The REST acronym stands for Representational State Transfer. In a sense that the RESTful API simply makes the information you have stored available using a common format. This way, an external application can interact with your application and your data, without having to connect directly into your database. To achieve this, we set the following urls in our Django application for getting access to template views and data visualization results.

****

Figure ‑ Urlpatterns in url.py.

As the data are dynamically stored into the couchdb databases by the harvesters, views in DB Server are set to refresh every 24 hours. To perform this, a python script in the web server will automatically run at the same time everyday to update the views of the couchdb databases in the dbServer. It deletes the existing views and create new ones. This script will also get the new views from the databases immediately after it updates them, and then it will execute the data visualization process.

As for the data visualization method, we make use of *pyecharts*. It is a python library for generating Echarts charts. [Echarts](https://github.com/ecomfe/echarts) is an easy-to-use, highly interactive and highly performant javascript visualization library under Apache license. *Pyecharts* can produce charts with the html format and save them into the server. To show the analysis results on the web pages, we use <iframe> tags to make these html charts as embedded html pages in our Django template views.

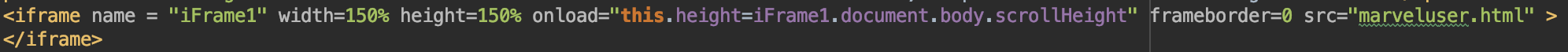


Figure ‑ Example code to create the HTML page.

