Livability of Melbourne: An Analysis of Crimes and Mental Health in Local Government Authorities Based on Tweet Data

Renwei Hu[1], Siwat Chairattanamanokorn[2], Chi Yin Wong[3], Renkai Liao[4], and Kaiquan Lin[5]

The University of Melbourne
[1]1067974, renweih@student.unimelb.edu.au
[2]1338152, siwat.chairattanamanokorn@student.unimelb.edu.au
[3]836872, chiw2@student.unimelb.edu.au
[4]1141584, renkai@student.unimelb.edu.au
[5]1147233, kaiquanl@student.unimelb.edu.au

17th May 2022

Abstract

This is the final report for COMP90024 Assignment 2 Team 53. The purpose of this assignment is to develop a cloud based solution to explore aspects of the livability of Melbourne through a set of livability indicators by developing scenarios for these indicators and analysing the scenarios based on real time tweet data. In this project, we focused on the neighbourhood and health indicators, specifically focusing on the crime and mental health aspect in Greater Melbourne.

***Keywords*—** AURIN, Twitter, CouchDB, Docker, Tweets, TwitterAPI, Ansible, NeCTAR, Melbourne Research Cloud

Contents

1 User Manual

In this section, we show the user manual of each component in this project, including Twitter Harvest, Ansible Automation and Front-end.

1.1 Twitter Harvester

Twitter harvester is used to get specific tweets using Twitter API 2.0 from Twitter server. This function can only run with an available CouchDB server. Multiple harvester instances can be deployed in the cloud or run locally. To deploy a tweet harvester/several harvesters in the cloud, please refer to section 1.2.

In order to run one tweet harvester, we need to use files in the crawler folder. First set the basic information in the `raw_tweet.py` file. In `BEAR_TOKENS`, write down all the tokens that will be used for this harvester, and provide a search query to define types of tweets that will be searched. Second, CouchDB server information must be provided, including IP address, username, password and database name. If no such database exists, the crawler will automatically create one.

There are two versions of crawlers. The first one will keep tweets with or without geo information and store them in a separate database. The other one will discard tweets without geo information. The number of tweets for each request can be specified, as well as how many requests the crawler should do before tweets are stored to the database. Moreover, the speed of the harvester can be enhanced by adding more Twitter tokens. In order to run multiple harvesters at the same time on the could, first create a duplicate of `raw_tweet.py` file, then change the configuration for the new harvester. A more detail guide on Twitter Harvester can be found in Tweet Harvester Demo Video.

1.2 Ansible Automation

Ansible is an open-source automation tool that provides the ability to configuration management, application deployment and repetitive maintenance tasks in a large scale server cluster. Ansible is considered more efficient and less time-consuming in terms of configuring systems. More importantly, the support for automation avoids any human errors which are common in operations in large scale systems. Therefore Ansible is used in this project to automate nearly all tasks.

1.2.1 Overall Workflow

All automation tasks are defined in a single Ansible playbook which is generally divided into two plays. The `mrc.yaml` file under the ansible folder is the Ansible playbook. As shown in Figure 1, there are two parts and each part is conceptualised as a single play in the playbook. The first play consists of tasks related to instance configuration on MRC. Since Ansible's agent less mechanism, it is possible to initialise remote instances with given APIs. The second play performs on the instances just created to set up the environment and deploy services like CouchDB and Twitter harvester.

```
22  - hosts: localhost
23    vars_files:
24      - host_vars/vars.yaml
25    gather_facts: true
26
27    roles:
28      - role: create-volume
29      - role: create-security-group
30      - role: create-instance
31
32  - hosts: MRC_NODES
33    vars_files:
34      - host_vars/vars.yaml
35    gather_facts: true
36    become: yes
37
38    roles:
39    - role: common
40    - role: mount-volumes
41    - role: install-docker
42    - role: init-couchdb
43    - role: set-couchdb-cluster
44    - role: deploy-harvester
45    - role: deploy-frontend
```

Figure 1: Ansible Playbook

Global variables are saved in `host_vars/vars.yaml` as configs to create volumes, security rules and instances. The role is the execution block to perform operations. Roles for each play are listed under `roles` keyword and all roles are stored as YAML files too under `/ansible/roles`. Each role represents a general task and the steps required to complete the task are defined in role files. Those steps are also called modules in Ansible and modules are granular and very specific such as copy files, installing packages or building a Docker image.

