Cluster and Cloud Computing Assignment 2- City Analytics on the Cloud

Team 43
Aidan McLoughney(1030836)
amcloughney@student.unimelb.edu.au
Thanaboon Muangwong(1049393)
tmuangwong@student.unimelb.edu.au
Nahid Tajik(1102790)
ntajik@student.unimelb.edu.au
Saket Khandelwal (1041999)
saketk@student.unimelb.edu.au
Shmuli Bloom(982837)
sbloom@student.unimelb.edu.au

July 19, 2020

Contents

1 Introduction

This report explains the cloud-based solution developed to use UniMelb Reasearch Cloud resources. The produced solution scales up/down and is run on the allocated instances for harvesting tweets from across the cities of Australia. Tweets are harvested by 4 different APIs such as Twitter API and Streaming and stored in the CouchDB database. We extract tweets by searching for a few keywords related to our scenarios. Also, various datasets availabe within the AURIN platform were considered alongside with the collected Twitter data to tell stories about trending topics and life in Australia. Therefore, a few analytic scenarios were implemented using CouchDB MapReduce capabilities and advanced Python plotting libraries to plot them. Finally, these scenarios and data were visualised using the front-end web application.

2 Project Management

Our team managed the development of this project and made peace with its tight deadline. We had weekly meetings to communicate about different parts of the project and update each other about the progress of it. Our team discussed about the system architecture, various technologies, scenarios and all challenges involved in this project. We used various project management tools such as Github and Backlog (Figure 1) to facilitate the development of the project and to keep track of tasks required to be completed by each team member.

2.1 Roles of Team Members

- **Aidan:**

  His role in this project was to design and manage the Ansible deployment scripts and the research cloud architecture. He organised the automated deployment and connecting of the CouchDB cluster, the tweet harvester and data analytics front end. He collaborated with Nahid to implement the CouchDB cluster setup commands. He collaborated with Saket to setup the automated deployment and testing of the harvester on the cloud resources. He collaborated with Thanaboon to setup the automated deployment of the data analytics front end.

- **Thanaboon:**

  His role in this project was to design user interface and develop the Front-End web application. He worked on organizing designing dashboard which help us to explain our scenario to the user, and using Nahid connector to get the data from the views and display it on the front-end. He collaborated with Saket and Shmuli to understand the scenario and which plots to show and in what order.

- **Nahid:**

  Her role in this project was to implement Python codes to connect with CouchDB and insert tweets to the database. She worked on the analysis of CouchDb views using Map/Reduce functions and created the Python connector which allows the front-end to fetch data from the views. She also collaborated with Aidan to implement CouchDB cluster setup commands using Ansible. This role required collaboration with Shmuli to understand scenarios.

- **Saket:**

  His role was design, implement and test the tweet harvesting component along with integrating data to the cloud system. He collaborated with Aidan to enable automated deployment and testing of the harvester on the research cloud, Shmuli to add the required analysis of the data for the scenarios to the harvester,and with Nahid for integration of CouchDB operations into the harvester.

- **Shmuli:**

  His role in this project was to devise scenarios and to prototype their implementation in Python. This role required collaboration primarily with Saket, who worked on the Twitter Harvester application, in order to communicate with Saket about what was required from Twitter. Collaboration was also required with Nahid, for her to understand what was required for the CouchDB views, and with

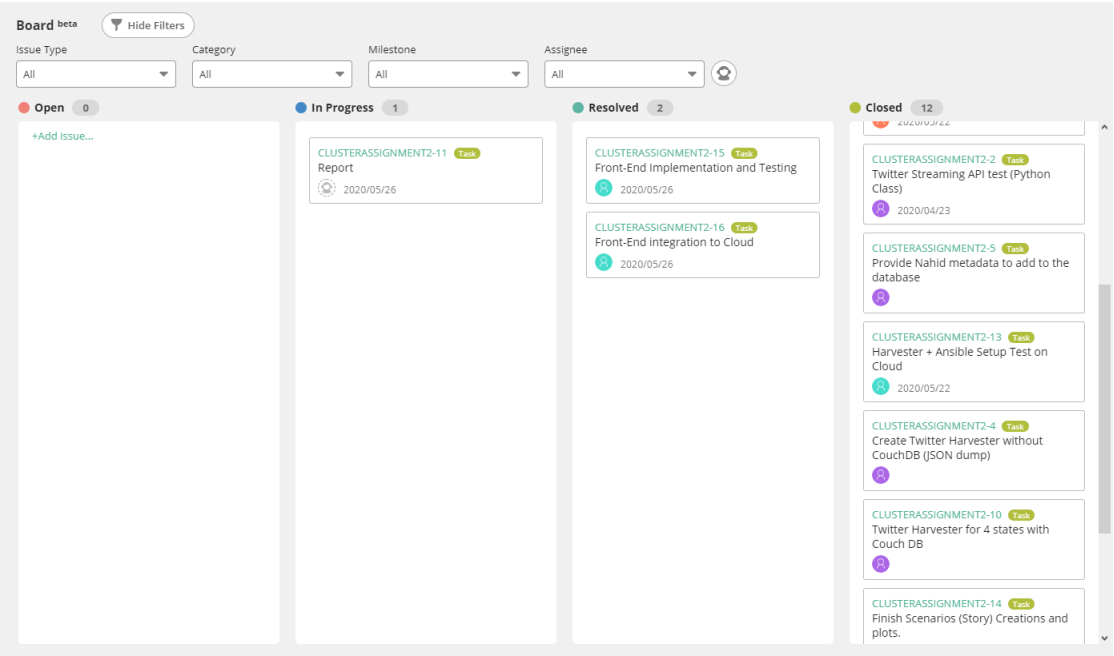


Figure 1: Project and Code Management Tool

Thanaboon to provide any details about the scenario implemenation that were required to build the front end web application.

3 System Architecture

One of our main focuses for our system design was to spread the workload evenly across the available cloud resources, and in turn reduce the number of single points of failure for the overall system. We also wanted to be able to run multiple Twitter harvesters that have separate and different configurations. We decided to run each of our application components inside Docker containers to reduce the need for dependency managements and setup configuration. After the system deployment, each instance is running identical resources and application components, excluding the harvester configurations, except that one of the instances also has the data analytics front end running as well.