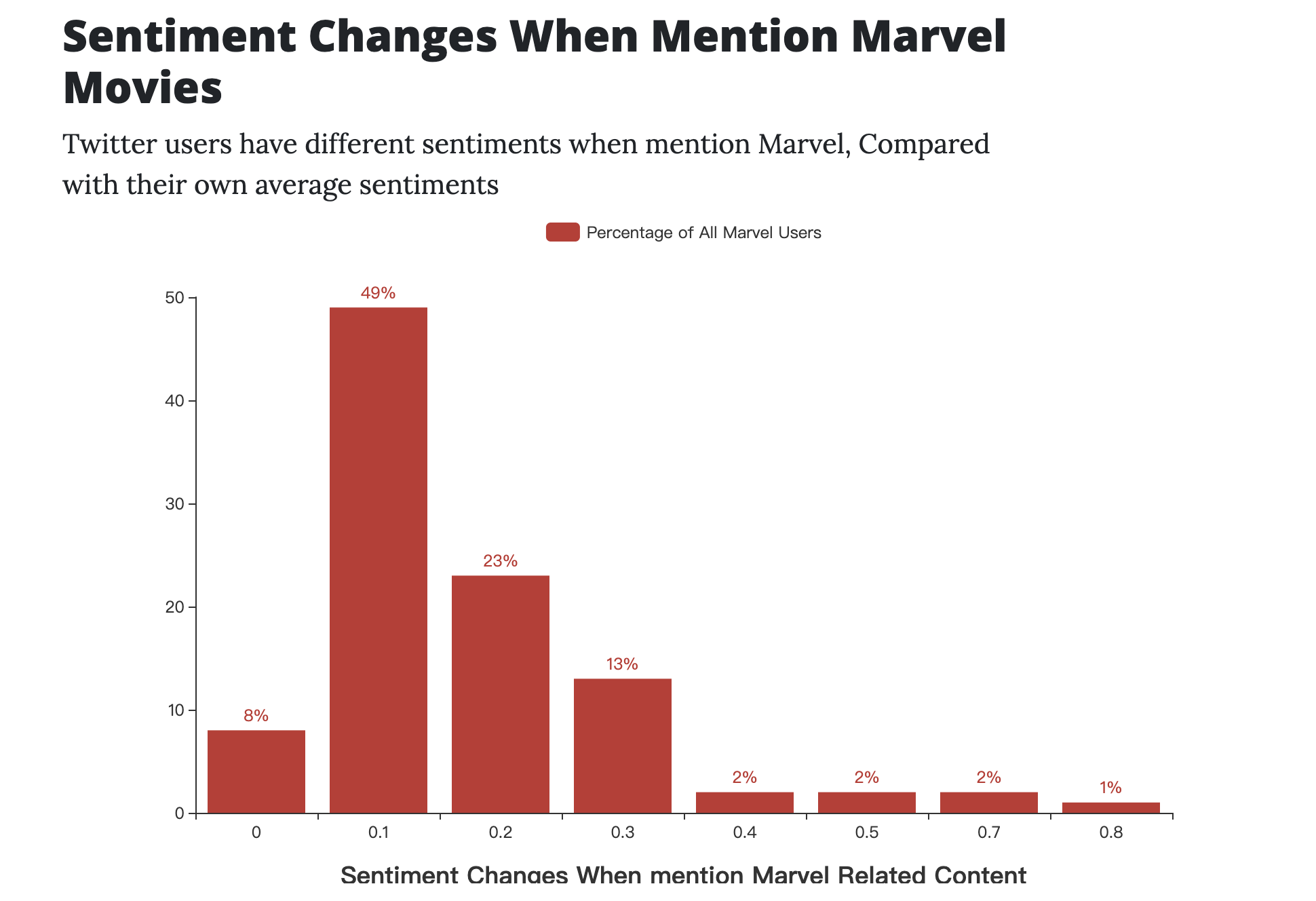


Figure ‑ An Example of Embedded HTML.

# Security, Fault Tolerance and Error Handling

## Security

The security of our system mainly relies on the built-in security feature of the nectar service. One can only access our instances under the network environment of UOM LAN. Besides, a unique private key is required for connecting to each instance. Moreover, a security group is attached to each instance and only several ports (5984, 4963, 9100-9200) are open to restricting communication among themselves. These CouchDB ports are not open to the public and the only one can connect to our instance can access to CouchDB.

****

Figure ‑ Example command line to access the CouchDB.

CouchDB is secured by a pair of username and password. In our system, “admin:admin” is just used for ease of coding and testing while such pair should be replaced with a more complex one to improve the system’s capability of security.

## Fault Tolerance and Error Handling

* **Issues and challenges in using the UniMelb Research Cloud:** This has been discussed detaily in part 5.2.
* There is a high probability that you cannot access to instance immediately after its creation with a private key. Hence, we set a sleep time after the instance is created and then do the configuration part.
* In case that connection to the instance is no longer authenticated and we have no other choice but to delete this instance and recreate one, we mount our CouchDB storage path into an attached volume so that we can recover our data without losing.
* **The problem in constructing a cluster of CouchDB.**
* Every node in the CouchDB cluster is available to respond to a client unless itself is down. When connection among the cluster is down occasionally, this cluster will split into several partitions, each of which will act normally. Besides, the isolated partition is keep sending reconnect request to other nodes. Eventually, consistency is also achieved by re-organizing automatically without losing data.
* One advantage of CouchDB is that it supports cluster operation and automatic replicator. In our system, the cluster is using default setting: q=8 and n=3, which means that each database (and secondary index) is split into 8 shards, with 3 replicas per shard. Therefore, there are 24 shard replica files distributed in 3 nodes. Then this system can bear 2 out of 3 nodes shuttled down at any time. Moreover, the active node will share updated data once the died node rejoined the cluster.
* **Limitations of language processing (e.g. sarcasm).**
* The sentiment analysis tools used in this system have certain challenges and limitations.
* First, judging the subjectivity and tone of the tweet is a challenge. The detection and analysis of subjective and objective texts are as important as their tone. In fact, the so-called objective text does not contain explicit emotions. For example, the emotions of the two texts are analyzed: "Marvel movies are great!!!" and “The Marvel movie is hilarious”. Most people think that the first emotion is positive and the second emotion is neutral. All predicates (adjectives, verbs, and certain nouns) should not be considered identical in creating emotions. In the above example, 'great' is more subjective than 'hilarious'.
* Meanwhile, all tweets are sent out at some point in time, in some places, to some people which means that all words are in the context. It is very difficult to analyze emotions without context. However, the machine cannot understand the context if it is not explicitly mentioned. One of the problems that arise from the context is the change in polarity. If we want to consider the partial background of the text, we need a lot of pre-processing or post-processing. However, how to pre-process or post-process data to capture contextual bits that help analyze emotions is not straightforward.
* Furthermore, defining neutrality in sentiment analysis is another challenge. As with all classification problems, marking a neutral category is one of the most important parts of the problem. What does neutral, positive or negative mean when doing emotional analysis. Because tag data requires consistent tagging standards, the problem must be well defined. For example, in tweets texts, hashtags do not have emotion elements, so these hashtags need to be marked as neutral when performing analysis.
* **Limitations of mining twitter content and using Twitter’s APIs.**

Twitter’s API provides a rich range of data and sets of useful functionalities while it still has some limitations.

