Twitter and Australia Politics

James Sun [*] Naresh Olladapu [†]
Aanchal Bhambhani [‡] Zixi Chen [§]
Zexi Liu [¶]

COMP90024 Cluster and Cloud Computing - Team 71

The University of Melbourne

19th May, 2021

Abstract

The topic selected in this project is politics and social media domains. In this report, 4 system functionalities were developed as well as some scenario findings and analysis with graphical results; A simple user guide in the form of a demo video link; A detailed system architecture were described along with the justified design choices for each component. Then pros and cons of Melbourne Research Cloud were discussed and the tools and processes for the deployment. Lastly, fault tolerance and dynamically scaling aspects of this system were briefly described. The github repo can be found in this url `https://github.com/olladapunaresh/Assignment-2_cluster_and_cloud_computing`.

[*]Student ID: 831860; Location: Melbourne, Australia
[†]Student ID: 1233759; Location: Mumbai, India
[‡]Student ID: 1235772; Location: Mumbai, India
[§]Student ID: 831860; Location: Melbourne, Australia
[¶]Student ID: 813212; Location: Melbourne, Australia

Contents

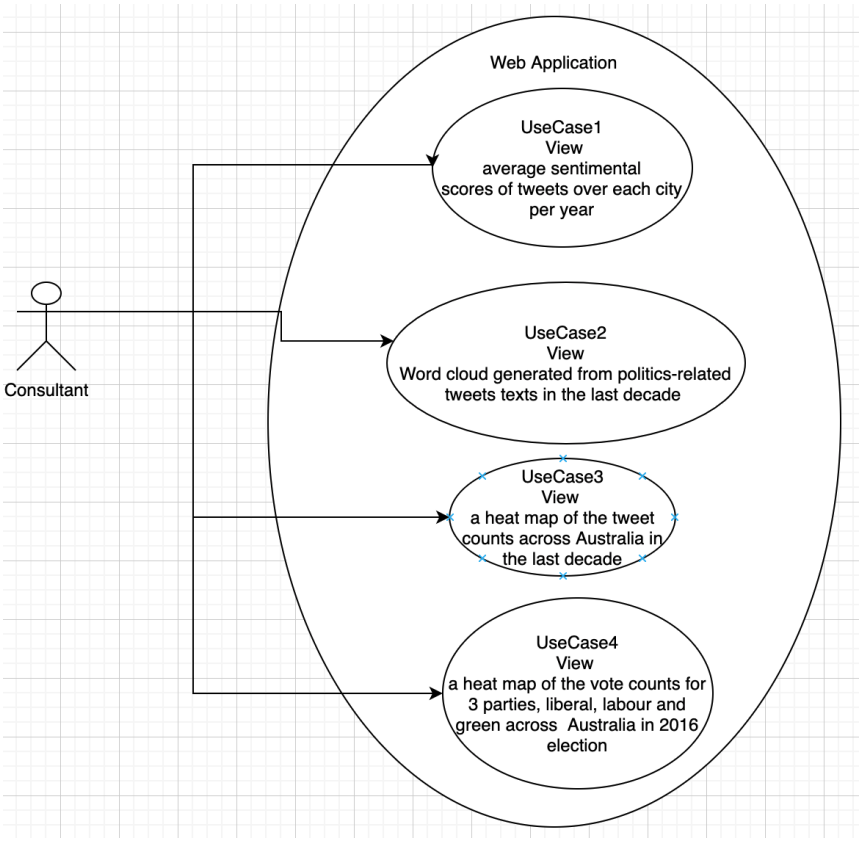


Figure 1: User story diagram

1 System Functionalities

The system devised in this project is catering for the needs of consultants in political domain and it gathers political sensitive information on Twitter and provides visualization tools to aid and deepen their analysis. As shown in Figure 1, we have 4 user stories in this system:

- User Story1: As a consultant, I want to view a Word Cloud[1] of political tweets, so that I can identify the viral topics on twitter and grasp what political topics people cared about.
- User Story2: As a consultant, I want to view the average sentimental scores aggregates over cities in Australia from 2010 to 2019, so that I can analyse emotions conveyed in political tweets in different cities across different time.
- User Story3: As a consultant, I want to view the number of political tweets counts on the Australia's map, so that I can identify clusters where voters are more active on Twitter. As a result, more resources will be used to target at the Twitter platforms in these regions.

[1]word clouds, or tag clouds "are visual presentations of a set of words, typically a set of tags, in which attributes of the text such as size, weight or colour can be used to represent features (e.g., frequency) of the associated terms"[4].

- User Story4: As a consultant, I want to view the number of votes counts on different parties on the Australia's map. Combined with the political tweets count, I can identify possible associations between party's popularity and the number of "twitterholic" in certain regions.

2 Scenarios Overview - Analysis and Results

2.1 Political Topics Identification - User Story 1

To highlight 3 scenarios for the use of word coulds in our project

1. Figure 2a shows "Submarine" is one of the trendy topics in Adelaide. Tags of "Submarine Price", "future submarine" and "aussie taxpayer" have big font size, which suggest the high frequency of these tags on appearing twitter text data. To some extends, this means Submarine is what locals in Adelaide cares about, which fits in the fact that Adelaide is where new submarines are being built for Australia

2. In Figure 2b, The word cloud of Canberra depicts topics on current policy. From this graph, we can view the very recent and ongoing "Travel ban", "Indian travel", "conceded india on the top right. Furthermore, "quarantine capacity", "repatriation flight" covid-related tags are in the bottom left.

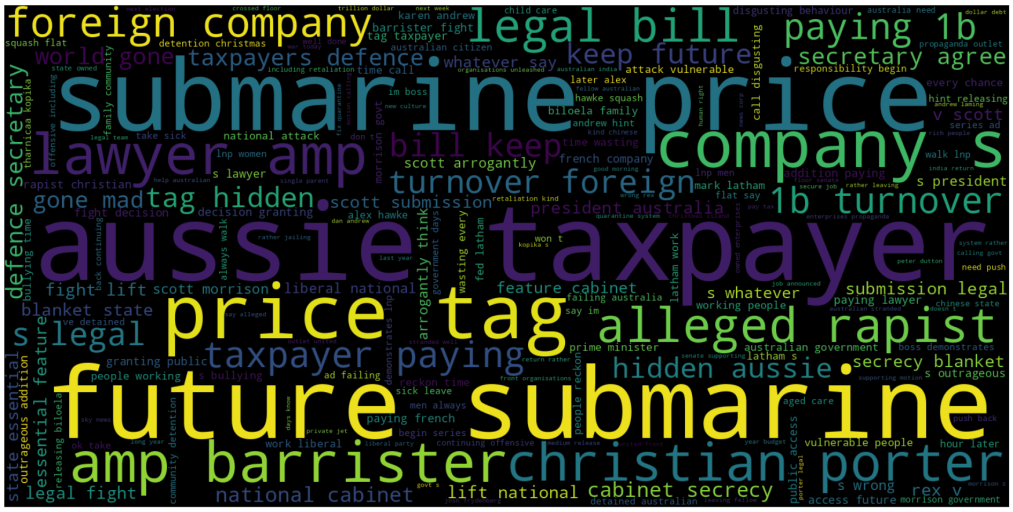
3. In Figure 2b, some controversial politicians appears in Canberra's word cloud too. For example, "Andrew Laming", a controversial MP of Queensland, who was involved bully scandals, yet refused to give up his "parliamentary committee" role in the middle area. In addition, "Christian porter", a MP for an electorate in Perth and he was named in a rape allegation. Last but not least, the current administration, "morrison government", "scott Morrison" is no doubt "popular" on twitter.