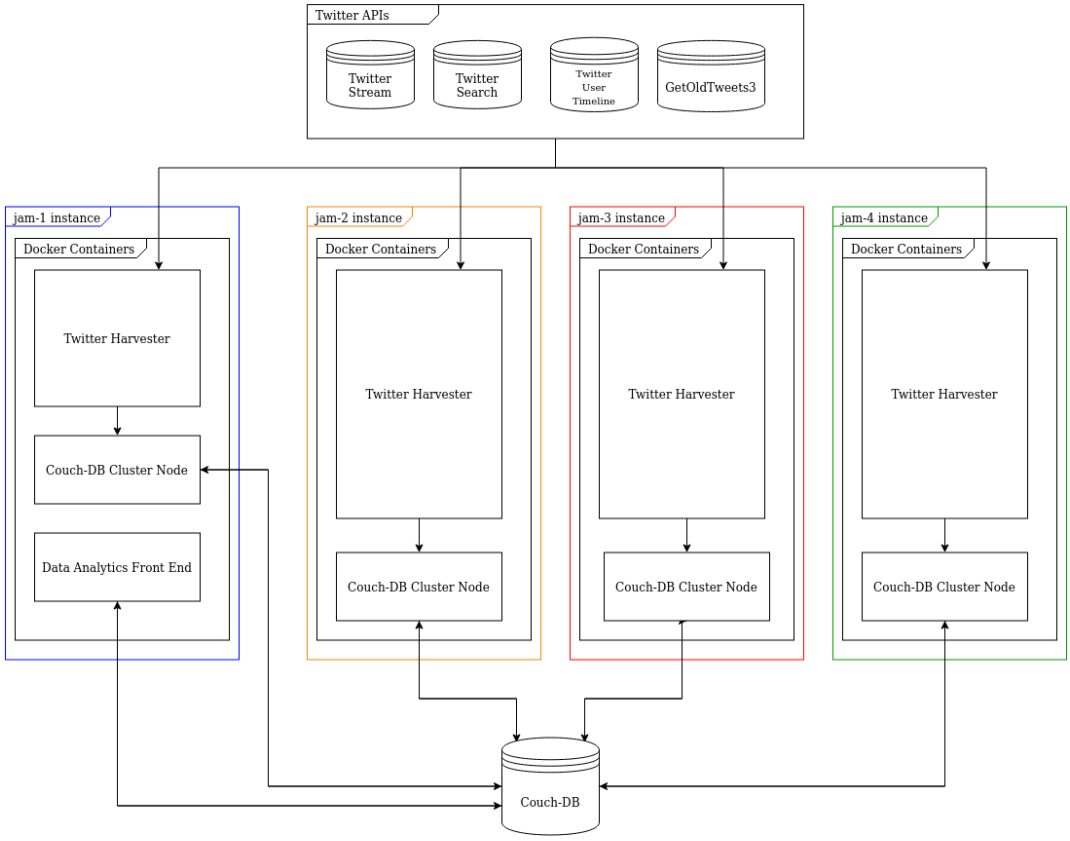


Figure 2: System Architecture

3.1 Scalability

Each of our system components were designed with scalability in mind. We have setup our main deployment script to be dynamic as possible where the user only has to designate the available/required resources, and the script will scale the application across the resources. We have also developed a scale up deployment script which will allow the user add additional resources to an existing cloud system. This script was built with the assumption it will be adding resources to a cloud system that was built with the full deployment script. The scale up script adds the new Couch-DB cluster nodes to the cluster but does not do any shard management, we decided we wanted a safer data management system rather than a more dynamic one with shard management.

3.2 Fault Tolerance

We have built in fault tolerance into each of our components in our cloud system. All of our docker container restart on failure, which means based on our testing, a cloud component will only unavailable if the instance has crashed. To minimise this threat to the data, we have designed two key approaches. Firstly, our Couch-DB cluster has a redundancy level of 3, which means that if one of our instances crashes, the data will still be available on two other nodes. Secondly, we have mapped the Couch-DB data folders inside our containers to separate volume resources, and have setup volume snapshots of this volume resources, so that in the worse case scenario where all of our instances crash and are unavailable, our harvested Twitter data will still be safe in the snapshots.

4 Deployment

4.1 Ansible

4.1.1 Cloud Infrastructure

The cloud Ansible playbooks were used to deploy/create all of the required cloud infrastructure on the Unimelb research cloud. These playbooks use the Openstack API to create the infrastructure on the cloud, majority of the playbooks are standard and are based on the COMP90024 workshop demos. The 'cloud-instance' playbook is the only noteworthy playbook, as it uses a zip loop to dynamically assign the assigned IP of the instances to different host groups so different Twitter harvester configurations can be set for different instances.

Figure 3: Ansible Full Deployment Playbook Structure

cloud-common: This role setups the required software dependencies for the local-host to run the playbook.

cloud-volume: To minimise data loss for our system, the 'cloud-volume' mount two volumes, one for the general docker system, and the other for the Couch-DB data folder. The purpose of this is to have persistent data storage in case the Docker containers fail.

cloud-security-group: The purpose of this role was to dynamically create the security groups for ssh access to the instances, and open up the required ports for the Couch-DB cluster.

cloud-instance: This role creates the Ubuntu VMs, and links them with the matching set of volumes. We found it was easier and more efficient to dynamically collect the assigned IPs and store them in host groups. We have a general host group called "Play", and individual host groups for each instance, e.g. the jam-1 instance belongs to the "Uno" host group. Making a host group for each instance allowed us to easily set

different configurations for the Twitter harvester, and also allowed us to scale up the different host groups based on the required load for each type of harvesting. We accomplished this dynamic group assigning with an Ansible zip loop, which allows us to set a list of assigned host groups for the list of IPs set/given by the Unimelb research cloud.

4.1.2 Dependencies

dep-common: This role sets up the required proxies for the instance, as the instances assigned do not have a public IP, and it also installs software dependencies for the applications.

dep-volumes: The purpose of this role is to create the volume directories and then mount the volume resources to their matching instances. This is important because it provides required additional data storage for our Couch-db cluster.

dep-docker: This role installs the Docker software dependencies, setups the apt Docker repository, installs Docker and Docker compose via Apt and Pip, and setups the proxy settings due the instance's non-public IP.

dep-couchdb: This role uses a Couch-DB Docker image to setup the Couch-DB software on each instance. It setups the admin username and password, opens up the relevant ports for access, and sets the cookie for the cluster setup.

4.1.3 Application

couchdb-cluster The Couch-DB cluster was setup by this role; we used the Couch-DB APIs and curl commands to add each non-master node to the cluster. The jam-1/Uno instance is our "master-node", it added each of the other instances, completed the cluster setup, created the required databases and the required view.

twitter-farmer This role loads the Twitter harvester source files onto each of the instances, dynamically sets the harvester configuration files based on the host group, and creates and starts the harvester Docker container.

4.1.4 Scale Up Script

The scale up script uses a different shell script (run-jam-scale-up.sh), and different host variable files (necter-scale-up.yaml, jam-scale-up.yaml). The scale up script playbook reuses the roles and plays from the main deployment playbook but uses a different Couch-DB cluster playbook, which just adds the new node to the existing cluster.

couchdb-cluster-scale-up This role is setup to add a new Couch-DB node to an existing Couch-DB cluster. It uses a pre-defined IP variable for the master-node of the existing Couch-DB cluster.

4.1.5 Volume Snapshot Script

We are using the volume mounting and volume snapshots to protect the data we have collected from Twitter, therefore we need to create the volume snapshots after the Twitter harvesting is done. So we have split the volume snapshot Ansible playbook from the main deployment playbook, so we could run it separately once the harvesting is done.

cloud-volume-snapshot: The volume snapshots setup by this role are used to minimize the amount of data lost due to technical problems for the volume.

4.2 Twitter Harvester Configuration

One of our key design ideas for our overall architecture was to setup dynamic configuration for the Twitter harvester Docker containers. The basic idea was to have multiple Twitter harvesters running at the same time but with different search parameters and requirements. We accomplished this by having multiple Ansible host groups, where each host group has different Twitter harvester configurations. We also dynamically setup the Couch-DB endpoint node for the harvester, so that the spread of Couch-DB API calls were spread across the nodes.

```
- name: Load Twitter Docker Files
  synchronize: src=harvester_files dest=/tmp

- name: Add Python Twitter Config
  become: yes
  lineinfile:
    path: /tmp/harvester_files/Dockerfile
    insertafter: FROM python:3
    line: ENV CONFIG_FILE {{ py_twitter_config }}

- name: Add Couchdb URL to Config
  become: yes
  lineinfile:
    path: /tmp/harvester_files/config/{{ py_twitter_config }}
    insertafter: "['couch']"
    line: URL = http://{{ couchdb_user }}:{{ couchdb_password }}@{{ ansible_default_ipv4.address }}:{{ couchdb_port }}/
```

Figure 4: Docker Dynamic Configuration Script

4.3 Couchdb

4.3.1 Installation and Configuration

We explored using Docker containers for our Couch-DB cluster, and explored building the Couch-DB software from the source code. We found that building Couch-DB from source was quite tedious, and there was clashed between the required versions of software dependencies and our selected Ubuntu version. Using the recommended Couch-DB image significantly reduces the amount of dependency setup and Couch-DB configuration. Therefore we decided to run our Couch-DB cluster nodes inside Docker containers to reduce the amount of required setup, and reduce the complexity of our solution.

