



COMP90024 Cluster and Cloud Computing

Semester 1 – 2019

Assignment 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
DINGHAO YONG	868878	dinghaoy@student.unimelb.edu.au
SHUMAO XU	851442	shumaox@student.unimelb.edu.au
SIYI LIU	710301	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: <https://youtu.be/fD7v0yeqw08>

ABSTRACT	1
1. INTRODUCTION.....	2
2. SYSTEM FUNCTIONALITY	3
3. USER GUIDE	3
4. SYSTEM ARCHITECTURE AND DESIGN	5
4.1. SYSTEM COMPONENTS.....	5
4.2. SYSTEM ARCHITECTURE	6
4.3. RESOURCE ALLOCATION.....	7
5. SCALABILITY AND DEPLOYMENT AUTOMATION DETAIL.....	8
5.1. SCALABILITY	8
5.2. INSTANCE CREATION WITH ANSIBLE.....	8
5.2.1. Issue discussion in interacting with NeCTAR cloud	9
5.3. ENVIRONMENT CONFIGURATION.....	10
6. DATA PROCESSING AND ANALYSIS.....	12
6.1. TWITTER HARVESTER.....	13
6.1.1. Data and Processes.....	13
6.1.2. Parameters	13
6.1.3. Used Twitter APIs.....	13
6.1.4. Running Scripts.....	14
6.2. INTERACT WITH COUCHDB: DATA CLEANING AND DATA STORAGE.....	14
6.3. DATA ANALYSIS AND MAPREDUCE	16
6.3.1. Sentiment change when someone mentions Marvel in his/her tweets	16
6.3.2. Sentiment change when someone mentions Game of Thrones in his/her tweets	17
6.3.3. Sentiment trend related to Marvel (from 2015 to 2019).....	18
6.3.4. Sentiment trend related to the Game of Thrones (Jan 2019 to May 2019)	19
7. WEB IMPLEMENTATION	20
8. SECURITY, FAULT TOLERANCE AND ERROR HANDLING	22
8.1. SECURITY	22
8.2. FAULT TOLERANCE AND ERROR HANDLING	23
9. CONCLUSION & FUTURE IMPROVEMENT	25
REFERENCE.....	26

1. 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 1-1 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

2. 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.

3. User Guide

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

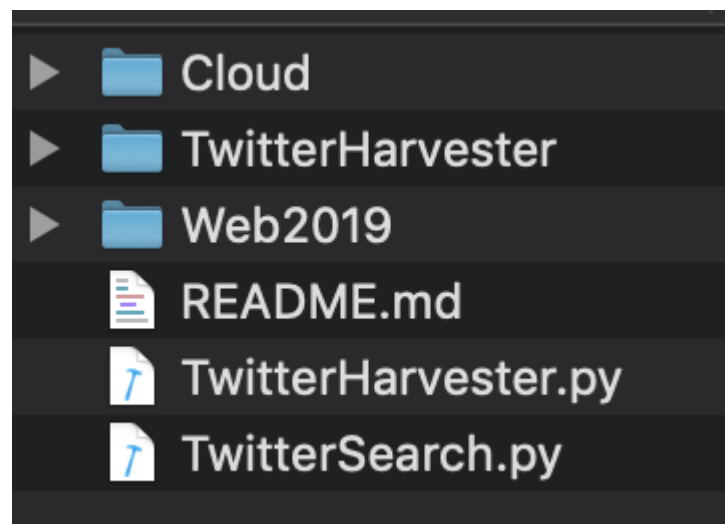


Figure 3-1 The repository tree of the project.

- 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).

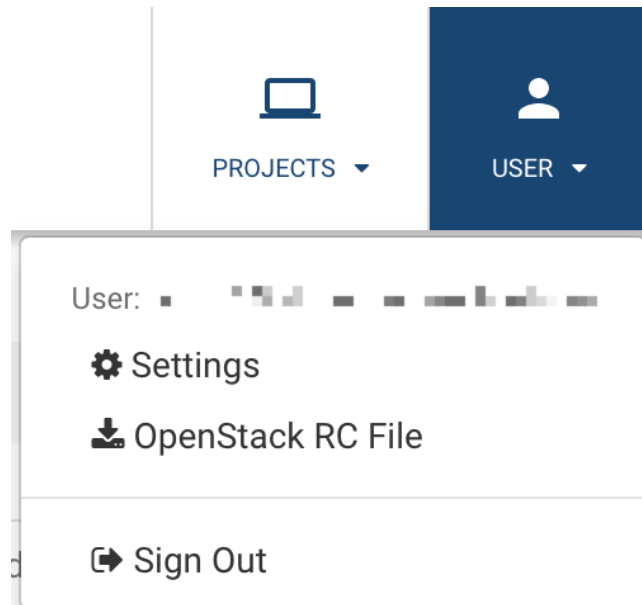


Figure 3-2 Download OpenStack RC File from the Cloud website.