2.2 Sentimental Analysis - User Story 2

In Figure 3, these graphs show that the average compound sentiment score of Tweets originating from particular cities in Australia over the past decade.

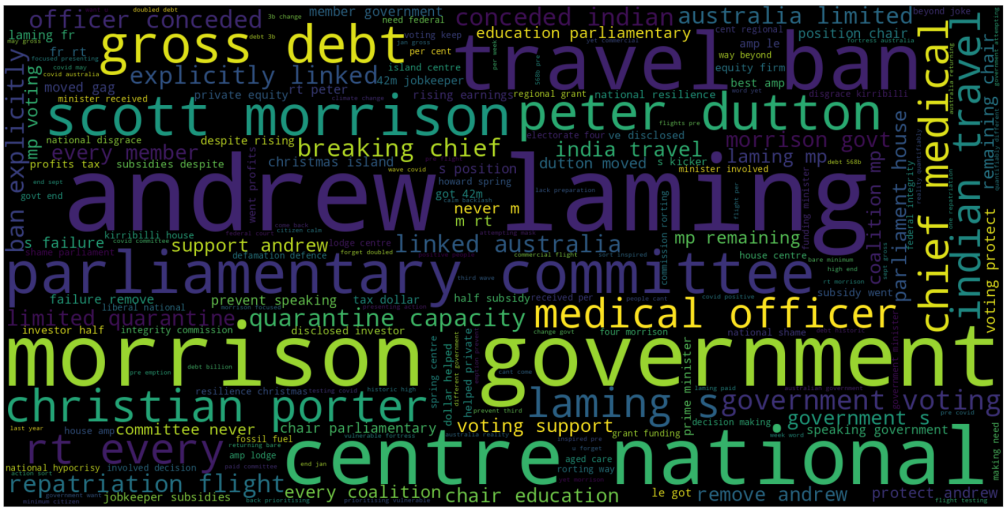
- None except Brisbane in 2012 used more emotive language or leaned towards a particular emotion on political tweets. In Figure 3, most cities' compound score fluctuates from the range of -0.2 to 0.2, meaning that there was no significant positive (+1) or negative(-1) emotion trends over the years. Except for Brisbane in 2012, had a maximum 0.4 compound score, suggesting relative positive emotion in the region. Furthermore, in Figure 4 , the average neutral sentimental scores for all cities are in the range of 0.6 to 0.85. This high-value range further supports neutral emotions dominate the political tweets over cities. Combined, these results indicated all cities overall holds neutral emotions on political tweets.

Adelaide



(a) word cloud in Adelaide

Canberra



(b) word cloud in Canberra

Figure 2: Political Topics Identification - User Story 1

Figure 3: Average Compound Sentimental Score Trends

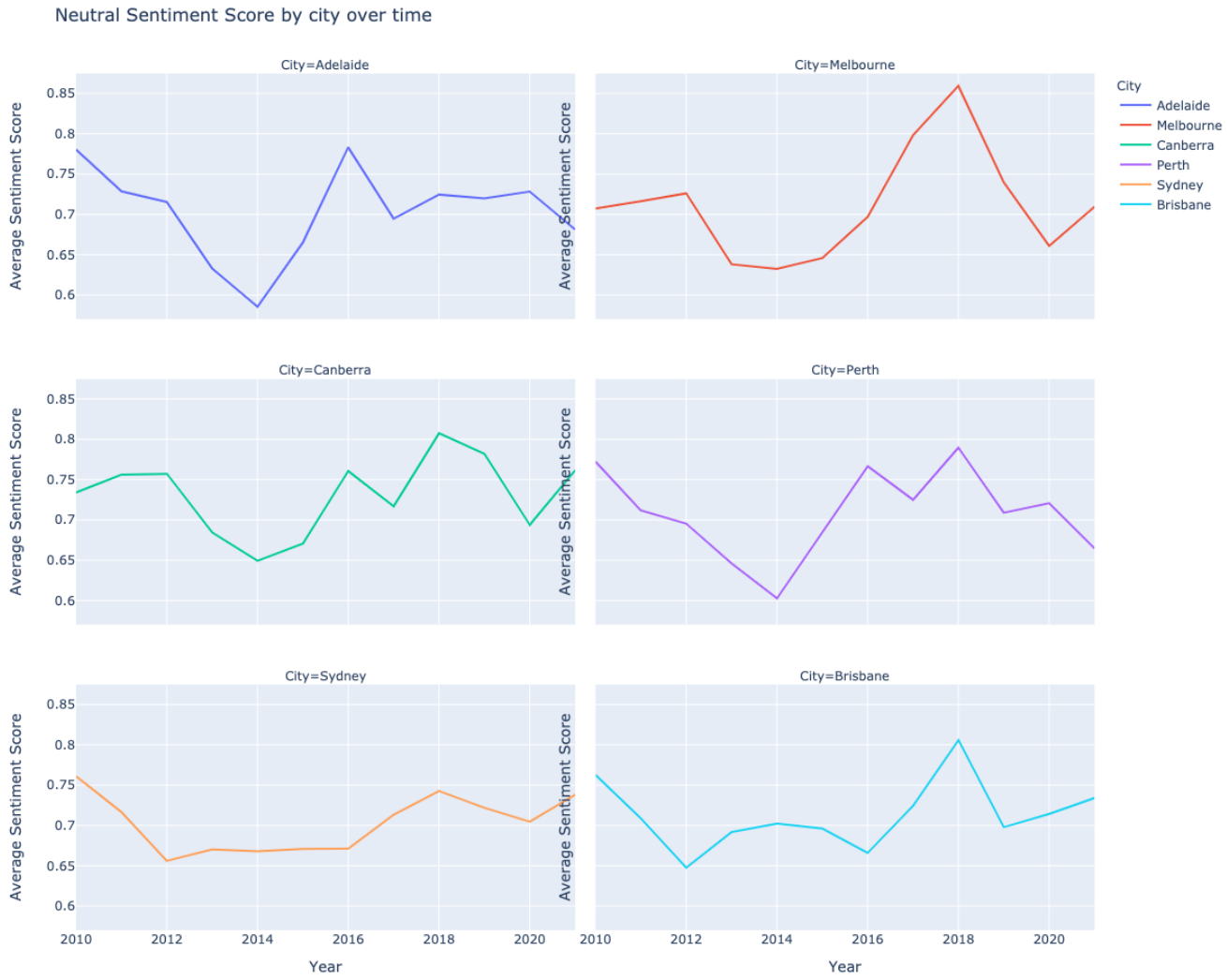


Figure 4: Average Neutral Sentimental Score Trends

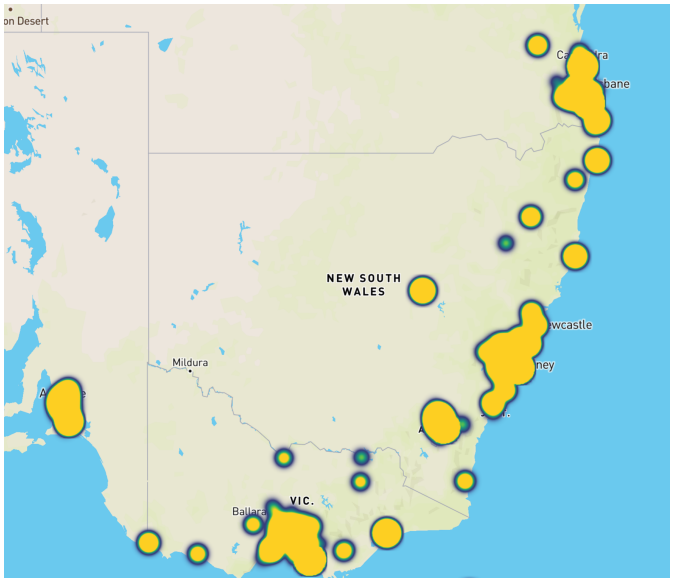


Figure 5: tweet counts across cities in Australia

2.3 Tweet's count, Vote's count Associations - User Stories 3,4

2.3.1 Tweet's count

In Figure 5, the yellow color measures the notion of "active" levels of Twitter users on politics. In other words, in the region with bright yellow, users are more active on Twitter debating politics, while in the region with green or blue color, it is not so much political discussion on Twitter.