1.2.2 Instance Configuration

The first play will create four volumes with 100GB each which will be attached to instances during initialisation. A list of security rules is created to enable incoming traffic for remote instances. The incoming traffic is disabled by default and services running on instances can not be accessed externally without these custom security rules. Four instances on MRC will also be created based on *Ubuntu 20.04 LTS (Focal)* image. Each instance has 2 cores and 9GB of memory. The volumes and security rules will be attached to instances in the meantime.

Since it is expected to use a single playbook to complete the whole configuration, in-memory inventory is used instead of an inventory file. As shown in Figure 1, the remote instances are added to a host group `MRC_NODES` and are able to be referenced in later tasks or other plays in this playbook.

1.2.3 Service Deployment

The second play will install packages and required dependencies to set up the environment on newly created servers. The volumes will be formatted and mounted to the specified mount point. After that, Docker will be installed on all instances for the later deployment of services. CouchDB will be initialised on 3 instances in Docker containers and joined together as a cluster via HTTP calls. Volumes will be bind-mounted to each container to allow permanent storage of data.

Similarly, 3 Twitter harvesters are deployed on 3 instances via Docker container as well. They will use the same Docker image but with different environment variables to determine its role. The Twitter harvester on 3 instances will focus on collecting raw data, crime data and mental health data respectively. Our front-end is a React service deployed on the last instance which is able to access the CouchDB to retrieve results already processed.

1.2.4 Execute Ansible Playbook

The shell command to run the Ansible playbook is saved in `run-mrc.sh`. The `config.sh` file is the OpenStack RC File downloaded from MRC for remote connection. Your private key is also required to be specified for MRC authentication. The `mrc.yaml` at the end is the playbook to execute. Since the command has already been saved, it is easy to execute by just typing `"sh run-mrc.sh"` in a terminal. Your OpenStack API password will be asked in order to start processing.

1.3 Front-end

Running React web app locally with docker:

1. Build docker image `"docker build -t webapp:latest ."`
2. Run docker container from the created image `"docker run -p 3000:3000 -it webapp:latest"`

After running both of these commands, a docker container will be created and the webapp is hosted at `http://localhost:3000`, which can be accessed from any browser on the local machine.

2 System Architecture and Design

In this section we described the system architecture and design with reasons why each method was selected.

2.1 System Architecture

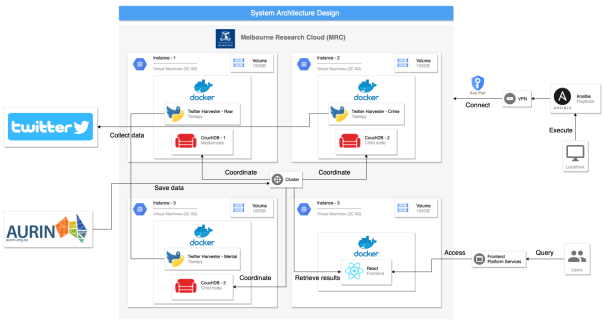

Figure 2: System Architecture Design

2.2 Melbourne Research Cloud (MRC)

The Melbourne Research Cloud (MRC) [1] is a cloud computing platform provided for students and researchers at the University of Melbourne. Unlike cloud providers like Amazon Web Services (AWS) and Microsoft Azure, MRC is a private cloud infrastructure hosted on the servers at the university. But similarly, they are all Infrastructure as a Service (IaaS) cloud computing which offers virtualised computing resources for their target users.

2.2.1 Instances

Four instances are allocated for our project on MRC with each having 2 virtual processor cores and 9GB of RAM. Figure 3 is a summary of the resources used for our project. The first three out of four instances are used to deploy CouchDB and Twitter harvester. Each instance will have 2 Docker containers running for CouchDB and harvester as illustrated in Figure 3. The last instance is designated to host our front-end service and allow users to access via port 3000 to review our collected results. *Ubuntu 20.04 LTS (Focal)* is selected as the system environment for stability. All instances are initialised and configured by the Ansible playbook to ensure consistency and free from human errors.