- 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.

```
samxu@SamdeMacBook-Pro ~/Desktop/COMP90024/Cloud master + sh ./run-nectar.sh
```

Figure 3-3 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).

```
Last login: Mon May 13 20:41:40 2019 from 10.13.196.209
ubuntu@harvester1:~$ nohup bash marvel.sh >marvel.out 2>&1 &
```

Figure 3-4 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.

```
Last login: Tue May 14 08:10:50 2019 from 10.13.196.209
ubuntu@webserver:~$ nohup bash view_db.sh >view_db.out 2>&1 &
```

Figure 3-5 The first command line to run on Webserver.

```
Last login: Tue May 14 08:10:50 2019 from 10.13.196.209
ubuntu@webserver:~$ nohup sh manage.sh &
```

Figure 3-6 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).

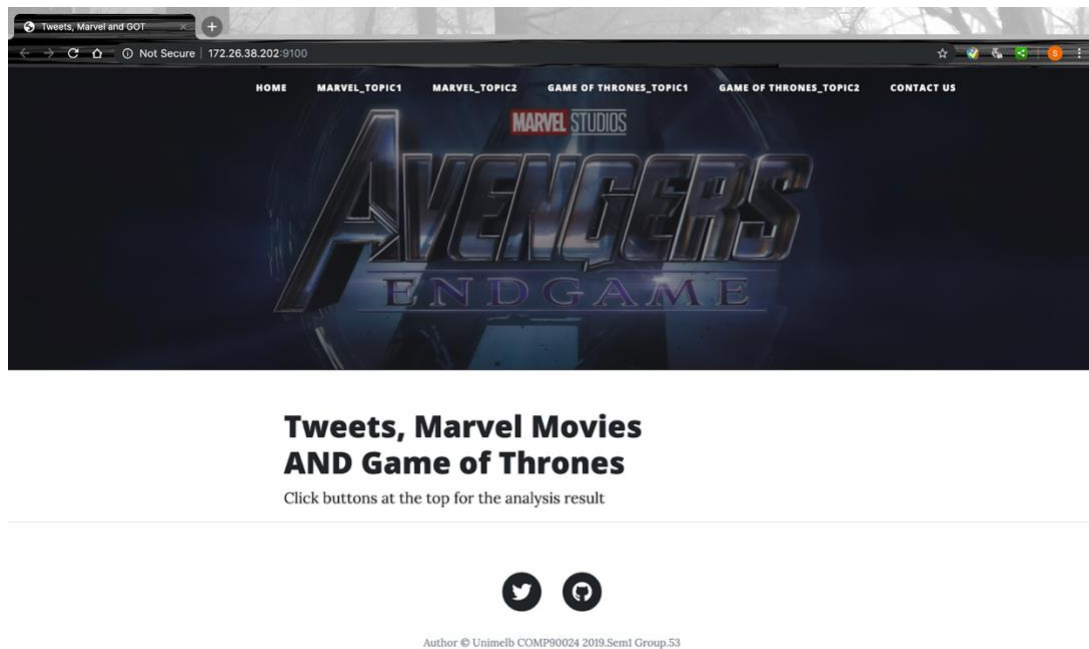


Figure 3-7 The website for data visualisation.

- Click buttons at the top for the analysis result.

4. System Architecture and Design

4.1. 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.