- In Figure 5, Melbourne, Sydney and Brisbane have larger clusters of bright yellow while Adelaide and Canberra have smaller yellow clusters.

2.3.2 Vote's count

In Figure 6, the colors of the circles on the map represent different parties, liberal is blue, labor is red, and green is green. The size of the circles represents the number of vote counts. The bigger the size, the more the vote counts for a party.

- According to the vote count geographic distribution in Figure 6, it suggests the inner city of Melbourne is pro green, the south east is pro liberal and the north west is pro labour. That is, the green has the biggest size in the CBD area and then significantly shrink across the suburb. The red size of circles dominates the southeast area and the blue size of circles dominates the southeast regions.

2.3.3 Associations

- To a certain extent, a positive association between the number of the green vote and the number of tweets count for the Melbourne region. That is, it is likely green voters are more active on social media. According

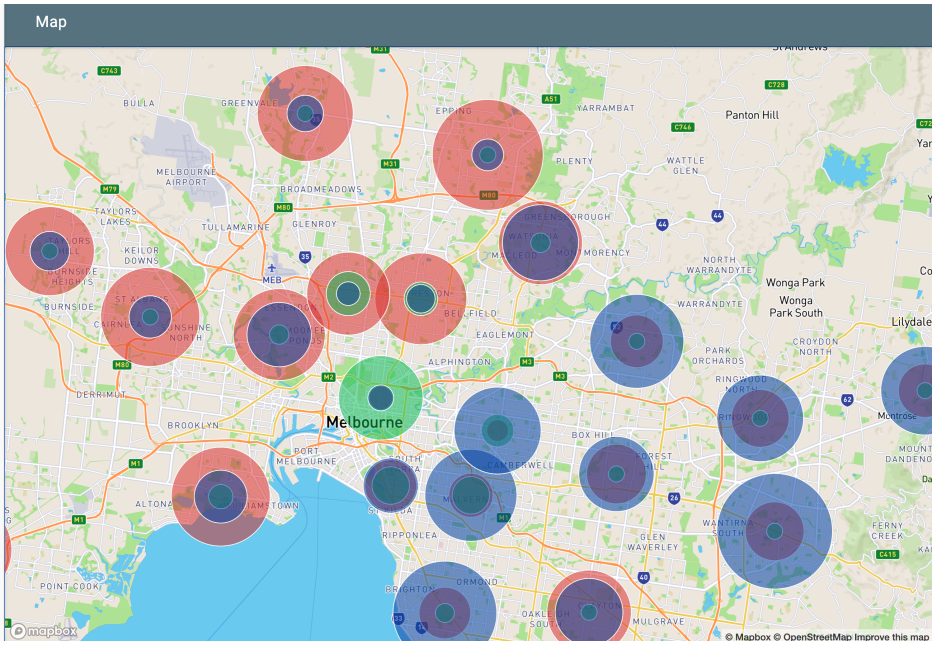


Figure 6: vote distribution in Melbourne

to Figure 7, the CBD is where the tweets are most active as well as having the largest number of green voters. In the surrounding suburbs, the colors of twitters are not as bright with smaller size circles of green voters.

- There may be a negative association between the number of labor votes and the number of tweet counts for the Melbourne region. In Figure 8, some of the biggest the labour clusters are in the northwest neighborhood, where almost no tweets are shown on the map. While in the CBD, with the most active Twitter users, the labor vote count is one of the smallest.

3 User Guide

3.1 System Deployment

Https proxy requires installation of certificates which we are not doing in this project. Due to this security risk, the system was not exposed to the public internet. Therefore, as a user, to use this system,

1. You may first need to connect to the University of Melbourne network by VPN or connect to University of Melbourne WiFi. Step by step VPN connection tutorial can be found in this url, https://how2connect.unimelb.edu.au/vpn/anyconnect.html

2. After you've logged on the network, open your browser, enter http://172.26.134.11/ in the search bar. This will lead you to the main page of our application.