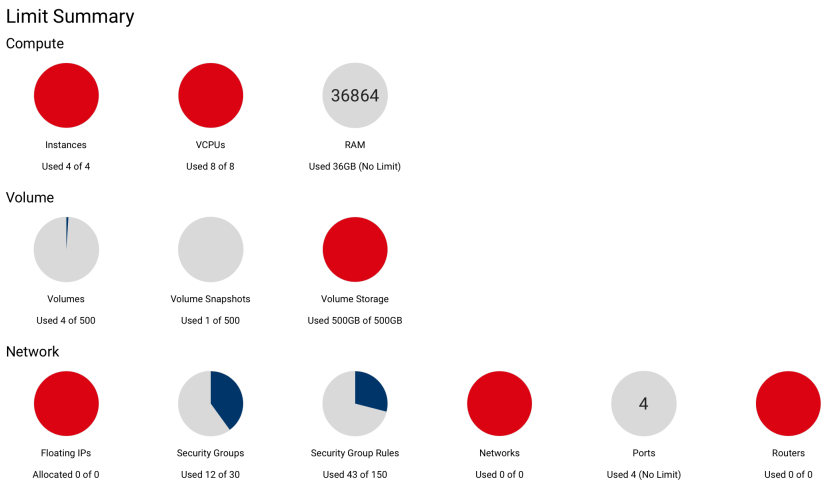

Figure 3: Summary of MRC

2.2.2 Volumes

We are allocated 500GB of volume to store data permanently. Therefore, 4 volumes of 100GB are created to be attached to each instance. The volumes are formatted and mounted to the mount point `/data` and link to the Docker containers via bind mount. Compared to the default volume mount managed by Docker itself, the bind mount is more flexible and easy to access the data from host machines.

Displaying 4 items

Name	Description	Size	Status	Group	Type	Attached To	Availability Zone	Bootable	Encrypted	Actions
data-vol-4	-	100GiB	In-use	-	standard	/dev/vdb on instance-4	melbourne-qh2-uom	No	No	Edit Volume
data-vol-3	-	100GiB	In-use	-	standard	/dev/vdb on instance-3	melbourne-qh2-uom	No	No	Edit Volume
data-vol-2	-	100GiB	In-use	-	standard	/dev/vdb on instance-2	melbourne-qh2-uom	No	No	Edit Volume
data-vol-1	-	100GiB	In-use	-	standard	/dev/vdb on instance-1	melbourne-qh2-uom	No	No	Edit Volume

Figure 4: Summary of MRC

2.2.3 Network and Security Rules

Since Melbourne Research Cloud is a private cloud platform, a VPN is required to establish secure connections to the servers due to security concerns. The MRC only opens to internal access from the university network and external queries will be blocked to prevent potential attacks from outside.

By default, every instance only permits outgoing network traffic and blocks incoming traffic. Therefore security rules have to be configured while initialising instances to allow incoming connections. Otherwise even the basic SSH connection will be blocked and it is impossible to access the servers from outside. A list of security rules is set for instances used in our project with several ports opened for external access. Those ports are used by services deployed on our instances and are required for their proper functioning.

- Port `22` - SSH connection
- Port `80, 443` - HTTP/HTTPS connection
- Port `4369, 5984, 9100` - CouchDB service
- Port `3000` - React service

2.2.4 Flexibility

Flexibility is another major feature of IaaS cloud computing. Unlike physical machines, virtual machines hosted on the cloud are easy to be configured with desired hardware combinations. It is easy to choose how many resources are allocated for each instance including processors and memory space. In our case, the number of instances to be launched and the size of volumes to be attached for each instance are decided by us with high flexibility. MRC makes it possible to create the arbitrary size of volumes to be used by each server while there is no way to split a physical hard drive and attach it to different machines. This level of flexibility also provides the ability to easily scale up or down for services deployed on MRC. We can easily create a new instance and deploy our services by just adding a few lines of code to Ansible scripts.