4.2. System Architecture

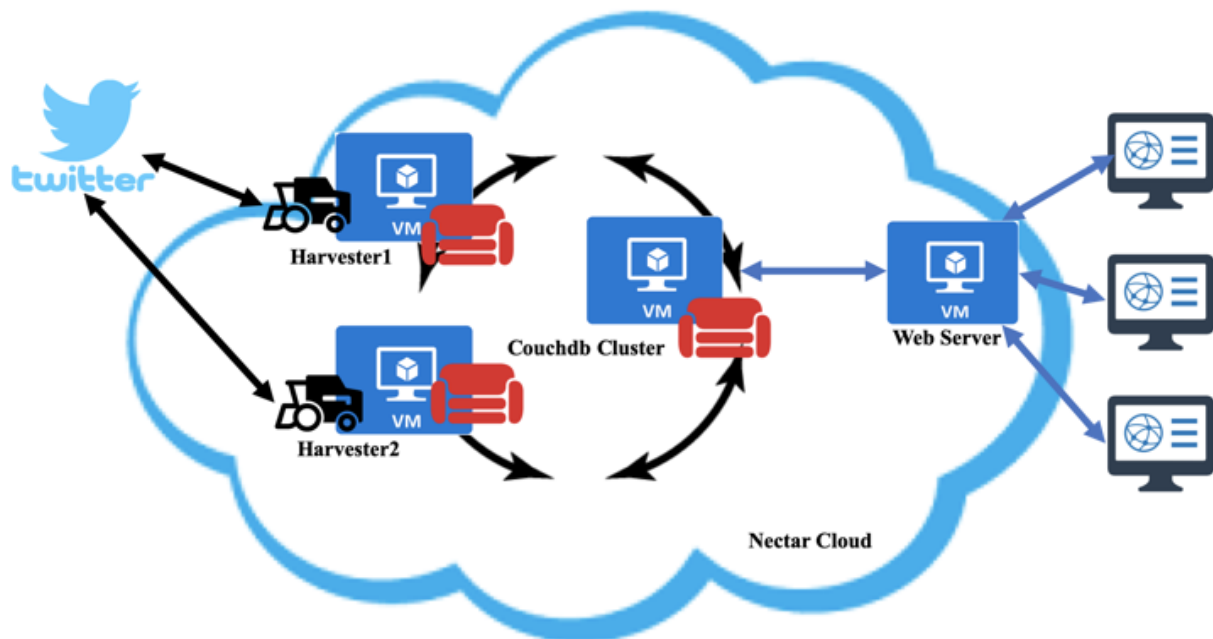


Figure 4-1 System Architecture.

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.

4.3. 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 4-1 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.

5. Scalability and Deployment Automation Detail

5.1. 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.

5.2. 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.

```

- hosts: localhost
  gather_facts: true
  vars_files:
    - host_vars/var.yaml

  roles:
    - role: openstack-common
    - role: openstack-images
    - role: openstack-volume
    - role: openstack-securitygroup
    - role: instance-dbServer
    - role: instance-webServer
    - role: instance-harvester1
    - role: instance-harvester2

```

Figure 5-1 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-harvester2*: 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.

5.2.1. 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.



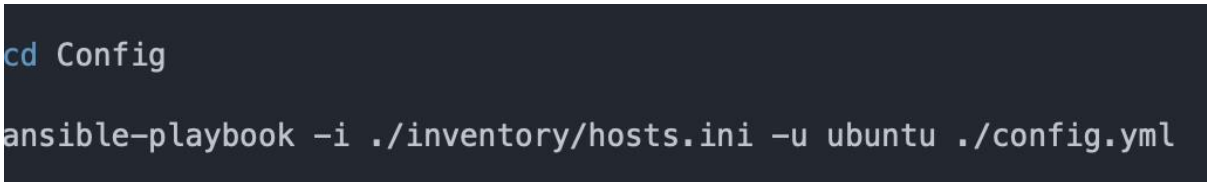
```
sleep 180
```

Figure 5-2 Using *sleep* command to avoid the unreachable problem for newly created instances.

5.3. 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:



```
cd Config  
ansible-playbook -i ./inventory/hosts.ini -u ubuntu ./config.yml
```

Figure 5-3 Command to run configuration playbook.