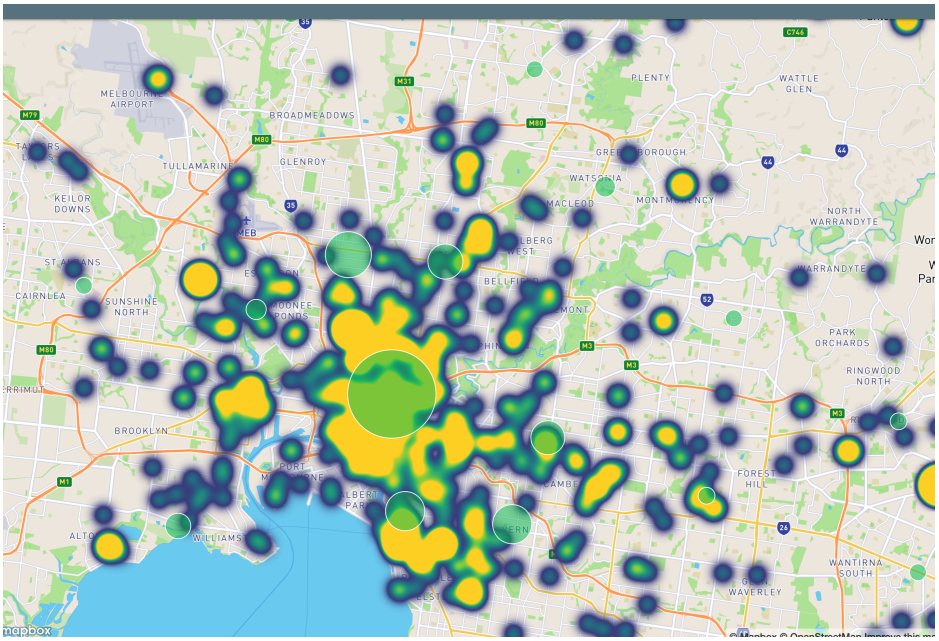


Figure 7: association: green vs tweets in Melbourne

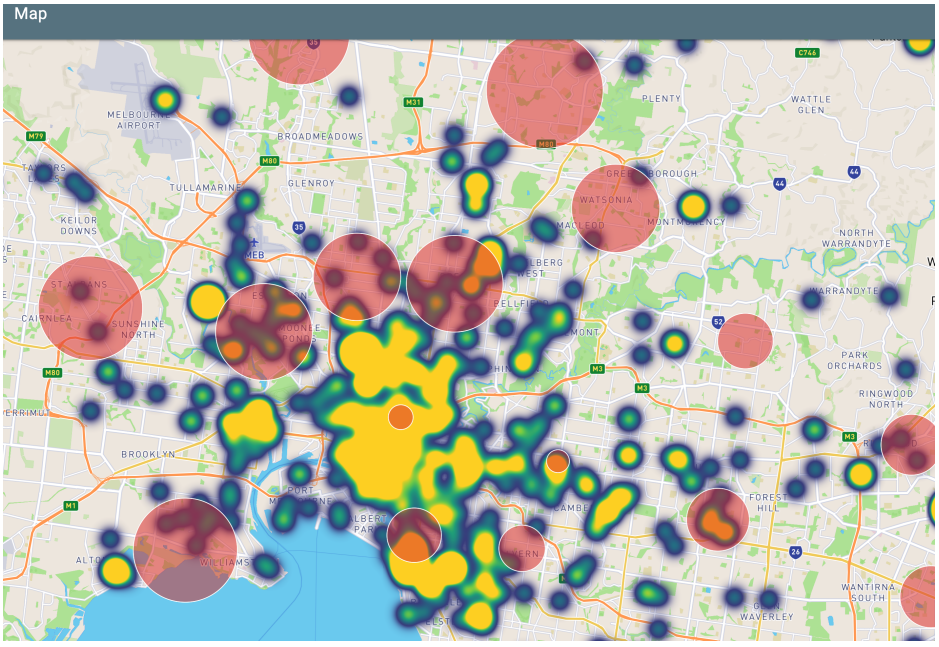


Figure 8: association labour vs tweets

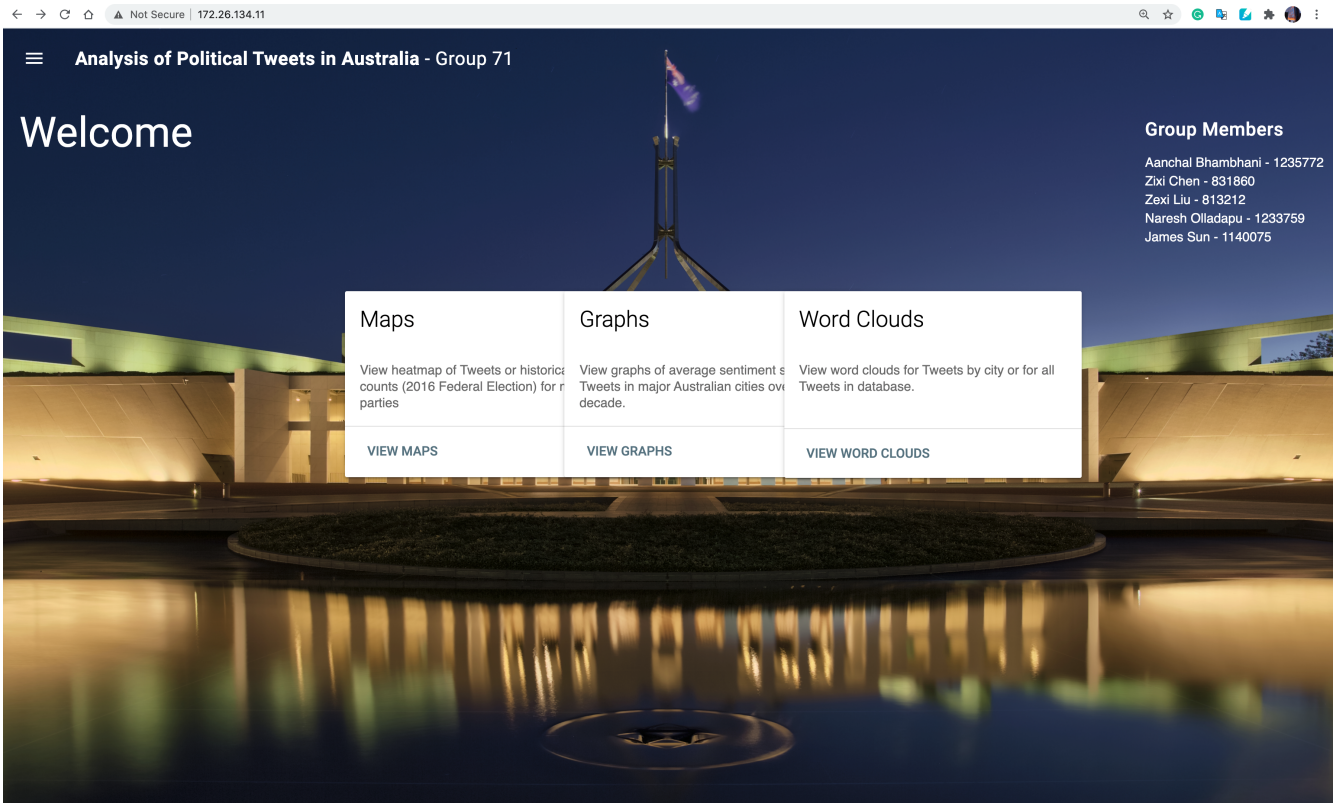


Figure 9: main page for users

3.2 End User Invocation

3.2.1 main page

The user story invocation from the main page will be shown from the demo in this url

3.2.2 User Story 1 invocation

3.2.3 User Story 2 invocation

3.2.4 User Story 3 invocation

3.2.5 User Story 4 invocation

4 System Design and Architecture

4.1 Overview

The system devised in this project was deployed on Melbourne Research Cloud. We created 4 docker nodes on 4 virtual machines. Each virtual machine has 1 docker node. A UML component diagram illustrates the system architecture design in Figure **??**.

- 3 docker nodes i.e. data nodes were designed to fetch, clean, process and store data.
- 1 docker node i.e. hosting node was designed to host front-end and back-end application

4.1.1 Data Nodes

Data nodes has 3 components:

- Twitter Harvester
- Data Analysis
- CouchDB database

The data node was built up as a complete pipeline solution. Twitter Harvester component first fetches data, then the Data Analysis component processes data and inserts them into the CouchDB component. This tight integration into the storage environment design simplifies the scalability of the system. That is, by replicating a data node and add up to this system.

This data pipeline design allows parallelization. Each data node has a Twitter harvester component, which allows harvesting different sources of data asynchronously; each data node has a Data Processing component, which allows parallelization in cleaning and calculations. Combined, this pipeline design will effectively handle an increase in the amount of data size.

4.1.2 Hosting Node

Hosting node has 2 components:

- Back End Flask API
- Front End Flask API/JavaScript

The hosting node was designed to handle user interactions, fetch derived analyzed data in the data nodes and then rendering analysis plots for the end user.

The Front-End component was delegated to listen to the user invocations and render GUI, while the Back-End component call requests to the database on data nodes and feeds the processed data to the front-end for visualization.

A sequential diagram in Figure 11 will show the system flow of our architecture.