4.3.2 Cluster Setup

Since we decided to run our Couch-DB nodes inside Docker containers, we used the cluster configurations from the Couch-DB tutorial as a basis for our configuration. We used Ansible to dynamically populate the variables, to minimize errors if we needed to adjust the configuration variables, e.g. the admin password. Following the same logic as the demo session, we treated one of our nodes as the "Master node", the first node in group "Uno", and had that particular node loop through and add all of the other nodes. We accomplished this by setting a conditional loop to go through the over-arching host group "Play". The commands in Figure 5 have been adjusted to fit in the screenshot.

```
- name: Enable Cluster
  shell: 'curl -XPOST "http://{{ couchdb_user }}:{{ couchdb_password }}@{{ groups["Uno"].0 }}:{{ couchdb_port }}/_cluster_setup"
         --header "Content-Type: application/json" --data "{\"action\": \"enable_cluster\", \"bind_address\":\"0.0.0.0\",
         \"username\": \"{{ couchdb_user }}\", \"password\":\"{{ couchdb_password }}\", \"port\": \"{{ couchdb_port }}\",
         \"remote_node\": \"{{ item }}\", \"node_count\": \"{{ couchdb_node_number }}\", \"remote_current_user\":\"{{ couchdb_user }}\",
         \"remote_current_password\":\"{{ couchdb_password }}\"}"'
  args:
    warn: yes
  loop: '{{ groups["Play"] }}'
  when: item != groups["Uno"].0


- name: Add Nodes to Cluster
  shell: 'curl -XPOST "http://{{ couchdb_user }}:{{ couchdb_password }}@{{ groups["Uno"].0 }}:{{ couchdb_port }}/_cluster_setup"
           --header "Content-Type: application/json"  --data "{\"action\": \"add_node\", \"host\":\"{{ item }}\", \"port\": \"{{ couchdb_port }}\",
           \"username\": \"{{ couchdb_user }}\", \"password\":\"{{ couchdb_password }}\"}"'
  args:
    warn: yes
  loop: '{{ groups["Play"] }}'
  when: item != groups["Uno"].0


- name: Finish cluster Setup
  args:
    warn: yes
  shell: 'curl -XPOST "http://{{ couchdb_user }}:{{ couchdb_password }}@{{ groups["Uno"].0 }}:{{ couchdb_port }}/_cluster_setup"
           --header "Content-Type: application/json" --data "{\"action\": \"finish_cluster\"}"'
```

Figure 5: Couch-DB Cluster Setup Script

4.3.3 Database and View Initialization

After completing the CouchDB cluster setup, we created tweets database with 8 shards and 3 replicas using Ansible code. Then, the design document (Figure 6) which contains views was copied to the server in order to be built in the tweets database. Therefore, the front-end is able to fetch data from the views. Views were deployed in the tweets database using the Ansible code (Figure 7).

```
{
 "_id": "_design/grp43",
 "views": {
  "scomo": {
   "map": "function (doc) {
        regexp = /scottyfrommarketing|scottmorrisonmp|scomo|scott morrision|scottmorrison|scomoresign/
        var cities = ['sydney','melbourne','brisbane','perth','adelaide','hobart']
        text = doc.full_text.toLowerCase()
        for(i=0; i < cities.length; i++){
          if(text.search(regexp) != -1 && doc.city.indexOf(cities[i]) != -1){
            emit(doc.created_at, [doc.city, doc.sentiment.compound,doc.user.id]);}}}"
  },
  "corona": {
   "map": "function (doc) {
        var cities = ['perth','sydney','melbourne','adelaide','brisbane']
        for(i=0; i < cities.length; i++){
          if(doc.keywords.includes('coronavirus') && doc.city.indexOf(cities[i]) != -1){
            emit(doc.created_at, [doc.city, doc.sentiment,doc.hashtags,doc.user.id]);}}}"
  },
  "politics": {
   "map": "function (doc) {
        if (doc.coordinates != null && doc.keywords.includes('politics'))
          emit(doc.created_at, [doc.city, doc.full_text, doc.sentiment.compound, doc.user.id, doc.coordinates]);}"
  }
 },
 "language": "javascript"
}
```

Figure 6: Map Functions Implemneted in CouchDB

```
- name: Copy view
  become: yes
  copy:
    src: ./couchdb_files/grp43View.json
    dest: /data/couchdbViews

- name: create view in tweets database
  args:
    warn: yes
    shell: 'curl -X PUT http://{{ couchdb_user }}:{{ couchdb_password }}@{{ groups["Uno"].0 }}:{{
     couchdb_port }}/twt_db/_design/grp43 --data-binary @/data/couchdbViews/grp43View.json'
```

Figure 7: View Deployment Using Ansible Commands

4.4 Error/Challenge Handling

There were many issues and learning curves faced while completing the deployment and architecture, the below issues were the key issues that required the most hours to solve. We also really appreciate all the

help fellow students, the tutors and Professor Richard Sinnott provided via the discussion boards.

4.4.1 Different Search Parameters for Twitter Harvester

One of our key design ideas was to have different Twitter harvester searching and harvesting different Twitter data but because we are running each harvester inside a Docker container, we needed to provide different configurations when building the Docker image. So we decided to set up different configurations for each of the cloud instances but it wasn't clear how to evenly and dynamically distribute the configurations across different resources architectures. We worked that we could use an Ansible zip loop dynamically distribute the assigned IP addresses across different host groups, and then assigned different configuration variables for each host group.

4.4.2 Couch-DB Volume mount Issue

We had a few issues while setting up the Couch-DB cluster on the research cloud, the issues mainly stemmed from our attempts to create the Couch-DB node container on a volume mounted directory. We realised that we needed to have data persistence for the Couch-DB cluster but mapping the entire Couch-DB folder caused issues with Docker managing the ini file. This was solved by just mapping the Couch-DB data folder. After solving the mapping error, the Docker daemon faced permission errors even when the Docker service created the mapped folder in the volume. We had to solve this by creating an Ansible task to adjust the permissions of the folder such the Docker service had permission to write to the folder.

4.5 Unimelb Research Cloud

As requested in the project spec, we discuss the benefits and issues of using the Unimelb Research Cloud in this section.

4.5.1 Pros of Unimelb Research Cloud

- Unimelb Research Cloud being built with Openstack, allows for easy and straight forward deployment with the Openstack API.
- Setting up security groups and rule is intuitive and straight forward.
- Creating and managing volume snapshots is effective and easy to manage.

4.5.2 Cons of Unimelb Research Cloud

- Other team member's key-pair is not visible or available when creating instances inside the Unimelb Research Cloud group project. Which means we have to either add it manually to the instance or our research cloud account.
- Not having public IP addresses, which is understandable given the circumstances, causes multiple minor issues with proxies and VPNs.
- We found that the Unimelb research cloud dashboard to be quite slow, and often unresponsive.

5 Tweeter Harvester

Harvesting Tweets was one of the most crucial tasks in order to create a variety of social media analytics. The goal of the harvester is to harvest a huge number of tweets across Australia utilising the various Twitter API endpoints available.

5.1 Considerations

5.1.1 Tweet Redundancy

According to the specification file, the system should be able to cope with duplicate tweets. This was easily handled by using the unique id provided by Twitter as the key while adding tweets to the CouchDB database. This ensured the uniqueness of each CouchDB document.

5.2 Data retrieval endpoints

Table 1 shows the limitations of 4 API endpoints considered for gathering tweets.

Table 1: Limitations of 4 API endpoints

End point	Description	Limitation
Twitter Stream API	A real-time stream endpoint of tweets provided by twitter.	Rate Limited
Twitter Search API	An endpoint returning a collection of tweets matching a specified query.	Rate Limited and only previous 7 day window.
Twitter User Timeline API	An endpoint to retrieve tweets by a specified user.	Rate Limited and a maximum of 3200 tweets allowed per user.
GetOldTweets3 API	A Python library to retrieve tweets older than a week.	

All the Twitter endpoints enforce retrieval rate limitation which is a set number of requests per 15-minute window.

5.3 Location Requirement

The goal of this assignment is to create social media data analytics scenarios with tweets across Australia. In order to meet this requirement, we retrieved tweets with location available. Based on the Twitter's documentation [Developer, 2020], location of tweets is determined by the 'Coordinates' field which is a GeoJSON field consisting of the latitude and longitude.

5.4 Maximum Efficiency Method

We realised that the use of Twitter's Search and Stream API was not enough to harvest a large amount of tweets in the given time frame of the project due to restrictions made on the endpoints. Therefore, we used additional endpoints such as User Timeline and Followers API to maximise the efficiency of gathering a significant amount of tweets.

5.5 Exception Handling

To have a robust harvester, we have handled all types of exceptions including normal python exceptions and specific ones caused by the Twitter endpoints. The error handling incorporates Tweepy's (Python Twitter library) module exceptions and the harvester proceeds after experiencing these errors.

5.6 Harvester Pipeline

The harvester pipeline consists of various components which are described below. The figure 8 shows a high level overview of the components process and a detailed flow of the process is illustrated in the figure 9.