2.3 Front-end Component

2.3.1 React

Due to the limited timeframe of this project, we decided to use React to develop the front-end because it is relatively simple and fast to learn React from online sources such as online tutorials and youtube videos. React

is an open-source front-end JavaScript library for building user interfaces (UI) based on components. As React is widely used by many developers around the world, there are many open-source React libraries available for us to select from. We chose KeplerGL to represent AURIN data in the form of a heatmap and plot Twitter data on top of it with cluster representation. We also used CanvasJSReact to visualise Twitter data fetch from the back-end CouchDB server in a form of a bar chart. Additionally, there are many free templates available to download in React.

2.3.2 Kepler.gl

Kepler.gl is an open-source high-performance web-based tool built on top of deck.gl used to visualise geospatial data created by Uber. We chose Kepler.gl because it has an API for us to visualise both geojson data and geo-coordinates (latitude and longitude) on the same map representation. Kepler.gl categorised input data into layers, for each layer we can analyse and select different types of representations such as points, clusters and hexagons. This allows us to plot both AURIN data and Twitter data on the same map representation with different data presentation formats.

2.3.3 CanvasJS React Component

CanvasJS React Component is an open-source JavaScript library allowing users to create diagrams, such as bar chart, pie chart and scatter chart, to display data according to their desires. We picked CanvasJS to visualise our Twitter data in the form of bar chart as this library is lightweight and simple to use and incorporate into the project.

2.4 Back-end Component

2.4.1 ReSTful-based Web Service Oriented Architecture

To manage a user's access to remote resources, we decided to adopt a ReSTful-based framework. An alternative option would have been to use a SOAP-based architecture. However, ReSTful has several advantages over SOAP. ReST allows us to use a wider variety of data formats such as JSON, whereas in a SOAP architecture, we would only be able to use XML. The JSON format has faster parsing and generally works better with data, and the JSON format offers superior support for browser clients compared to XML. Another advantage is that ReST requires significantly less bandwidth when making requests to the server through HTML calls because it mostly contain small JSON messages, whereas, a SOAP message contains much more information. Overall, accessing data from CouchDB using ReST is faster. Thirdly, any requests made by the client are considered "safe" since the client would only require the HTTP method GET. Safe methods do not change, so repeating a call is equivalent to not making a call at all, preventing any duplicate requests. Lastly, our project does not need to maintain a state of information from one request to another since each request will be separate without the need of flowing some information from one request to another. The ReST API allows us to achieve this statelessness. In summary, the ReST framework is much more suitable for our project than SOAP.

2.4.2 HTTP Methods

Using the ReSTful system frameowrk, we were able to use simple HTTP methods with JSON messages to manipulate resources. As previously mentioned, ReST has superior performance to SOAP when making server requests, requiring less bandwidth, and achieving quicker speeds. The main HTTP methods we used when building our backend application were GET calls to retrieve queried data from CouchDB and POST calls to upload historical tweet data onto CouchDB. The bulk docs API was used to upload the large historical twitter file onto the database, ensuring that each individual tweet was stored as a separate document.

2.5 Tweet Harvester

The Twitter harvester is designed to run in the cloud or alone in a local machine(with an available CouchDB server) with good scalability. It is implemented by Python and the structure is shown below. To maximize the scalability in the cloud, all functions needed for the harvester are wrapped in a single file `get_tweet.py`. In a tweet harvester instance, we only need to follow the template file `raw_tweet.py` and simply change the basic settings. Thus it's easy to expand the harvester instances. Users can easily create multiple instance files to deploy multiple instances for different purposes, or even use one file to deploy multiple instances(simply wrap every instance as a function). It's more like to use the tweet harvester like a 3rd party Python library.