Hosts and the related private key statement:

```

hosts.ini x
1  [dbServer]
2  172.26.37.214 ansible_ssh_private_key_file=./keys/dbServer.key
3
4  [webServer]
5  172.26.38.202 ansible_ssh_private_key_file=./keys/webServer.key
6
7  [harvester1]
8  172.26.38.160 ansible_ssh_private_key_file=./keys/harvester1.key
9
10 [harvester2]
11 172.26.38.84  ansible_ssh_private_key_file=./keys/harvester2.key
12

```

Figure 5-4 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.

```

[defaults]
host_key_checking=False

```

Figure 5-5 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.

```

- name: Move storage path
  shell: mv -f /var/lib/docker /data

- name: Link storage
  shell: ln -s /data/docker /var/lib/docker

```

Figure 5-6 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).

```
- name: Install couchdb container
  sudo: yes
  docker_container:
    name: couchdb
    image: "couchdb:2.3.0"
    ports:
      - "5984:5984"
      - "4369:4369"
      - "9100:9100"
      - "9200:9200"
    env:
      COUCHDB_USER: "admin"
      COUCHDB_PASSWORD: "admin"
      NODENAME: '{{ harvester1 }}
```

Figure 5-7 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 pycharts 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.

6. 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.

6.1. Twitter Harvester

6.1.1. 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.

6.1.2. Parameters

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

Table 6-1 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', 'marvelcominics', 'avengers'
Example queries for topic 'Game of Thrones'	'game of thrones'

6.1.3. 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.

6.1.4. 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:

```
#!/usr/bin/env bash
python3 TwitterSearch.py -q '#marvel' -d 'marvel_tweets' -t 'marvel_user_timelines' -u 'marvel_users'
```

Figure 6-1 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.

6.2. 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.


```
{
  "_id": "1123723951581143042",
  "_rev": "1-e2097645928c3f8b7ea5be6b98575c3f",
  "text": "Celebrate the end of @GameOfThrones in",
  "hashtags": [
    "luxury",
    "GameofThrones",
    "GOT"
  ],
  "symbols": [],
  "user_screen_name": "CEOMagazineAU",
  "geo": null,
  "coordinates": null,
  "place": null,
  "is_retweet": false,
  "sentiment_score": 0,
  "subjectivity": 0,
  "week": "Wed",
  "month": "May",
  "day": "01",
  "time": "23:00:42",
  "year": "2019"
}
```

Figure 6-2 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.

```
{
  "_id": "5S0SQUEEN1992",
  "_rev": "1-a31d79bd45aa0e598010e7343f2df548",
  "name": "I'm suddenly Bethany",
  "id_str": "2594926345",
  "location": "Sydney, New South Wales",
  "description": "I used to be vegan, but then I became obsessed with Meet & Greet."
}
```

Figure 6-3 Example user object.

- A status object:

```
{
  "_id": "1000000656218902530",
  "_rev": "1-10cd119c9e2ae544639a3cf5be2e3b3e",
  "text": "@jonnoxrevanche @JillJacksoff Ugh omg. Fucking hot",
  "user": "neurosisfan1991",
  "sentiment": 0.25,
  "subjectivity": 0.85
}
```

Figure 6-4 Example status object.

6.3. Data Analysis and MapReduce

In this section, we will introduce how our system uses the built-in MapReduce function in CouchDB to perform the data analysis for different scenarios. A brief presentation and analysis of final result will also be included.

6.3.1. Sentiment change when someone mentions Marvel in his/her tweets

This scenario is to compare the sentiment score of the one user's tweets related to Marvel Movies with the sentiment scores of all his own tweets.

Marvel movies are popular all over the world. According to the "Sydney Morning Herald", Marvel's latest superhero movie Avengers Endgame has become the highest-grossing movie in Australian history. More than 95% of the audience praised the film (Carmody, B., 2019). This has raised our curiosity that we would like to figure out what is the relationship between LUST and the tweets posted by Sydney audience. The sin LUST includes strong love for certain things and sentiment changes (Seven Deadly Sins, 2019).

```
map_function_realted = """function (doc) {
  emit(doc.user_screen_name, doc.sentiment_score);
}"""
```

Figure 6-5 Map function_average sentiments for different users.

```
reduce_function = """function (keys, values, rereduce) {
  return sum(values)/values.length
}"""
```

Figure 6-6 Reduce function_average sentiments for different users.

For this scenario, we create two views. The first view is to get the sentiment score for each user based on all of his/her tweets. The second view is to get a second sentiment score for each user based on his/her tweets related to Marvel movies. Then for each user, we calculate the absolute value of the difference between these two scores. The following bar chart shows the percentages of users with different values.