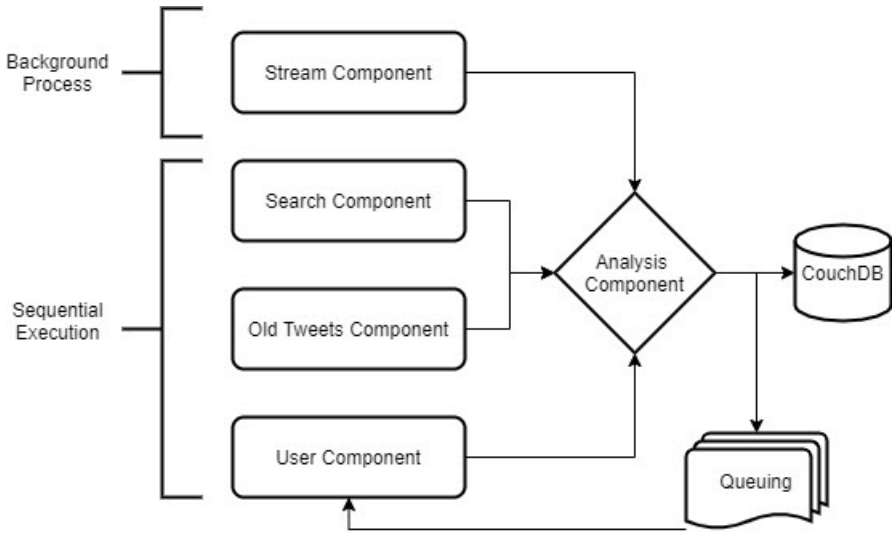


Figure 8: High level overview of the harvester components

5.6.1 Streaming Component

Streaming component collects tweets from the real-time Twitter Stream API based on a specified bounding box (location area) varying with each instance of the harvester in multi-instance deployment case.

5.6.2 Search Component

The harvester makes use of the Twitter Search API to gather a collection of tweets based on a specified query which is handled by this component. It extracts tweets by specifying the location radius and search terms.

5.6.3 Old Tweets Component

We extracts older tweets through the GetOldTweets3 Python package [Mottl, 2020] , which is not accessible by the available Twitter endpoints due to the 7-day time constraint imposed. GetOldTweets3 Python searches the oldest tweets by mimicking the behaviour of the scroll loader to accumulate more tweets. The location radius and search-terms were provided to query the older tweets.

5.6.4 Analysis Component

All tweets gathered from the above components are sent to the analysis component. The harvester extracts relevant tweets meeting the requirements of our scenarios by analysis component. These requirements guarantee that a tweet comes from the desired location and contains relevant information concerning the scenario topics. To handle redundancy, we check the existence of the processed tweets in the database before inserting them.

Since deciding scenarios were dependent on the data we possess, all the tweets satisfying either of the above requirements are added to the CouchDB database. If a tweet meets both requirements, it is inserted to the database with an additional field specifying that the tweet is relevant. Then we add the user who tweeted the relevant tweet to a priority queue to be further processed by the user component as shown in the flow chart above.

5.6.5 User Component

The user queue is processed by extracting all tweets for each of the users using the Twitter User Timeline endpoint. Then, we pass all extracted tweets to the Analysis component for further processing. In order

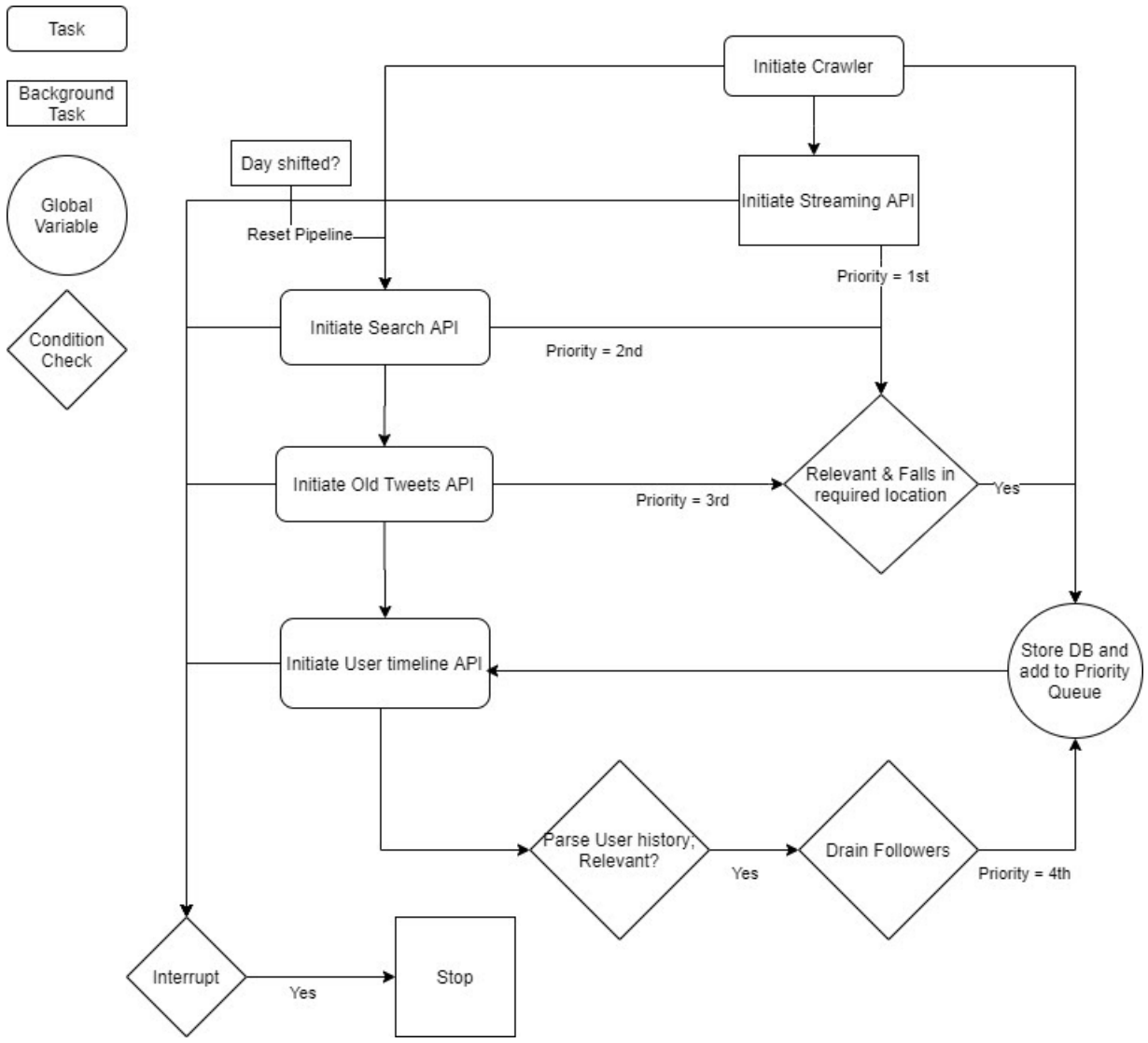


Figure 9: Detailed flow of all conditions and processes

to deal with the rate limitation made by Twitter, we expanded the search space by accessing the followers of the users who possess relevant tweets and adding them to the priority queue as we see in the diagram above.

5.6.6 Tweets Storing

As mentioned in the considerations section before, we store tweets in the database using the unique tweet identification provided by twitter. Although the actual tweet object consists of many fields, the harvester only stores selected fields required by the system and analytics component.

5.6.7 Configurations

As we harvest tweets on multiple instances, it was essential to design the harvester in a way that maximises the utilisation of the endpoints and not harvest redundant data. We have configured the harvesters in a way that each instance harvests from specific regions of Australia as the location parameters are set for each of Twitter's endpoints.

5.7 Challenges

5.8 Search API Exhaustion

With the 7-day limit enforced by Twitter on the Search API endpoint, the harvester exhausts the API for the given period and starts draining the user queue for potential tweets and followers. However, the 7-day window shifts every day, to address this the harvester resets its pipeline by initialising the search component every day to gather tweets for that day.

5.9 Persistent Processing History

Harvester keeps track of users in the queue which have been processed to mitigate the possibility of processing them again. This operation is not persistent as it stores the list of processed users in memory as the list can be refreshed every time the harvester is redeployed or rerun. Also, the processed user list is stored separately in each harvester in multiple instances environment. This leads to the possibility of processing the same user over and over. To tackle this issue, we store processed users in the database.

We check if the user exists in the processed user database whenever the harvester processes a user to drain his related tweets. This prevents wasted processing time as well as the data loss when the harvester is reset.

5.9.1 Set Limits

It would be ideal to process the whole user timeline or all followers to collect their tweets, but some users may have a huge number of followers, for example, celebrities. This makes it hard to collect all their tweets regarding the limitations made by Twitter API. To handle this issue, we decided to check for a certain set of tweets to make sure that they are relevant and skip the rest if tweets are irrelevant. The same concept applies to followers and a limit has been set for acquiring the list of followers. The applied limitations enable the harvester to process and gather tweets efficiently.