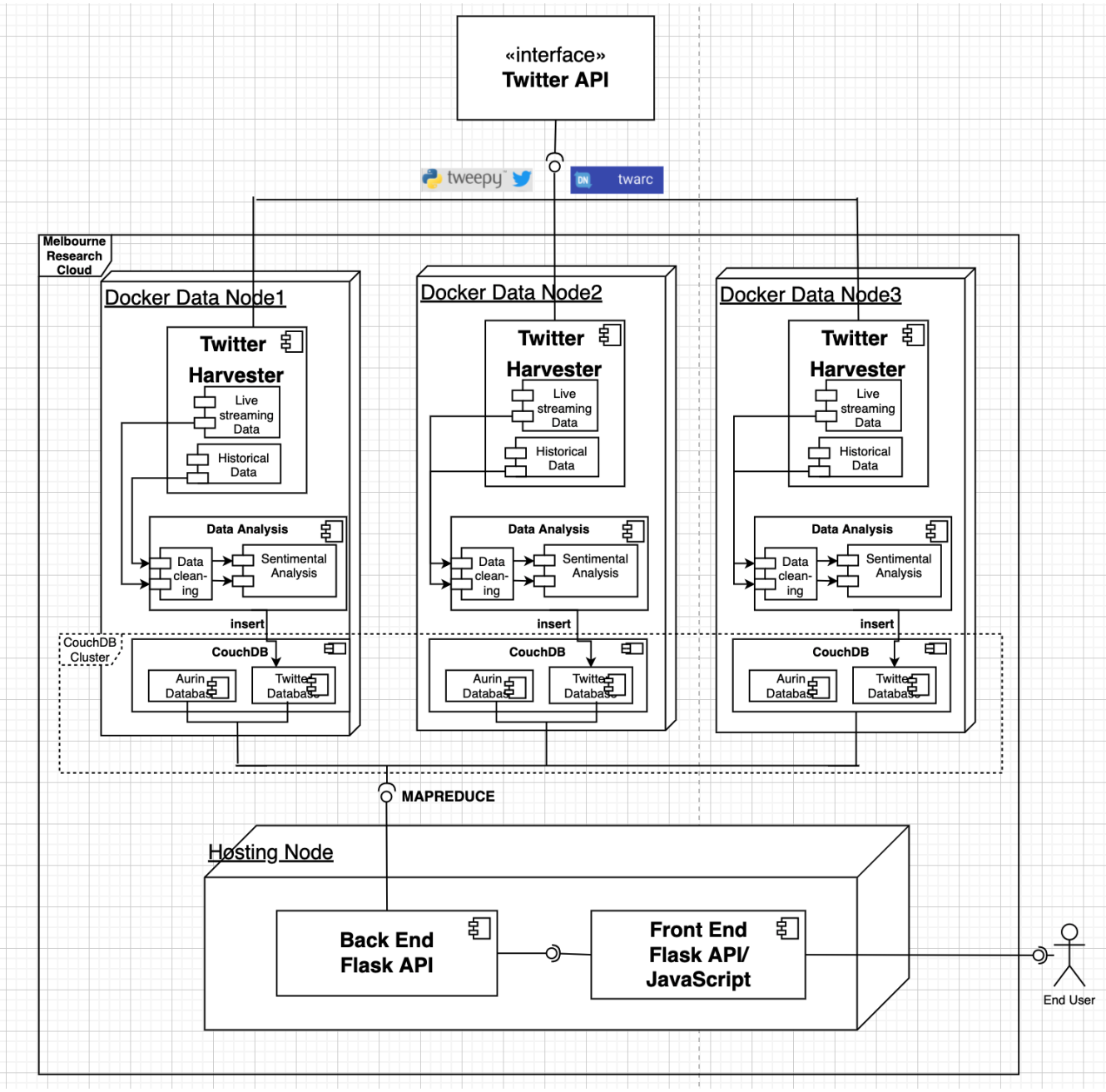


Figure 10: System Architecture - UML Component Diagram

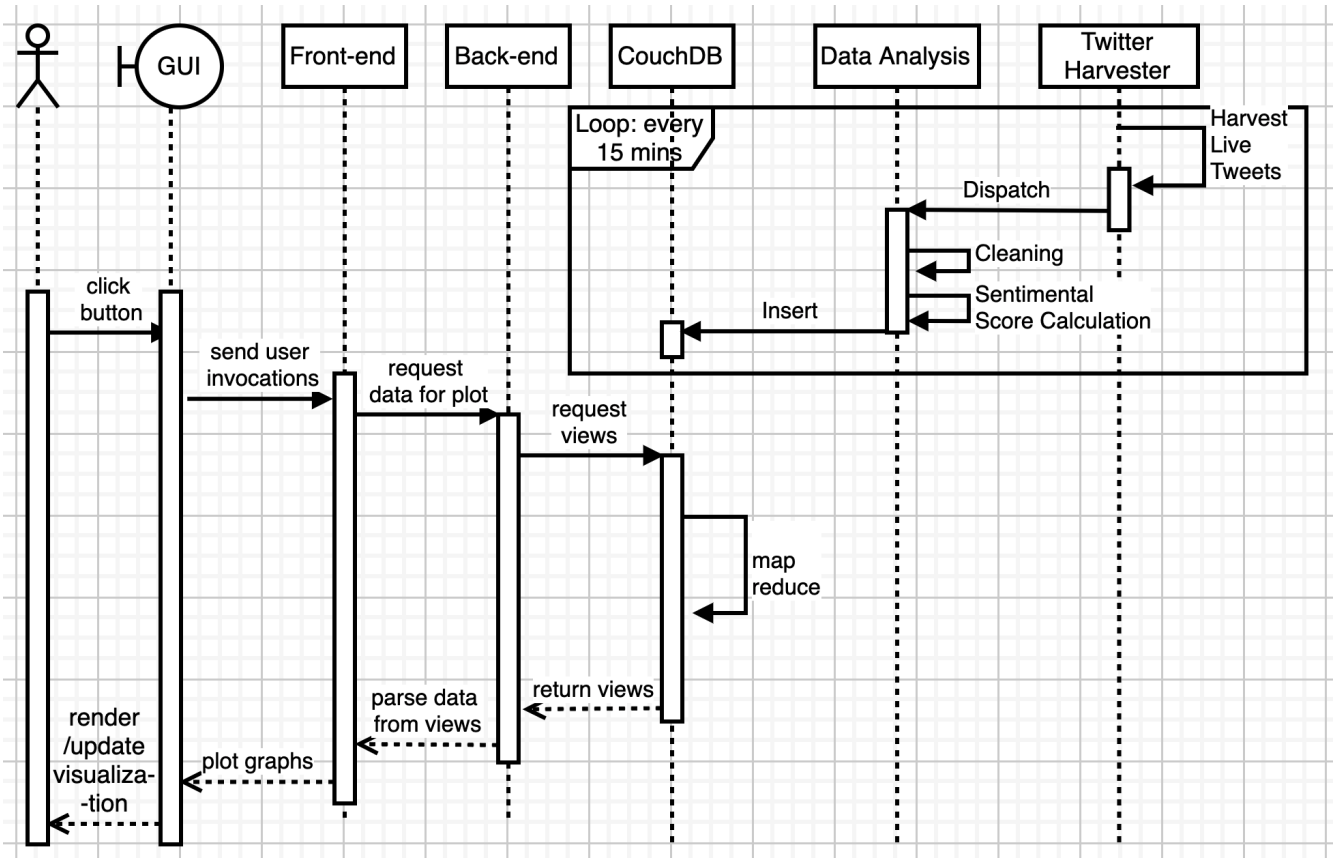


Figure 11: Sequential Diagram

Hashtags	'#Morrison','#climatechange','#Labor','#BlamingLabor' '#NewsPoll', '#ScottyMustGo','#ScottyDoesNothing','#SelfishAlbo' '#scottyfromarketing','#ScottyFromPhotoOps','#LNPfail' '#ScoMo','#Albo2022' ,'#EnoughIsEnough','#NotMyPM' '#liberal','#ScottyFromMarketing','#policyfail','#ScottyFromCoverUps' '#March4Justice','#AlboForPM','#AFPraids','#ShameonLiberals' '#SelfishAlbo','#auspoll','#onyourside', '#LeadershipFail' '#auspol', '#COVID19', '#AusVotes2019', '#ausvotes'
Politician Twitter account	James merlino, MarkMcGowanMp, AnnastaciaMP, DanielAndrewsMP

Table 1: list of search keywords

4.2 Twitter Harvester

4.2.1 How the tweets of interest were filtered

The Twitter harvester was implemented to filter tweets of interest in two ways. The Twitter harvest was written in Python. Both Tweepy and Twarc python libraries were used to access the Twitter API calls. Namely, the Twitter search API was called to fetch tweets with two filtering conditions.