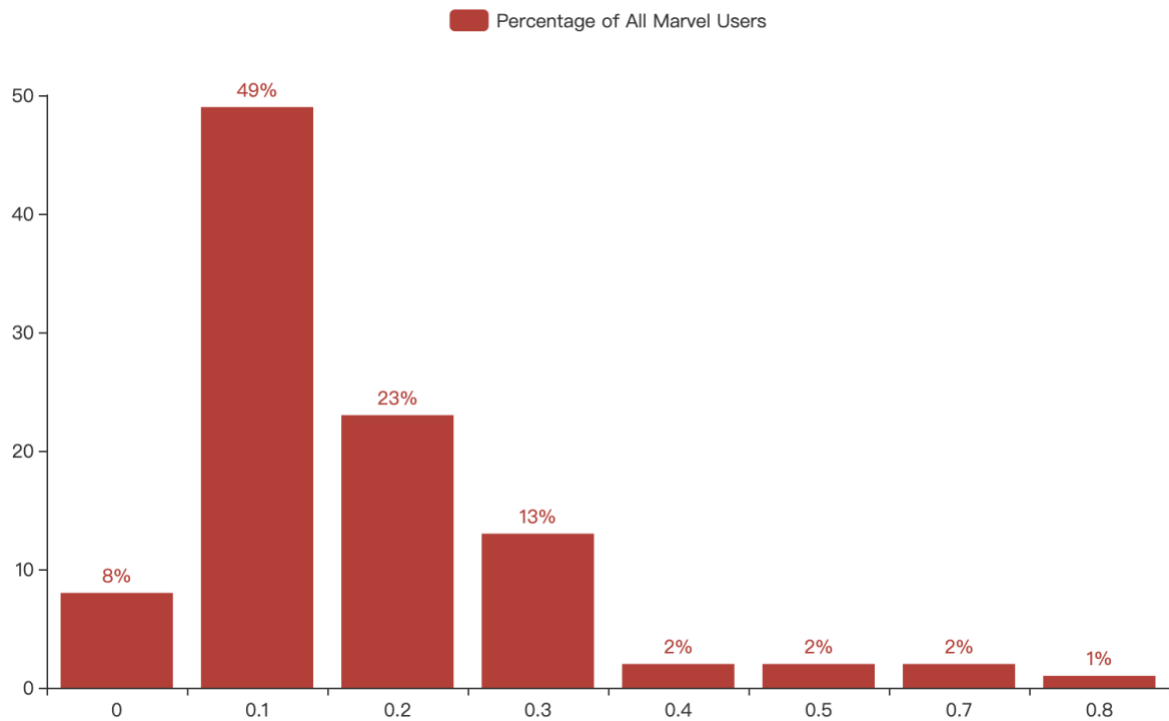


Figure 6-7 Sentiment Changes When Mention Marvel Related Content.

In this figure, about 85% of users have a low-level change, and the proportion of users with big emotional changes are not high. Hence, to some extent we can say the sentiments of Marvel audiences are not influenced much by this series of movies.

6.3.2. Sentiment change when someone mentions Game of Thrones in his/her tweets

An analysis similar to figure 6-7 has been conducted to Game of Thrones audiences in Sydney. This scenario is to compare the sentiment score of the one user's tweets related to Game of Thrones with the sentiment scores of all his own tweets.

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). Like what we want from first analysis on Marvel audiences, we want to know the LUST attitude of the audience to the Game of Throne.

Same as analyzing this topic about Marvel movies, we create two views. The first view is to get the sentiment score of each user based on all of his/her tweets. The second view is to get a second sentiment score of each user based on his/her tweets related to Game of Thrones. Then for each user, we calculate the absolute value of the difference between these two scores. The following bar chart shows the percentages of users with different values.