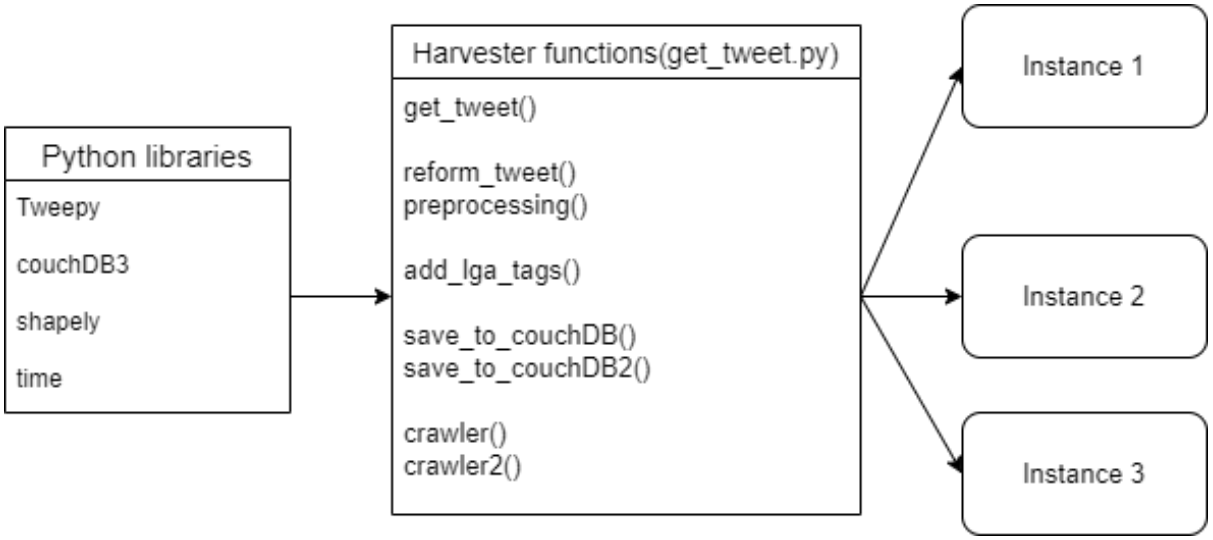

Figure 5: Twitter Harvester Structure

The twitter harvester contains four parts: data fetching, data pre-processing, data filtering and saving data to CouchDB. In the data fetching part, the main Twitter API used for harvesting is keyword search in Twitter API v2. Function `get_tweet()` is designed for this part. A search query is used to define what kind of tweets we want to get from Twitter. A search query can contain keywords, hashtags, languages and more. Users can define different queries for different kinds of tweets. The number of tweets in each request can also be changed from 10 to 100. This function will get tweet id, author id, tweet text, created time and geo information(if exist). This part cannot be changed. Due to the rate limit of the basic token, the function will sleep after every request. The sleep time depends on the number of tokens and the number of tweets each request gets.

Data pre-processing is used to reform the data type for later analysis. Function `reform_tweet()`, `preprocessing()` and `add_lga_tags()` are used for this part. The first two functions are used to transform a tweet object into a dictionary type. This is mainly used to fit the CouchDB request. Function `add_lga_tags()` is used to attach LGA tags to tweets with coordinates. The LGA tags will be used in later data analysis.

Twitter harvester uses a two-step filter system. The first filter is the search query, which defines the kind of tweets we want. But due to the limitation of our tokens(details are in section 5.1), a second filter is defined to get more accurate results. The second filter is integrated with saving data to CouchDB. Two versions of the second filter are defined, users can choose any of them by using a different crawler function. The first version will keep all tweets with or without geo information, and they will store them into separate databases. The second filter will not only discard tweets without geo information, but also discard tweets if the coordinates are not in the great Melbourne. All tweets will be stored in one database in this version.

Besides scalability, two other features we care about are Internet loading and error handling. Fetching tweets from Twitter and saving tweets to CouchDB server can both increase Internet traffic. To reduce traffic from Twitter, each request we get as many tweets as possible(100 tweets in this case). For the CouchDB side, it's more complicated. If saving tweets after a lot of requests, and then the CouchDB server is unavailable, these tweets will be discarded; if saving tweets after every single request, it will add heavy traffic to the CouchDB server and decrease the total performance. After some experiments, we conclude that saving 200-1000 tweets one time is the best range. This depends on the specific scenario, and users can adjust this number.

Tweet harvester highly relies on the Internet, which is a very unstable environment. Therefore error handling is very important, otherwise we have to deploy the instances again every time an exception occurs. The exceptions the tweet harvester may encounter are plenty. Table 1 shows some of the possible errors it might have when fetching tweets. Also the CouchDB server can also be unavailable. But all these problems have a similar feature: they cannot be controlled or anticipated by the users. In other words, the problems are on the Internet or Twitter server or CouchDB server, not the harvester. Therefore the best solution is to wait some time and try again. In tweet harvester, once it receives any kind of exception, it will sleep for 15 minutes and then try again. It will also print the exception message for users to check. Thus we don't have to deploy the harvester instances again for some exception, and we can also know the exception type.