* Rate limit restrictions: For example, the ‘GET statuses/user\_timeline’ API has a rate limit of 900 requests per 15-min window. In order to avoid the potential disconnection situation affected by rate limit error, we set our Twitter harvester to wait for 180 seconds when a rate limit error occurs.
* Lack of location information: Twitter has a large amount of data for analysing while only a little part of them contain location information, which makes it harder to perform related geographical analytics. This problem could be improved by using natural language processing and machine learning techniques to predict the user’s location with the tweet context.
* Restriction for collecting historical tweet objects: Twitter has a restriction that only the latest 7 days’ historical tweets could be collected by using the standard search API. Therefore, a large amount of data is hard to collect in practice which might influence the analysis results.
* **Deal with duplicates of tweets.**
* In order to deal with the potential problem of tweets conflicts, we decide to use the integer representation of the unique identifier that provided by Twitter API as the ‘\_id’ parameter while storing it into the database, instead of using the automatically generated ‘\_id” by CouchDB. In detail, we use the ‘id’ parameter of the original tweet object and user object provided by Twitter as the ‘\_id’ parameter of the object stored in CouchDB. Since ‘\_id’ and ‘rev’ is CouchDB’s housekeeping and acts as an identification for a particular instance of a document, the problem of tweets conflicts could be avoided.

# Conclusion & Future Improvement

As a summary, this project builds a system to harvest tweets from Sydney and provides suitable visualisation on the data for further analysis. Based on the definition of the seven sins and the tweets data we analyzed, we came up with some interesting stories about Marvel and Game of Throne that happened in Sydney. Besides, this front-end website can be used continuously for end-users since data is streaming into this system in high-velocity and large-volume.

This system deployed in this project is secured, fault-tolerant and scalable to some extent with the help of a cluster of CouchDB to synchronize data between different instances. However, since this is a simple design, there are still multiple areas of the project that could be improved. For example, there is a high probability that instances can be died so we can not guarantee an online website and errors need be managed manually. Besides, data used for analysis is limited due to the limited time of collecting data and the analysis is simple and straightforward.

# Reference

Carmody, B. (2019). *Avengers: Endgame breaks Australian box office records*. [online] The Sydney Morning Herald. Available at: https://www.smh.com.au/entertainment/movies/avengers-endgame-breaks-australian-box-office-records-20190429-p51i9q.html [Accessed 13 May 2019].

Developer.twitter.com. (2019). *GET statuses/user\_timeline*. [online] Available at: https://developer.twitter.com/en/docs/tweets/timelines/api-reference/get-statuses-user\_timeline [Accessed 14 May 2019].

Developer.twitter.com. (2019). *Standard search API*. [online] Available at: https://developer.twitter.com/en/docs/tweets/search/api-reference/get-search-tweets [Accessed 14 May 2019].

Developer.twitter.com. (2019). *POST statuses/filter*. [online] Available at: https://developer.twitter.com/en/docs/tweets/filter-realtime/api-reference/post-statuses-filter [Accessed 14 May 2019].

Djangoproject.com. (2019). *The Web framework for perfectionists with deadlines | Django*. [online] Available at: https://www.djangoproject.com/ [Accessed 14 May 2019].

Docs.couchdb.org. (2019). *2.2. Cluster Set Up — Apache CouchDB® 2.3 Documentation*. [online] Available at: https://docs.couchdb.org/en/master/setup/cluster.html [Accessed 14 May 2019].

IMDb. (2019). *22 Marvel Cinematic Universe Movies As Ranked by IMDb Users*. [online] Available at: https://www.imdb.com/imdbpicks/MCU-movies-ranked-by-imdb/ls038472133/mediaviewer/rm3369838080 [Accessed 13 May 2019].

Looker. (2019). *Data of Thrones Part IV: Were our Season 7 Game of Thrones Predictions Right?*. [online] Available at: https://looker.com/blog/data-of-thrones-part-iv [Accessed 13 May 2019].

Numerator.com. (2019). Insights Are Coming: Winning the Game of Thrones Fan. [online] Available at: https://www.numerator.com/resources/blog/insights-are-coming-winning-game-thrones-fan [Accessed 13 May 2019].

Seven Deadly Sins. (2019). *Seven Deadly Sins*. [online] Available at: http://www.deadlysins.com/ [Accessed 13 May 2019].

Winter is Coming. (2016). *All 67 episode of Game of Thrones, ranked worst to best*. [online] Available at: https://winteriscoming.net/game-of-thrones-episode-rankings/ [Accessed 13 May 2019]