6 Social Media Analysis

6.1 Introduction

In this assignment we are required to harvest tweets from across the cities of Australia and undertake a variety of social media analyses that tell interesting stories of life in Australian cities. We are also required to compare the twitter data to the data available within the AURIN platform.

6.2 Approach and Limitations

To tell meaningful stories we reasoned that collecting a large amount of data is required. We were also interested in performing spatial analysis at a fairly granular level, so ideally we would be able to find a large quantity of location tagged data. Since the proportion of exact location-tagged tweets on twitter is generally very low, we believed that choosing topics for our scenarios that would allow us to obtain a very large number of tweets would improve our chances of being able to undertake this spatial analysis. Therefore, we searched for trending topics on twitter, first through searching for trending hashtags and then by also including keywords that commonly co-occur with those hashtags. Commonly occurring hashtags were, auspol, scomo, scottmorrison, scottyfrommarketing, australiaburns, climateemergency, coronavirus, covid19 as well as well as a lot of other variations on coronavirus/covid. Keywords that we also searched for included, ”election”, ”vote”, ”politics”, ”nswpol”, ”vicpol”, ”qldpol”, ”sapol”, ”wapol”, ”rubyprincess”, ”climatechange”.

Despite considering trending topics we still struggled to acquire a huge amount of data. As discussed above, the twitter streaming and search APIs could only pull real time data and data from the past week, respectively. As such we could only acquire older data using the Twitter User Timelines API and to some extent using the GetOldTweets3 Python package. Although these did allow us to mine some older tweets,

we were still restricted in the amount of historical content that we could obtain. We were provided with historical twitter data by the COMP90024 teaching team, however this data is quite old and we were interested in more topical analysis.

Given the spatial nature of the AURIN data, we were particularly interested in differences in interests, attitudes and sentiment across different scales of location in the twitter data. To capture data in major Australian cities, we specified location parameters in our queries to the twitter APIs. Other than restrictions on mining twitter content that limited the volume of content that we could mine, especially of historical tweets, the lack of available exact location-tagged tweets proved to be a major limitation on the types of analyses that we could undertake. For two out of three analyses/scenarios that we performed, we compared major Australian cities rather than more geographically specific areas. For the third scenario, we did compare the suburbs of Melbourne. However, the amount of data used for that analysis was smaller by a factor of 20 to 50 times.

The other challenge that we faced in undertaking analysis was performing natural language processing to gain an understanding of the content of a huge number of tweets in an automated fashion. We attempted to determine the content of tweets based on co-occurring hashtags with the trending hashtags, but these tended not to shed any light on the specific contents of the tweets. Similarly, looking at frequently occurring keywords, both 1 gram and 2 grams, was not helpful either as the keywords tended to be common English words rather than indicate the topic. Ultimately, we used the Python sentiment analysis library vaderSentiment to determine the overall sentiment of the tweets in all our scenarios as a means of learning about the tweets. We then compared this sentiment towards particular topics across locations.

6.3 Final Scenarios

The scenarios that we decided on, ultimately, are based on trending topics, namely coronavirus and politics. We analyse both general political tweets and tweets about Scott Morrison specifically, as he is a particularly hot political topic. We chose these topics because we were able to acquire a sizable number of recent tweets for all of them and therefore could be more confident in our analysis. We were able to pull in the order of 500,000 tweets about coronavirus and 100,000 tweets about Scott Morrison and almost 2 million political tweets, of which about 4000 were tagged with exact locations in Melbourne.

In order to tell richer stories about scenarios, we augment our sentiment analysis with data sourced from AURIN. Specifically, we use SA4 level aggregated demographic data for the Scott Morrison and coronavirus scenarios because for those scenarios we are comparing tweets across cities. We use SA2 level aggregated demographic data for our general politics scenario because there we are comparing twitter data across selected suburbs of Melbourne. Because each city contains multiple SA4 areas, we aggregate the AURIN data to obtain data at the city level in each of the first two scenarios.

6.4 Implementation

For all our scenarios, we use the Python API for Plotly, the advanced plotting library that uses JavaScript under the hood. This allows us to produce visually appealing plots and uncover interesting relationships in the data.

6.4.1 First Scenario

For the first scenario, we analyse the sentiment variations of Scott Morrison tweets across cities and over time (Figure 10). The Scott Morrison tweets acquired date back 10 years, but we restrict our analysis to tweets created after the 1st Sep 2018 (because this period is recent and the vast majority of the tweets were created after this date). In the front end, we provide the user with a drop down menu to select which demographic information from AURIN that they would like to compare the Scott Morrison sentiment with. The options provided are, median weekly rent, median monthly mortgage repayment, average household size and average number of persons per bedroom and a few different types of median weekly income choices.

6.4.2 Second Scenario

For the second scenario, we analyse the sentiment of coronavirus related tweets. We similarly analyse the variations in the sentiment of coronavirus tweets both across cities and over time (over weeks since the start of the pandemic). For this scenario we consider the coronavirus sentiment vs. percentage of the population with internet access (Figure 11).

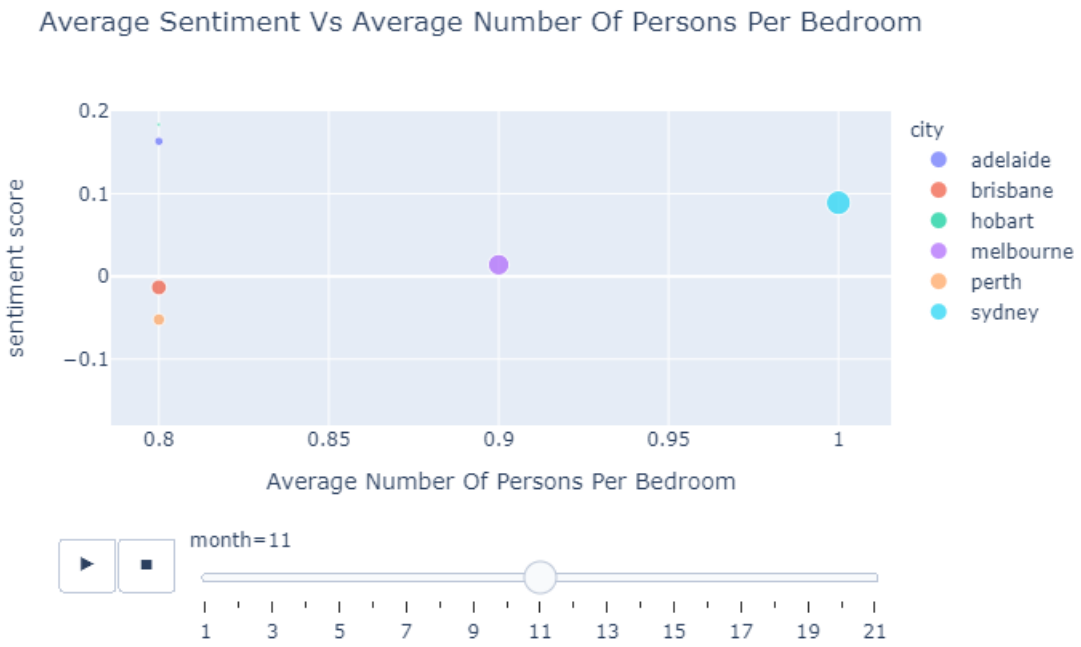


Figure 10: Scott Morrison sentiment vs. Example demographic variable

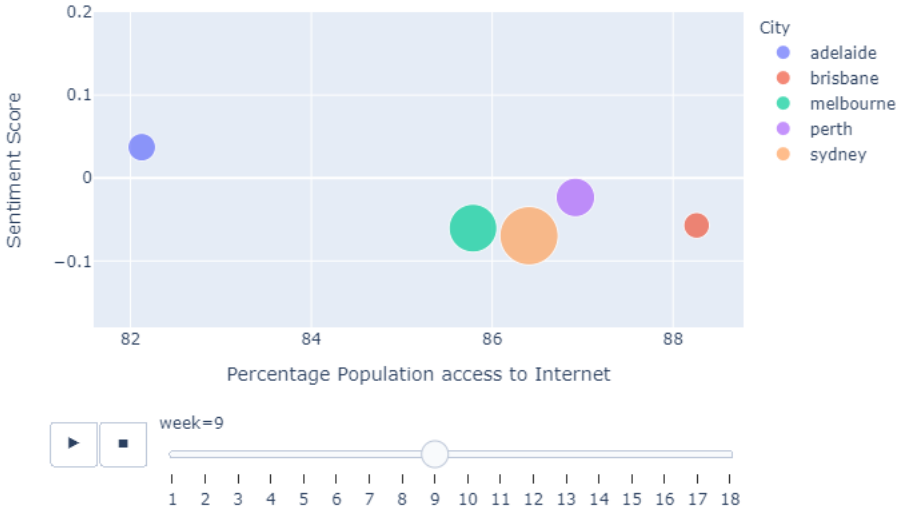


Figure 11: Coronavirus sentiment vs. Percentage of the population with internet access

6.4.3 Third Scenario

For the third scenario, we analyse average sentiment for general political tweets across suburbs of Melbourne. Only those suburbs which had at least 25 associated tweets were included in our analysis. Due to a lack of plentiful location tagged tweets, most suburbs in Melbourne were excluded and even among those which were included, many of them did not exceed the threshold for inclusion by much.