1. Tweets that are outside of Australia were filtered out, as we are interested in the political activity in Australia.
2. We used a list of politics-related hashtags, and only fetch those tweets that contains these hashtags. The list of hashtags is showing in Table 1.
3. We used a list of politician's name, and pull all the tweets under their twitter account. The list of name are in Table 2

4.2.2 Proper removal of duplicate tweets

There are no duplicate tweets in CouchDB. This was done by making the couch a unique identifier, the tweet's unique id. This ensures no matter how many times the same tweet was inserted, this tweet will still be one row with its unique tweet id as the identifier in CouchDB.

4.2.3 Problem: lacking tweets

However, the basic free twitter developer API has a timeframe limit on API calls, where no tweets will be returned for a date older than one week. This had led to the lack of data for plotting and scenario analysis during the development process.

This problem was solved in two ways:

1. Firstly, we've decided to use academic twitter API calls to expand the time span of available tweets. In this way, with the increase in the total amount, the quantity of politics-related tweets reached enough for scenarios and use cases. Although this solution was not perfect, in exchange, it gave us more time to focus on other priorities in this project.
2. The second solution to lacking-tweets problem is that our twitter harvesters are actively listening to the Twitter API every 15 minutes, consequently, it will fetch any new tweets tweeted in the past 15 minutes.

4.2.4 Historical tweets vs Live tweets

Above two solutions as to lacking tweets problem defines what are historical tweets and live tweets in this project.

- Historical data: are the tweets fetched and filtered by academic twitter api in the last decade.
- Live data: are the tweets that are in past 7 days until now, and will be updated every 15 minutes given new tweets emerged.

4.3 Data Analysis

All historical tweets and live tweets are cleaned, processed and analysed the same way in Data Analysis component. The steps are as follows

1. Break the JSON files of the fetched tweet data (3GB) into smaller chunks (300MB)
2. Read the smaller JSON chunck into pandas.DataFrame object
3. Pick up of interest column value, these columns are "created date", "unique-id", "raw text", "location" and "coordinates". Create a new DataFrame to store these columns.
4. Clean raw text by lowering cases, removing stopwords, whitespace, hyperlinks, emojis and lemmatization.
5. Generate sentimental scores of the cleaned text and append values to the new DataFrame as columns.
6. Dump DataFrame as JSON object and push them to CouchDB

4.3.1 Proper handling of the errors tweets

The errors occurred in terms of missing value. The value was missing in step3 before dumping the DataFrame into the Json object. For example, there are tweets that do not have "coordinate" values. This will cause the heterogeneous data problem if we insert this Json data into CouchDB.

To solve this problem, we simply interpolated "null" as the value if the coordinate value is missing. To ensure robustness, we then programmed the rest of the codes to handle missing value cases if the coordinate is "null".

4.4 Front-end and Back-end Flask

The back end and front end were both implemented in Flask web framework. Flask is a Python web framework, meaning that it is a third-party Python library used for developing web applications [5]. This framework was chosen because it is easy to pick up by the team therefore risk-averse.

4.4.1 Design Choice: the necessity of Back-end layer

Some may realize, why don't we integrate front-end and back-end into a single component since they are using the same framework?

To answer this question, the necessity of a back-end service layer is to prevent security risk. That is, to act as a proxy between the database and the end-user. This will prevent attackers from directly accessing our database

through the browser. In other words, suppose there is no back-end layer, the attacker can directly query CouchDB data from the browser search bar, even delete data in CouchDB. Exposing our system in this way will never be considered a good design.

4.5 Database: CouchDB

The NoSQL database CouchDB was used in this project to store tweets data and aurin data. Furthermore, its built-in map-reduce query interface was frequently called in the back end.

4.5.1 MapReduce in CouchDB

In couchDB, the combination of the map function and the reduce function is used to create views. The map function loops through each document in the database and may choose to emit some data in the document depending on the logic of the map function[6]. Then with the reduce function the output of the map function can be aggregated according to their keys.

In this project, the built-in MapReduce functions were used to aggregate different types of sentimental scores (i.e. neutral, positive, negative and compound) in separate views according to the location of the tweet. Built-in couchDB reduce function was preferred over custom reduce functions because the built-in function runs faster than latter. Although custom reduce functions come with their own advantages, that is, less redundant statistics. For example, in this project, by using _stats, the built-in views contains redundant statistics such as min, max,sumsqr, which were not used in the front-end). However, it is a worthwhile trade-off between efficiency of building the views with built-in reduce functions and 'fatter' views as they run significantly faster than custom function, which would be written in JavaScript, according to the couchDB official documentation[10].

For this application, the MapReduce mechanism of couchDB was used to handle the large amount of data inputted from the Twitter harvester and extract useful information from the piles and output to the front end through views.

4.5.2 Why use MapReduce?

Cleaning the data ensures that the views generated from these data would not contain any 'noise' (i.e. useless information) that is not necessarily valuable for the front end to display, and hence the code for constructing the view would be much simpler and as a result more efficient, and the views would take shorter to build. Furthermore, as the views are simpler it saves computational resources when constructing the views as there is no need for filtering data to be happened through the views by setting various if-conditions and enhances efficiency when the views are updated with new tweets.

In the context of the application, the benefit could be huge since views would inevitably take longer to build when 1. the logic of the map function and/or reduce function is complex, and/or 2. the amount of data to be processed for building the views are huge, so as solution to 2., namely reducing the amount of data aggregated by a view does not sound sensible, only the solution to 1., that is, to simplify the logic of the view as much as possible, would work.

4.5.3 Database Design

In the system, two databases are used to store the data which would form the basis of map reduced views and hence the input for various user-friendly data representation in the front end (for example, graphs and maps). Namely they are 'politicaltrend' and 'aurin' respectively.

- 'politicaltrend' database: stores the historical and live tweets data and each tweet had been processed before it was inserted into the database.
- the database 'aurin' stores the filtered and processed data fetched from a AURIN dataset, i.e $AEC\_Federal\_Election\_First\_Party\_Preference\_by\_Polling\_Place\_Point\_2019$ dataset, the data is used for mapping the election vote count results on the map.

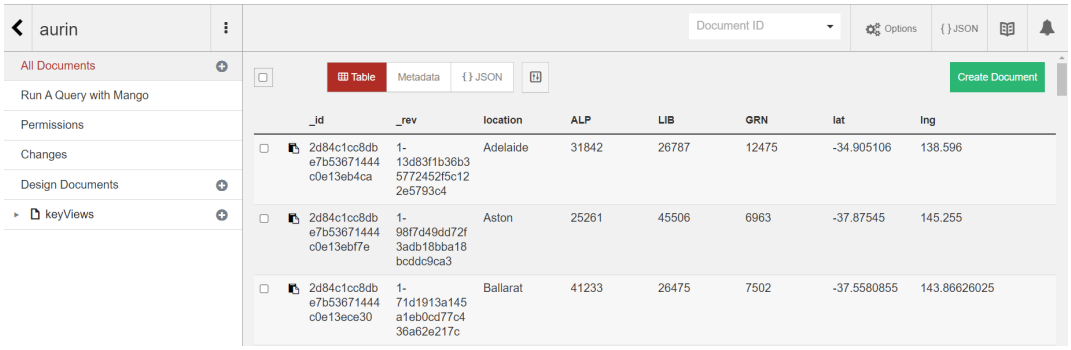
Some may ask why this design?