Error Code	Error Message
400	Bad request
403	Forbidden
429	Too many request
500	Internal server error
502	Bad gateway
503	Service unavailable

Table 1: Error code and message

2.6 CouchDB

The CouchDB is chosen as the database and the CouchDB cluster running on the three nodes is set up. The CouchDB will store the tweets harvested from the crawler and provide the MapReduce functionality for further analysis. The access way of CouchDB is based on the HTTP. Because of the popular use of HTTP protocol, the learning curve of using the CouchDB will be smooth, and it makes manipulating the database simple and convenient by calling HTTP query. CouchDB also provides a graphical user interface called Fauxton UI. By entering the address of the CouchDB in the browser, accessing the state and functionalities of the database is intuitive and easy, which helps to manage data and design the MapReduce views. The appropriate data store format is an advantage of CouchDB. As the format of the fetched tweets and the format of data stored in CouchDB are JSON, the CouchDB is suitable for this project to store the data simply without the extra transformation of data. The data in JSON also helps to understand the structure of the tweets easily and give the feedback of the work of the crawler. In addition, the robustness of the database is also a consideration. The crawler running 24 hours per day required the stability and safety of the database. CouchDB is the cluster database which can be deployed and run on different nodes. The consistency and validity of data can be promised under the use of the Couch Replication Protocol. Thus, a failure in a node will not affect the availability of the database using CouchDB and ensure the crawler can store the tweets successfully. The accessing database via HTTP APIs, use of JSON and the property of cluster database make the use of the CouchDB reasonable and appropriate in this project. Therefore, CouchDB is selected as the database.

2.7 Containerisation

2.7.1 Benefits

One of the key challenges in software development is the inconsistency between the development and production environment. Due to the variety of operating systems and environment configurations, programs that can run in development settings may not properly function in the production environment. For example, the huge difference between Windows and other Linux based systems in terms of their architecture determines the program running on Windows needs to be re-configured so that it can run smoothly on Linux. Different servers running on Linux may still have issues related to solving dependencies and configuring the development environment.

Containerisation turns out to be the perfect solution for that by encapsulating or packaging up software code and all its dependencies so that it can run uniformly and consistently on any infrastructure. Containerisation bundles the application code, configuration files and libraries required together into a single image which can be easily deployed on any machine regardless of operating systems or hardware. Unlike virtualisation which simulates the whole computer, services running in containers share the same host OS to reduce wasted resources.

2.7.2 Docker

Docker is currently the leading software container tool and is used in this project for service containerisation. Docker can be easily installed on Windows, macOS and Linux so we can test our programs as containers on local machines and then transfer them to run on MRC. Docker has rich documentation and an active community to provide solutions for any issues we encounter. Some base images are uploaded to Docker Hub which can be easily pulled and run directly.

We use Docker to host the CouchDB cluster based on the image developed by IBM. It is attempted to use Docker Swarm for hosting the CouchDB service as a distributed system while achieving high availability. However, it's much more complicated than what we expected so we eventually have 3 CouchDB containers running on

3 instances. Configurations such as database username and password are passed to build the Docker image via Ansible scripts. The ports used by CouchDB are properly mapped to the container which allows to set up of a cluster of databases.

To preserve the data stored in CouchDB, a volume attached to the host machine is bind mounted to the Docker container. Generally, containers can save data with Docker volumes which are created and managed by the Docker engine by default. However, when external volumes are specified to be bind mounted to containers, it's more flexible to access and manage the data we generated inside containers. In our case, the data stored in CouchDB can be directly accessed via the host machine while this is not allowed if using Docker volumes.

Apart from the CouchDB, both the harvester and the frontend services are running inside Docker containers as well. The instructions for building the image are defined via a Dockerfile with given arguments, environment variables and dependencies required. To deploy a service, we can use Ansible scripts to copy the source files to MRC instances and use Docker commands to build and run the service as a container. Docker will install all dependencies specified in Dockerfile and execute commands to launch the service. With proper port-mapping, the services running inside containers can be accessed and interacted with just like running on usual machines.