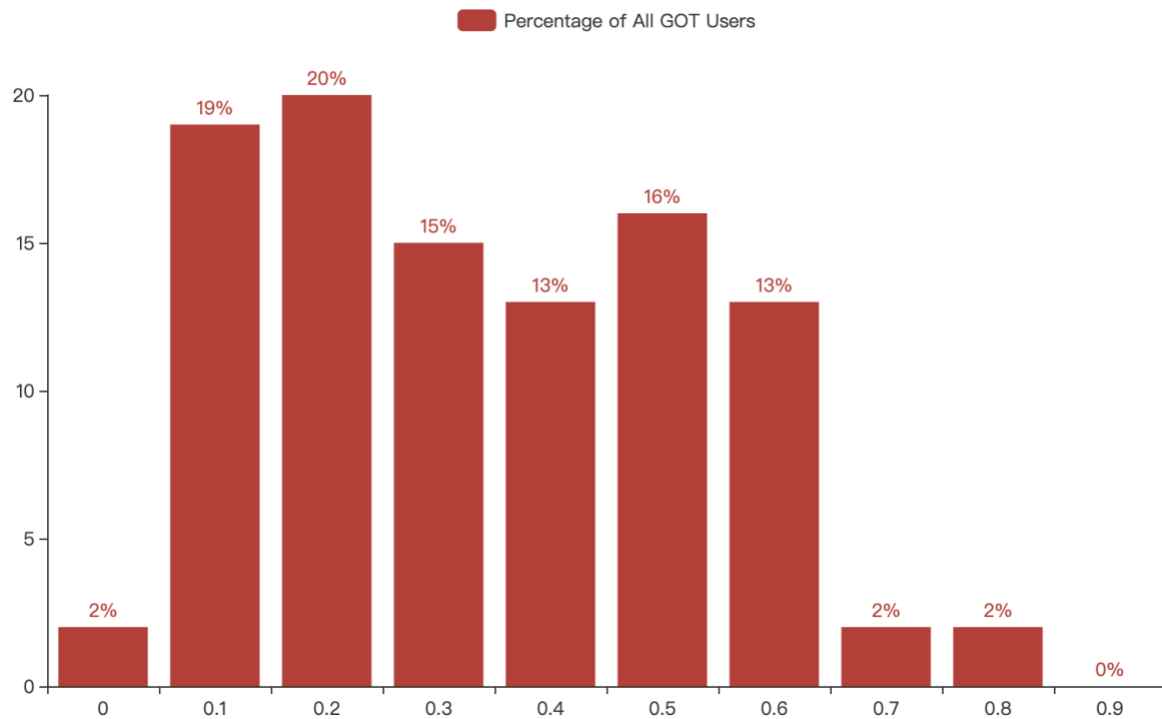


Figure 6-8 Sentiment Changes When Mention GOT Related Content.

As shown in the figure, most of them have a quite high-level sentiment changes when mention Game of Thrones. Compared this result with that of Marvel, there is a higher probability that Game of Thrones audiences are more emotion sensitive than Marvel audiences when they are discussing related content.

6.3.3. Sentiment trend related to Marvel (from 2015 to 2019)

This scenario is to analyze the trend of the average sentiment scores related to Marvel in recent years. We are curious about how the impact of Marvel movies on Sydney citizen would change as time goes by. To figure out this, we need to calculate the average sentiment scores of tweets over the years to see if there is any orderliness.

```
map_function_year = """function (doc) {
    emit(doc.year, doc.sentiment_score);
}"""
```

Figure 6-9 Map function for average sentiments over years.

```
reduce_function = """function (keys, values, rereduce) {
  return sum(values)/values.length
}"""
```

Figure 6-10 Reduce function for average sentiments over years.

We create one view for this scenario. This view is to calculate the average sentiment scores of tweets for different years. We filter the result and only choose the result of recent five years. This bar chart is the final result, showing average sentiments of tweets related to Marvel in recent years.