- These two datasets('aurin' dataset and the 'politicaltrend' tweets) have different sets of fields. Only some of their fields are shown in Figure 12a,12b, but we can still see the difference. Hence storing them separately allows the implementation of simpler views(i.e.less filtering). As a result it has the benefits described above.
- Secondly, if one of the database fails, some of the functionality will still work in our system. for instance, if 'aurin' failed, then the User story 4 will not be realized. But the rest of the user story functionality can still work. The user can still view the graphs that's only based on the data from "politicaltrend" database.

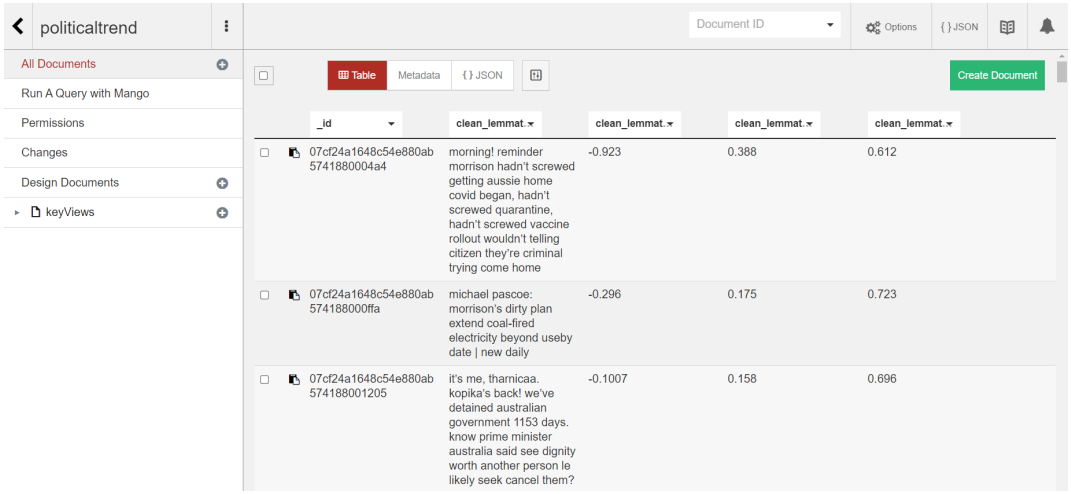
4.5.4 Advantages of CouchDB

CouchDB has the following benefits, especially under the context of developing this application:

- CouchDB allows the storage of schema-less documents. The significance of this characteristics is that the databases deployed for the application can comfortably deal with the heterogeneity of the JSON file (that contains information about each tweet), more specifically, there are different numbers of fields for each entry in the data base[12] For instance some entries in the politicaltrend database do not have coordinate information attached to it due to the lack of availability of such information for that particular source of historical tweets, but as ultimately those tweets can be grouped together because they convey similar information, splitting them into different databases purely for the sake of maintaining a rigour structure is not a sensible decision to make here. Although we input "null" for missing value, here is where couchDB will still come in handy, in that data can be grouped with applying filters when coding up views and as a result the end-user of the database (e.g. the front-end) can still be given succinct data that suits their purposes.
- CouchDB is easy to use, thanks to the couchDB API design that makes use of HTTP, making it possible to perform CURD operations on data/views by sending HTTP requests[11]. It means that for the developers using CouchDB, the views can be easily accessed with a browser, which is convenient for testing purpose. Moreover the views from the browser also resemble what the front-end would be likely to see (it is 'likely' here as for the application that was developed, the format of the views were twisted by the Flask API connecting the database and the front end), and hence making it much easier to communicate with the front end,as the couchDB API hides all the complexity, therefore they can type in the URL in the browser, and see what the views are like, without knowing much about how the views were built.



(a) Aurin Database



(b) Politicaltrend Database

Figure 12: different datasets has different fields

5 Cloud Infrastructure

5.1 Automation Tools - Ansible

Ansible is a simple automation language, supports adhoc task execution and configuration management[7]. With the need of executing a task in Ansible more than once, Ansible playbook is written to deploy this multi-machine system[7].

5.1.1 Tasks Summary

For this project, these following tasks were executed on Melbourne Research Cloud:

Ansible Playbook Tasks Summary

Part1: Automate Cloud Provisioning

- connected Melbouren Resaerch Cloud using OpenStack
- created volumes, security groups and instances
- binded volumes with instances

Part2: Install and Prepare Docker

- installed Docker
- created docker instances and mount volumes on them
- prepared Data Nodes and Processing Node for docker instances

Part3: CouchDb on Data Nodes

- installed CouchDB on Data Nodes
- configured and deployed a CouchDB cluster

Part4: Twitter harvester on Data Nodes

- installed Python and required python packages on Data Node
- configured and deployed twitter harvester and data analysis component into a complete data pipeline solution

Part5: Flask API on Hosting Node

- installed Python, JavaScript, libraries and Flask API web framework on Hosting Node
- configured and deployed back-end, front-end Flask API application

5.1.2 Advantages of Ansible

Ansible tool was chosen because of the following three benefits:

- Documentation. Ansible can document the history of what was required to deploy a system. As a system administrator, dependencies need to be remembered for deploying servers. By writing down each step as lines of code, reproducing the same deployment is possible. However, some argue snapshot can also be a documentation tool. But snapshot has disadvantages because it only is the representation of the system at the time. In other words, a snapshot can restore the system, but it can not inform how to get to the current system or what has changed since the last stage. For this reason, we prefer Ansible in this project.
- Correctness. Manually deploy server is error-prone because of human errors. However, that is not to say human error does not occur in Ansible Playbook. Correctness means that a well-documented script (playbook) makes double-checking more convenient. It also allows colleagues to help and scrutinize the errors. Most importantly, just like a programming language, in Ansible, "try" and "catch" error handling syntax is available for use.
- Scalability. Ansible allows deploying repeatable tasks on different servers on a large scale. For example, as a System administrator, software packages are expected to update regularly. Without Ansible, manually log-in servers and type in the install/update command is the standard procedure. Consequently, it is not the most efficient use of time, or even impossible when the number of servers boosts. To solve this problem, the Ansible playbook offers automation to execute the same tasks on a large number of servers.

5.2 Container Technology - Docker

Containerization allows the virtual instance to share a single host OS, associated drivers and binaries. Each virtual machine includes complete disk OS and binary file, which caused duplication amongst virtual machines, it's a waste on the server resources, which can limits the number of virtual machine that a physical server can handle. In other words, if we see the virtual machine as a individual house, container are more like apartment built on the shared land and shared facility. Therefore, to certain extends, the container technology reduces the waste of resources.

In the real world, containers and virtual machines often live together, for example, in this project, one container is running inside of each virtual machine on MRC. For the purpose of scaling, it is possible to create more containers inside of a virtual machine.

We used docker container technology simply because it is the leading one with 79 percent of the market share, moreover, it was heavily taught in this subject.