For this scenario, we are able to produce a more granular spatial analysis. We produce a choropleth map (Figure 12) where individual polygons are for selected suburbs of Melbourne. The colour on the map represents the average sentiment of political tweets for that suburb. We also overlay markers for each tweet with hover boxes containing the full text of the tweets.

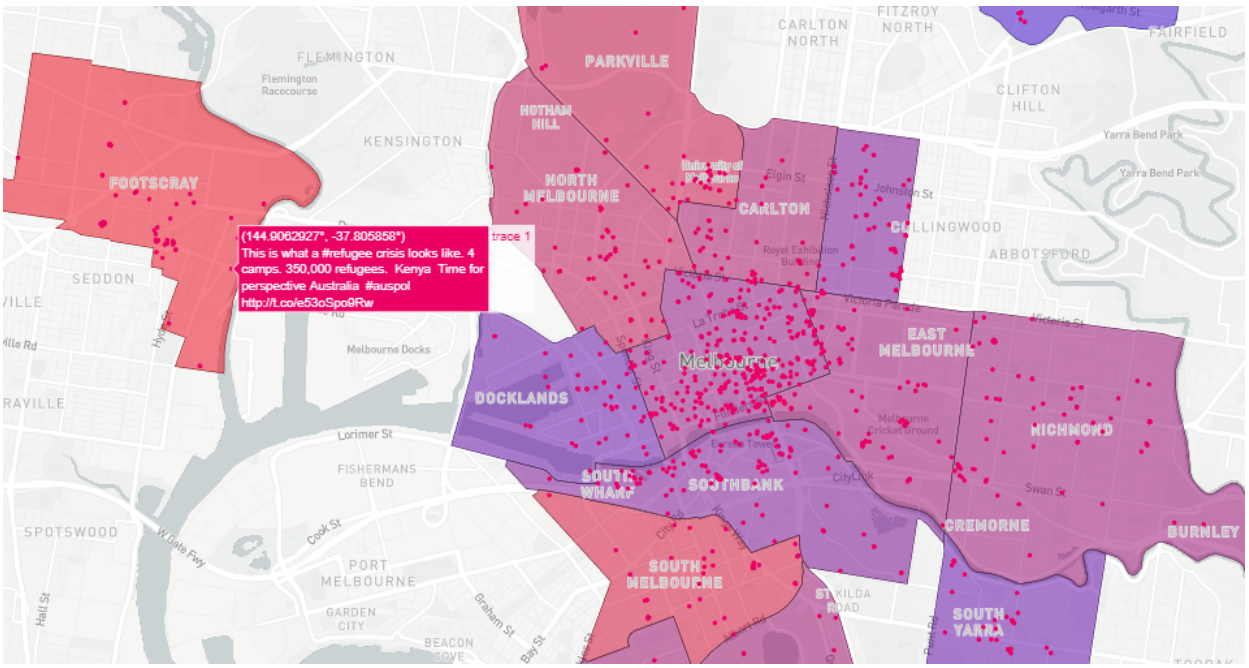


Figure 12: Average political sentiment across selected suburbs of Melbourne

Accompanying the choropleth map, which provides a spatial distribution of political sentiment in Melbourne, we produce a scatter plot (Figure 13) indicating the correlation between political sentiment and median weekly rent in the suburbs of Melbourne. We selected this particular demographic variable as it shows the strongest correlation with political sentiment. We cautiously speculate that the moderately strong positive correlation observed indicates that those who live in nicer dwellings are inclined to generally be more positive.

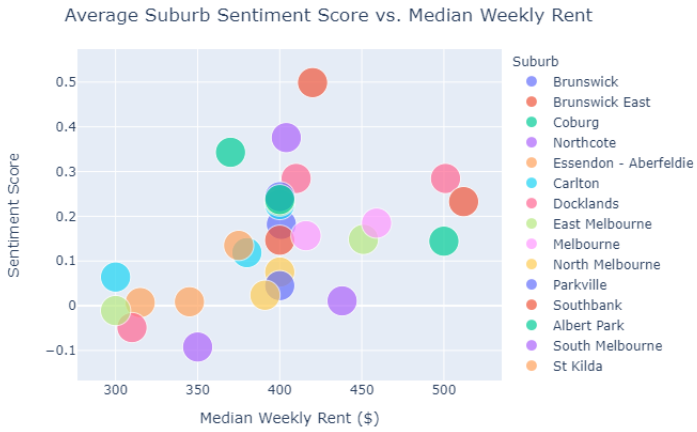


Figure 13: ”Average political sentiment vs. Median weekly rent

We also include a pie chart to visualise the distribution of political tweets across suburbs (Figure 14) and an annotated timeline (Figure 15) of the number of monthly political tweets. As can seen from the pie chart, the vast majority of tweets occur in and around the CBD. The annotated timeline, interestingly, but not surprisingly, has peaks around key political events, such as state or federal elections.

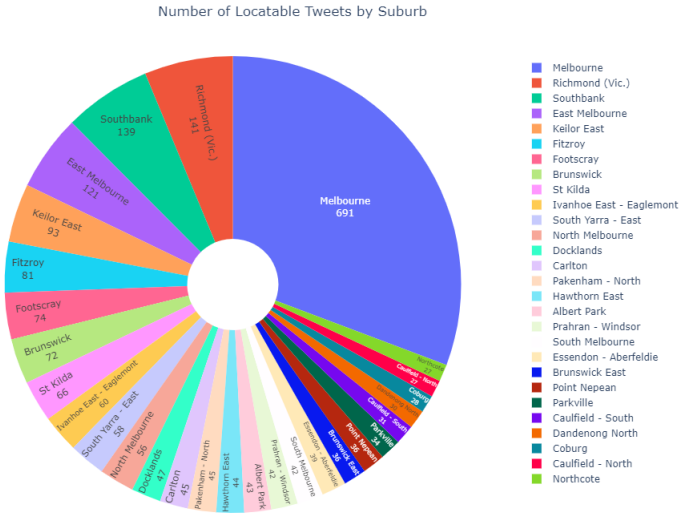


Figure 14: Distribution of political tweets across suburbs of Melbourne

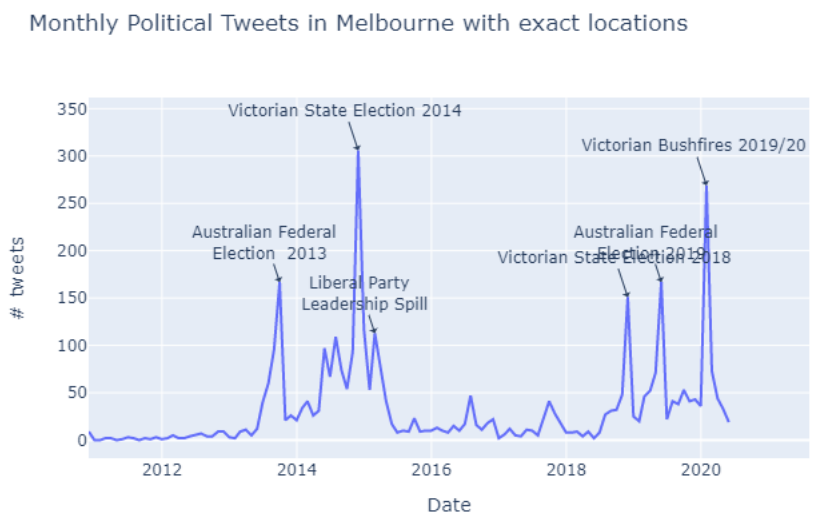


Figure 15: Timeline of political tweets in Melbourne

7 Front End

In the front-end application, we used Dash by Plotly as our visualization tool. Dash by Plotly is an open-sourced library python framework for building web applications. It is built on top of Flask, Plotly.js, React, and React js. Dash allowed us to build an interactive dashboard by using only Python. This was convenient for us because the analysis of the scenario has already been done using Plotly. Therefore, using dash as front-end simplifies this. Furthermore, Plotly can also incorporate the Bootstrap CSS for styling our application.