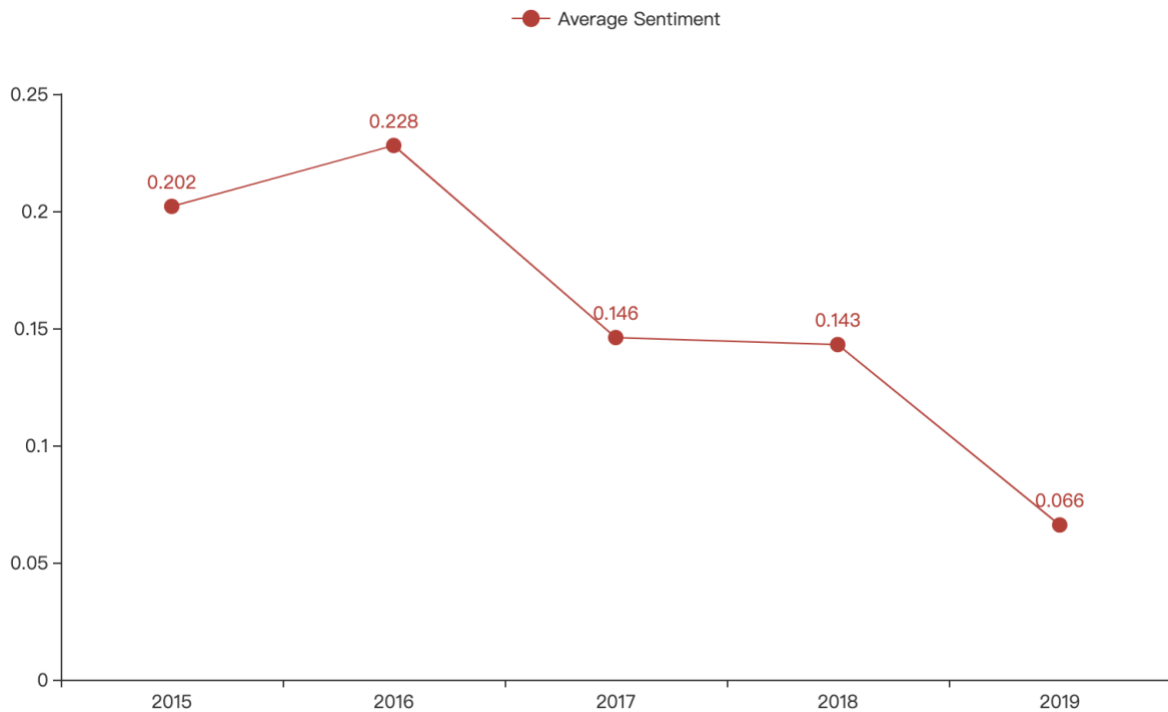


Figure 6-11 Average Sentiment Related To Marvel Movies in Recent 5 Years.

As you can see, the impact is positive the whole time. However, an interesting trend can be observed. The scores in 2017 and 2018 have dropped significantly compared to the previous two years. Therefore, as the impact of Marvel movies on people's emotion drop, we would assume that people are greedy that their interest in similar movies seems to become lower. In other words, people's expectations of Marvel movies seem to become higher and higher.

6.3.4. Sentiment trend related to the Game of Thrones (Jan 2019 to May 2019)

Similar to figure 6-8, this scenario is to analyze the trend of the average sentiment scores related to Game of Thrones in recent months.

We create one view for this scenario. This view is to calculate the average sentiment scores of tweets for different months. We filter the result and only choose the result of recent five months in 2019. This bar chart is the final result, showing average sentiments of tweets related to Marvel in recent 5 months.