5.3 Melbourne Research Cloud

UniMelb Research Cloud is a private cloud owned by the University of Melbourne, thereby sharing the same pros and cons of any other generic private clouds. Furthermore, OpenStack, a family of open source softwares, forms the bases of the Melbourne Research Cloud. As a result of this, benefits and shortcomings will be discussed in the following as well.

5.4 Pros and Cons as a private cloud

UniMelb Research Cloud shares three main advantages as a private cloud infrastructure.

- More Control. Most private cloud is closed under certain organizations. To certain extent, issues be discussed and dealt with internally e.g. disputes on decision-making and user level clearance. Therefore, it is easier to be controlled.

- Security. For the same reason, a private cloud is closed under the same organization. It also follows that the data will be securely stored and the data integrity can be trusted on the private cloud.
- Relevance. The Unimelb Research Cloud provides with research opportunities pertain to the core business of the University of Melbourne, that is, a leading research institution in Australia.

However, a drawback comes along these benefits.

- The University of Melbourne is liable for the total expense of the Unimelb Research Cloud. For example, the pay for cloud system admin team, hardware equipment fees, especially the landfill disposal fee after upgrading the hardware. Last but not least, initial planning is difficult. Because the hardware will be lock-in for the next few year, any future purchase for update will be thoroughly discussed and calculated according to various factors e.g. the expected computing power demands in the next few years, funding and budgets.

5.5 Pros and Cons as a Open Stack cloud

The Open Stack software offers three benefits for Melbourne Research Cloud

- open source. The Open Stack software and the source code is openly available to anyone and the program can be used for free without any limitations [**?**]. This allows modularity of the design, and making it more flexible for building Melbourne Reseach Cloud.
- Dashboard. The University Research Cloud integrated and customized the Open Stack Dash board into a visually-appealing GUI for the University of Melbourne.
- LightWeight OpenStack command line tool. However, there is also alternative to query and control your cloud resources without needing to use the browser-based dashboard. In your terminal, OpenStack provides command line clients that is designed to work with the OpenStack API[Unimelb Reference].
- Portability. The source code and the Ansible scripts of this project can run on any host that has OpenStack software. Therefore it prevents the lock-in effects that will otherwise be experiencing by using other industrial alternatives i.e. AWS EC2.

However, OpenStack has constraints on requesting resources.

- That is, you can't state the number of CPUs, memories, and storage for each virtual machine[9]. But this can be done differently. You can instead choose from a variety of flavors that prescribes the size of a VM[9]. All flavors on the Melbourne Research cloud come with 30GB storage and the different flavors provide different sizes of CPU power and memories[9].

6 Fault Tolerance

6.1 Set Up

6.1.1 CouchDB: Sharding

To handle database failure, at the set-up stage, the CouchDB cluster automatically split the documents in the database into different shards and distribute them to various databases. Because each shard may have more than one replicas, therefore, even if one node of the CouchDB cluster fails, the application can still get access to a shard replica in other nodes of the CouchDB cluster[13].

```
CONTAINER ID   IMAGE                                          COMMAND           CREATED          STATUS          PORTS                NAMES
520907ef2b50   cloud-assignment-2/couchdb-connector:latest   "python app.py"   13 seconds ago   Up 9 seconds    8000/tcp             couch_replicator.5.b4uktnystceu3ss7w15kzzp8
bdabc8c8bf06   cloud-assignment-2/couchdb-connector:latest   "python app.py"   13 seconds ago   Up 10 seconds   8000/tcp             couch_replicator.4.kt6t0fcdolcnrk24jkicqnm8o
36efad9f15d9   cloud-assignment-2/couchdb-connector:latest   "python app.py"   13 seconds ago   Up 9 seconds    8000/tcp             couch_replicator.2.2qoyckvheh1791q5qcoudheve
d24792359de9   cloud-assignment-2/couchdb-connector:latest   "python app.py"   13 seconds ago   Up 9 seconds    8000/tcp             couch_replicator.3.304wxf2118no7sceo5kk95r78
1ae787f586d0   cloud-assignment-2/couchdb-connector:latest   "python app.py"   13 seconds ago   Up 10 seconds   8000/tcp             couch_replicator.1.j2bzc57nprb41r0xp6r02jpw3
```

Figure 13: couchDB connector replicas

6.2 Single Point-of-Failure

6.2.1 Twitter Harvester: Credentials and Connections

These following steps is to ensure the connection to Twitter API, with the handling of the credential errors.

1. When connecting to Twitter API, if the authentication of the given credentials fails, then the program prints "connection " errors.

2. After printing the failure message, the program will try again to authenticate with the same credentials.

6.3 Connection between Flask API and CouchDB

Try and catch loops are written to handle and check session timeouts issues. For example, CouchDB connection. If the request call from Flask API to CouchDB has timed out, then aborted the call and tries to create a new connection to CouchDB then call API again.

7 Dynamical Scaliblity

7.1 Twitter Harvester: Dynamically data injection into CouchDB

A sleep timer of twitter harvester is listening to any new tweets every 15 minutes. The couchDB data is dynamically increasing while the server is running.

7.2 Docker Service: Scaling up the number of Containers

We have supported the option to scale up the number of docker containers on a docker node. This is done by using the docker service to create docker replicas. For example, in the Hosting Node, to handle the traffic when users increase, we created 5 replicas of the back-end containers i.e. couchdbconnector, to share the loads coming from the users when they fetch data. However, in this project, we did not deploy this approach because of the relatively small user pool. But it is ready to go and can be shown in in Figure 13.

References

[1] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1993.

[2] Albert Einstein. *Zur Elektrodynamik bewegter Körper*. (German) [*On the electrodynamics of moving bodies*]. Annalen der Physik, 322(10):891–921, 1905.

[3] Knuth: Computers and Typesetting,
http://www-cs-faculty.stanford.edu/~uno/abcde.html

[4] Martin Halvey and Mark T. Keane, *An Assessment of Tag Presentation Techniques* Archived 2017-05-14 at the Wayback Machine, poster presentation at WWW 2007, 2007

[5] Patrick Smyth, *"Creating Web APIs with Python and Flask"* The Programming Historian 7 (2018), https://doi.org/10.46430/phen0072

[6] J. Chris Anderson, Jan Lehnardt, Noah Slater, *"CouchDB: The Definitive Guide"*, January 2010 Publisher(s): O'Reilly Media, Inc, https://learning.oreilly.com/library/view/couchdb-the-definitive/9780596158156/ch03.html

[7] Ansible, Docs, User Guide, Intro to playbooks
https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html

[8] OpenStack, Documentation, Overview
https://docs.openstack.org/install-guide/overview.html

[9] UniMelb Research Cloud Documentation infrastructure, R., Services, R., services, O., Cloud, R. (2020). Research Cloud. Retrieved 28 April 2020, https://docs.cloud.unimelb.edu.au/training/concepts/

[10] 3.1. Design Documents — Apache CouchDB® 3.1 Documentation
https://docs.couchdb.org/en/latest/ddocs/ddocs.html

[11] 1.1. API Basics — Apache CouchDB® 3.1 Documentation
https://docs.couchdb.org/en/stable/api/basics.html?highlight=httpapi-basics

[12] J. Chris Anderson, Jan Lehnardt, Noah Slater, *"CouchDB: The Definitive Guide"*, January 2010 Publisher(s): O'Reilly Media, Inc, https://learning.oreilly.com/library/view/couchdb-the-definitive/9780596158156/ch04.html

[13] 4.4. Shard Management — Apache CouchDB® 3.1 Documentation
https://docs.couchdb.org/en/stable/cluster/sharding.html