7.1 Interface

Our web application has three tabs for easy access to view the scenarios. As shown in figure 16, the user can switch the tab according to the scenario. In each tab, we included the number of tweets and the number of users that were used in analysis of each scenario. Furthermore, the objects displayed on the website. Users interacting with it can export the image from the plot in the period that the user is interested and can select the desired area of the plot by simply using a drawing rectangle by mouse gesture on the plot. The styling used in our application is from Plotly gallery oil and gas example [Shammamah, 2019]. The reason for this selection was because of its simplicity and the styling was sufficient for our application.

7.2 Layout

We used the container to store our objects. This helps us to make it more readable and more flexible. We set the number of 12 columns width in our front-end. Columns are the unit of length in the display which we can specify how wide our objects should be. This is to contain the object not to stray out of the container. The width and height of each plot is auto-resized based on the user display.

7.3 Getting Data

We used CouchDB python package to communicate between the database and our front-end. The plot is pre-processed from the data before displaying all the plot such that it will not have latency. All of the pre-processed data is store in the memory so when the user changes the tab or selects another AURIN data to correlate it will not process the data again.

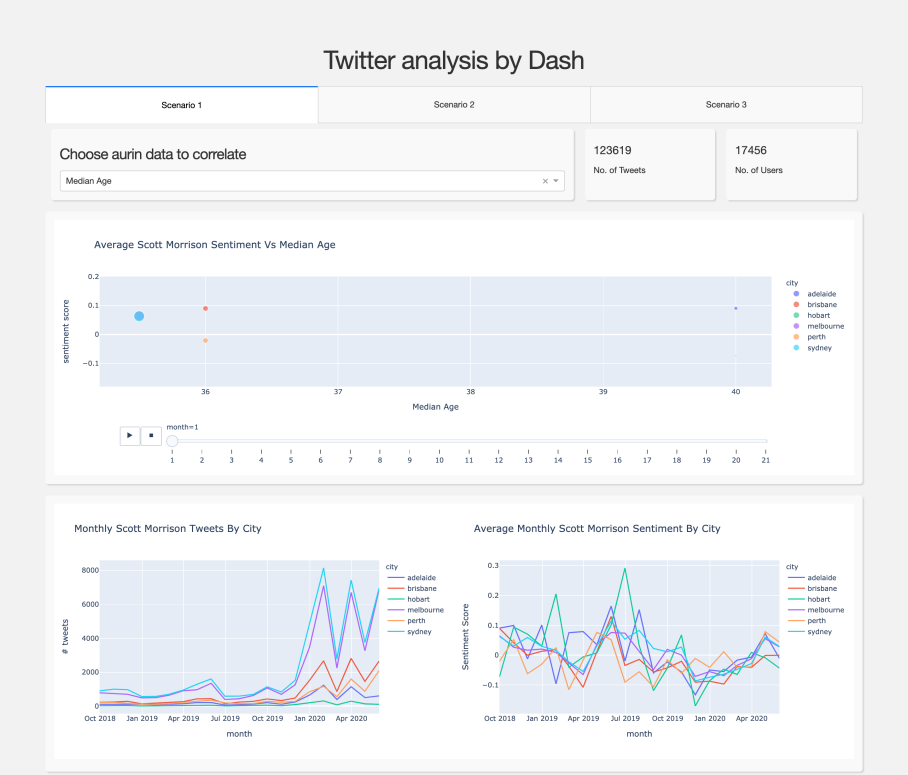


Figure 16: Scenario 1

7.4 Scenario 1

In scenario 1 page (Figure 16), there are 3 plots and a drop-down of AURIN data in which the user can select which data to correlate with. The Average Sentiment Vs AURIN variable can be changed by the user. The plot also provides the sentiment score over time which the user can press the play button to see the score changes over time. The two plots at the bottom of the dashboard show the amount of Scott Morrison tweets from each city for a period of time and the sentiment score for Scott Morrison of each city. In the first plot, the user can use mouse over the point in the plot to reveal different attributes such as city, month, median age, sentiment score, and the number of tweets related to that point. The same happens for the second and the third plot.

7.5 Scenario 2

In scenario 2 page(Figure 17), there are 2 plots. The sentiment score from each city with respect to the percentage of the population on the internet and bar graph of hashtag count from twitter. Users can press the play button to see the sentiment score changes over time.

https://www.overleaf.com/project/5ec89a3210b3e50001349f0f

7.6 Scenario 3

In scenario 3 page(Figure 18), there are 4 plots. On this page, we display a choropleth map analysis, the average suburban sentiment against the weekly rent, pie chart of tweets that can be located by suburb, and monthly political tweets in Melbourne with an exact location.

The color of regions in the choropleth map represent the sentiment score of each suburb in Melbourne. The color range from red to blue in which red represents a low score and blue is a high score. The point in this map represents the tweet in which the user can go over and see the tweet as in this figure(Figure 19).

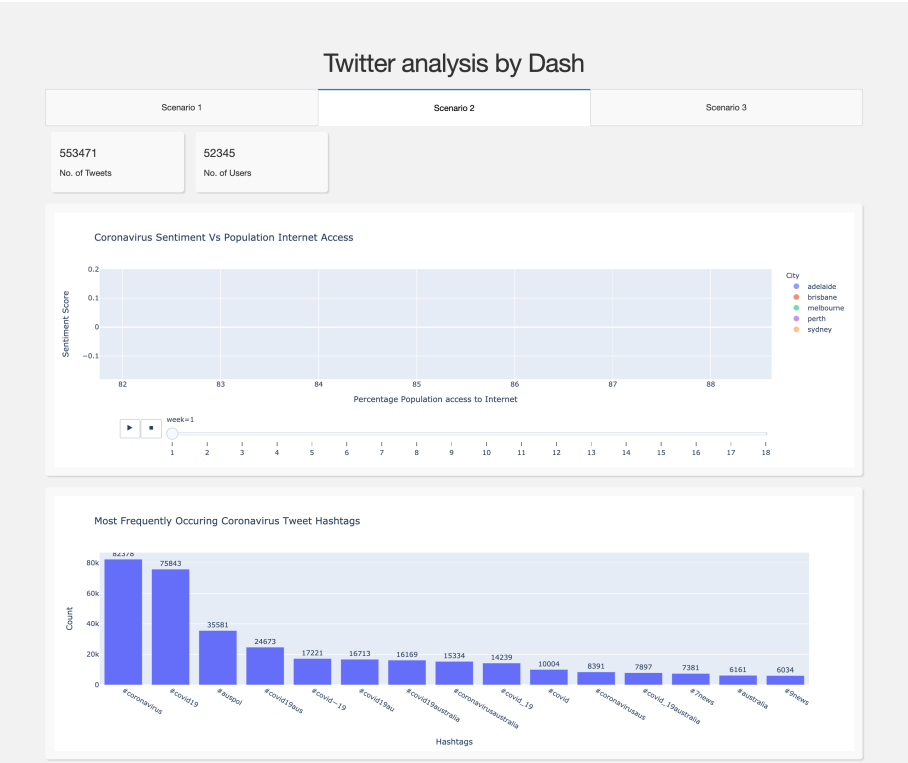
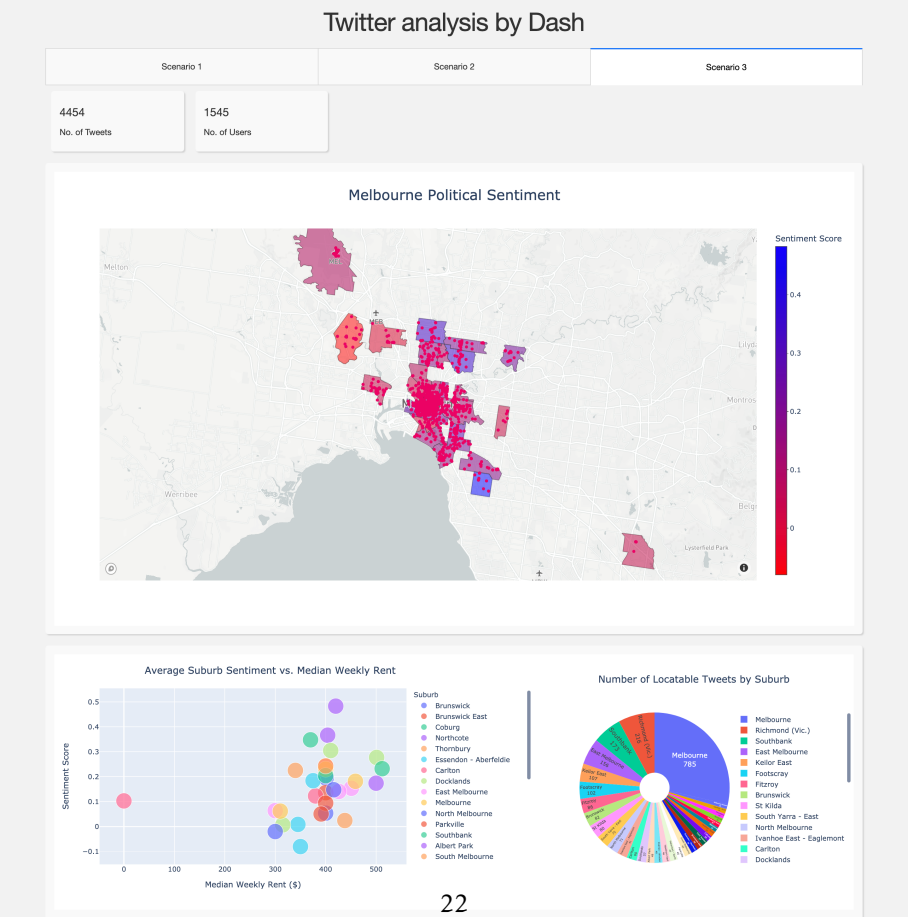