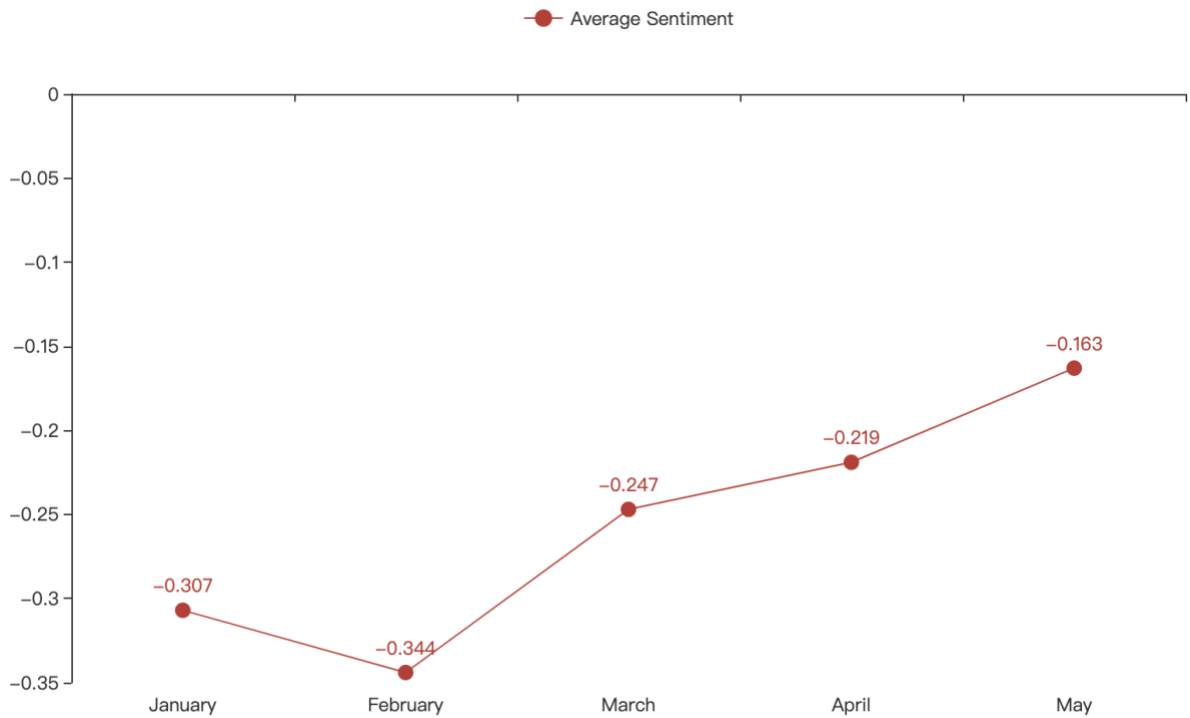


Figure 6-12 Average Sentiment Related To Game of Thrones in Recent 5 Months.

As shown in the figure, the impact of Game of Thrones on people's emotion is negative all the time, which is different from that of Marvel movies. However, the value is approaching 0, which means that the impact of the Game of Throne series on people's emotion is decreasing. Therefore, we could assume that people are becoming indifferent about them.

7. 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).

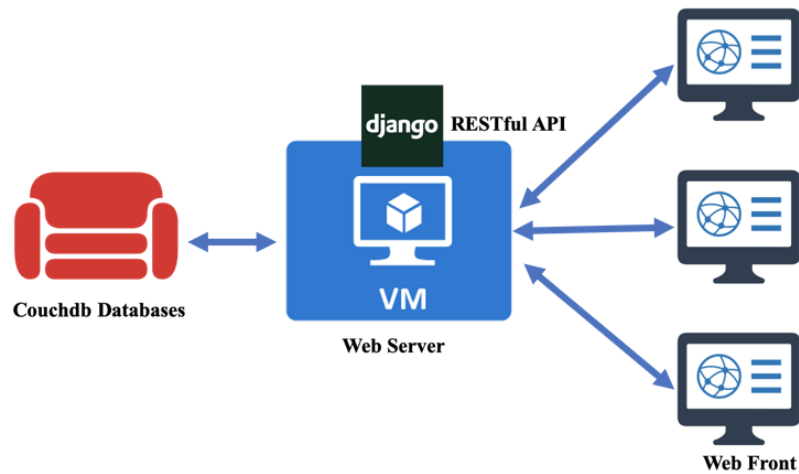


Figure 7-1 Web Application Design.

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 7-2 Base Header 1.



Figure 7-3 Base Header 2.



Author © Unimelb COMP90024 2019.Sem1 Group.53

Figure 7-4 Base Footer.

As required, we should build **RESTful APIs** 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.

```
url(r'^$', views.HomeTemplateView.as_view(), name='home_page'),
url(r'^contactinfo/$', views.TeamTemplateView.as_view(), name='contact_info'),
url(r'^marvel_topic1/$', views.MarvelTemplateView1.as_view(), name='analysis_marvel_user'),
url(r'^marvel_topic2/$', views.MarvelTemplateView2.as_view(), name='analysis_marvel_year'),
url(r'^marvel_topic1/marveluser.html$', views.MarvelPicture1.as_view(), name='analysis_marvel_user_picture'),
url(r'^marvel_topic2/marvelyear.html$', views.MarvelPicture2.as_view(), name='analysis_marvel_year_picture'),
url(r'^got_topic1/$', views.GotTemplateView1.as_view(), name='analysis_gameOfThrones_user'),
url(r'^got_topic2/$', views.GotTemplateView2.as_view(), name='analysis_gameOfThrones_month'),
url(r'^got_topic1/gotuser.html$', views.GotPicture1.as_view(), name='analysis_gameOfThrones_user_picture'),
url(r'^got_topic2/gotmonth.html$', views.GotPicture2.as_view(), name='analysis_gameOfThrones_month_picture'),
```

Figure 7-5 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 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.

8. Security, Fault Tolerance and Error Handling

8.1. 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.


```
CLNs-MacBook-Pro:keys chenlingna$ ssh -L 9000:localhost:5984 ubuntu@172.26.38.160
ubuntu@172.26.38.160: Permission denied (publickey).
```

Figure 8-1 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.

8.2. Fault Tolerance and Error Handling

- **Issues and challenges in using the UniMelb Research Cloud:** This has been discussed detailly 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.

9. 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 cannot 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]