Figure 17: Scenario 2



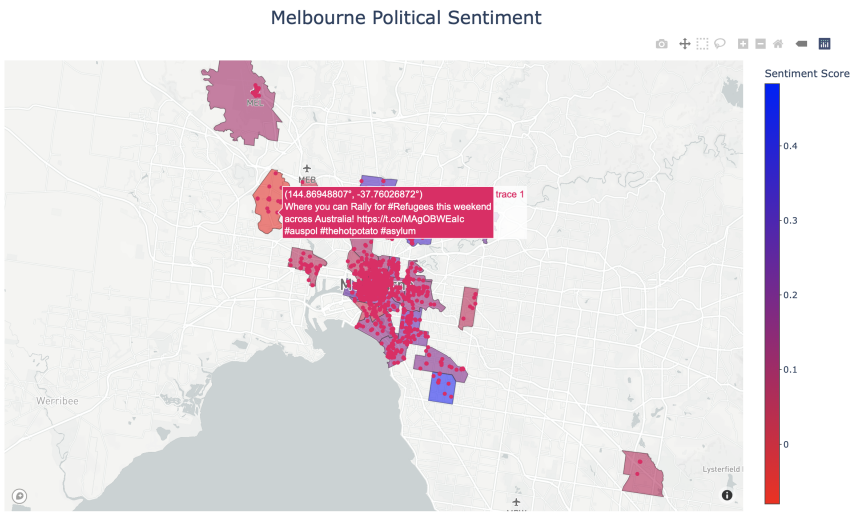


Figure 19: Mouse over action to see the tweet

8 User Guide

In this section we provide a guide on how a new user/developer can deploy our cloud architecture, and how a user can use/interact with the data analytic front end.

8.1 Architecture Deployment

The following section describes how to run the Ansible script to build and deploy the entire cloud infrastructure, dependencies, and harvester applications. We used Ansible as our deployment technology; we used Ansible version 2.9.7 and Python 3. Before following the below deployment steps, the user must have Ansible installed on their control/deployment system.

8.1.1 Standard Full Deployment

- Download the openrc.sh file from the Unimelb Research Cloud, and save it in the root of the Ansible project folder, replacing the existing ”openrc.sh” file.
- Make sure you have a key-pair configured on your Unimelb Research Cloud and local machine, and change the ”instance_key_name” variable to match your key-pair name in the ”nectar.yaml” and ”nectar-scale-up.yaml” files.
- Generate and save your Openstack API password, for the Research Cloud, somewhere on your local machine, and have it ready when running the script as you will be prompted for it.
- Make sure the desired resources are correctly set in the ”nectar.yaml” host variable file. The current default resources is 4 compute instances, 8 volumes, 4 volume snapshots with 240GB total.
- Run the ”run-jam.sh” shell script inside the root of the ”twitter_jam” folder.
- The user will be prompted for their OpenStack password, and then their local machine password. When the script first connects to the instances, the user will be prompted to confirm the host fingerprints, which is the last prompt. Once the script is done, the Twitter harvester will be automatically running and populating the database.

8.1.2 Scale Up Deployment

The scale up deployment script is written with the assumption that the standard full deployment script was previous run, and the user wishes to increase the number of deployed instances, Couch-DB nodes, and Twitter harvesters.

- Download the openrc.sh file from the Unimelb Research Cloud, and save it in the root of the Ansible project folder, replacing the existing openrc.sh file.
- Make sure you have a key-pair configured on your Unimelb Research Cloud and local machine, and change the "instance_key_name" variable to match your key-pair name in the "nectar.yaml" and "nectar-scale-up.yaml" files.
- The user will need to setup the IP for the master node in the Couch-DB cluster, by setting the "master_node_ip" to the IP of the jam-1 node in the "jam-scale-up.yaml" file.
- Generate and save your Openstack API password, for the Research Cloud, somewhere on your local machine, and have it ready when running the script as you will be prompted for it.
- Make sure the desired resources to be added are correctly set in the "nectar-scale-up.yaml" host variable file. The current default resources is 1 compute instances, 2 volumes, 1 volume snapshots with 35GB total.
- Run the "run-jam-scale-up.sh" shell script inside the root of the "twitter_jam" folder.
- The user will be prompted for their OpenStack password, and then their local machine password. When the script first connects to the instances, the user will be prompted to confirm the host fingerprints, which is the last prompt. Once the script is done, the Twitter harvester will be automatically running and populating the database.

8.1.3 User Interface guide

This is a guide for user of how to use our front-end. User can perform several actions to visualize our scenario.

- In Scenario Tab, user can switch the tab to view the scenario desired by clicking on the tab.
- In scenario 1, the user can select which AURIN data to correlate by using drop down option and select it. It will automatically load the plot.
- In each of the plot object, the user can select the specific region of the plot by draw a rectangle on the graph which it will automatically zoom. User can gloss over the point in the plot and user can observe the data information such as tweet, sentiment analytic and etc.
- In each of the plot object, the user can select the city not to show on the plot by clicking on the city name on the right hand of the plot. The plot will remove the selected city and show the remaining ones.
- The user can export the plot as an image by using the mouse the gloss over the plot and click on the image icon which it automatically generates an image and download for the user.
- The user can zoom in and out by using scrolling gesture and user can gloss over to the point to see the tweet.

8.1.4 Project links

- **User Interface:** Make sure you are connected with UniMelb VPN in order to open http://172.26.130.9:8050
- **GitHub Repository:** https://github.com/saket404/CloudTweetAnalyzer
- **Youtube Video:**
  Deployment video: https://www.youtube.com/watch?v=xNvjbCEzq2Q
  UI video: https://www.youtube.com/watch?v=IhH2FLoNKW0
- **Overleaf LaTex:** https://www.overleaf.com/6192118193ytfcrcrcdbyh

References

Twitter Developer. Twitter documentation, 2020. URL https://developer.twitter.com/en/docs/tweets/data-dictionary/overview/tweet-object.

Dmitry Mottl. Getoldtweets3 pyhton library to access old tweets, 2020. URL https://github.com/Mottl/GetOldTweets3?fbclid=IwAR0LTitci8ZwJSo8IX4pE0hoQ-X1XHtTYveJTtC8K-4LGVN5Elg-Vtd24GA.

Shammamah. Dash oil and gas demo, 2019. URL https://github.com/plotly/dash-oil-and-gas-demo?fbclid=IwAR0LuICiIwKd6q2WV9W8TftJQdziMfoeiSVopE9eF6R5pszFZDC82XuY_DU.