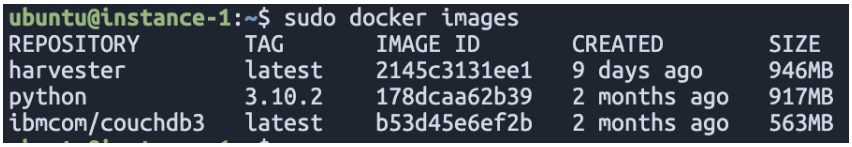

Figure 6: Docker images

Docker provides an easy and more efficient way to bundle programs without worrying about solving dependencies and compatibility of operating systems. The isolation provided by containers improves the security and inconsistency of environments while services can be properly accessed with correct port mapping.

3 Data Delivery

In this section, we demonstrate the how we collected, processed and visualise data of this project.

3.1 AURIN Data Collection

Initially, our team discussed the possible scenarios for this project and came up with 2 main scenarios related to crime and mental health. For crime, we decided to collect data related to drug use and family violence as these incidents usually occur in the area where the villain and victim live. For mental health, our scenario is whether poor or rich people are more prone to mental illness, thus we collected income and distress rates from the AURIN portal. The desired dataset was then searched in Victoria, Australia. The output data returned all the information according to our search topic categorised by LGA. Then, data from the AURIN portal is downloaded to the local machine in JSON format, as both the front-end and back-end required data to be in JSON format. Furthermore, the downloaded dataset from AURIN was preprocessed to remove any unnecessary data from other LGAs in Victoria apart from Greater Melbourne.

3.2 Tweet Data Collection

We deployed 3 tweet harvester instances running in parallel in MRC. `tweet_topicA.py` is used for scenario A, `tweet_topicB.py` is used for scenario B. In each instance we used keywords and coordinates as filters. Keywords used for scenario A and scenario B are shown in Table 2. These two instances only keep tweets which check all our requirements, including keywords and coordinates. Tweets for scenario A was stored in `crime_tweets`, and tweets for scenario B was stored in `mental_tweets`.

Scenarios	Keywords
Scenario A: Crime	abduction, abuse, accident, arson, assassination, assault, bigamy, blackmail, bombing, bribery, burglary, corruption, crime, cybercrime, domestic violence, drug, embezzlement, espionage, family violence, felony, forgery, fraud, gang, genocide, hijacking, hit and run, homicide, hooliganism, identity theft, incident, infraction, kidnapping, looting, lynching, manslaughter, mugging, murder, pickpocketing, pilfering, poaching, rape, riot, robbery, shoplifting, slander, smuggling, terrorism, theft, trafficking, transgression, trespassing, vandalism, voyeurism, violation
Scenario B: Mental Health	anxiety, craziness, delusion, depression, disturbed mind, dying, emotional disorder, emotional instability, exhausted, hallucination, insanity, mental disorder, mental disease, mental sickness, mental health, nervous disorder, neurosis, neurotic disorder, personality disorder, schizophrenia, self-harm, self harm, suicide

Table 2: Scenarios and related keywords

Due to the limitation of the tweet harvester(details are in section 5.1), we anticipated that these two instances are very difficult to get any tweet. Therefore we deployed the 3rd instance to fetch general tweets about Melbourne. We used keyword "Melbourne" and hashtag "#Melbourne" to capture all tweets related to Melbourne in case the other two instances can't get any tweet. These tweets will also keep tweets without geo information to database "raw_tweets", and tweets with geo information to database "geo_tweets". Figure 7 shows the CouchDB server info about the number of tweets we have already fetched(the screenshot was taken on 14th May). From Figure 7 we can see that no tweet has been found for scenario A or B due to the tweet harvester limitation. But over 100,000 tweets about Melbourne were successfully found and saved to the database. Among them there were over 1000 tweets with geo information were found(about 500 tweets were lost due to database issues).

3.3 MapReduce Views in CouchDB

The built-in MapReduce functions were used to query specific tweets from our CouchDB database. These were done through the Fauxton graphical user interface.

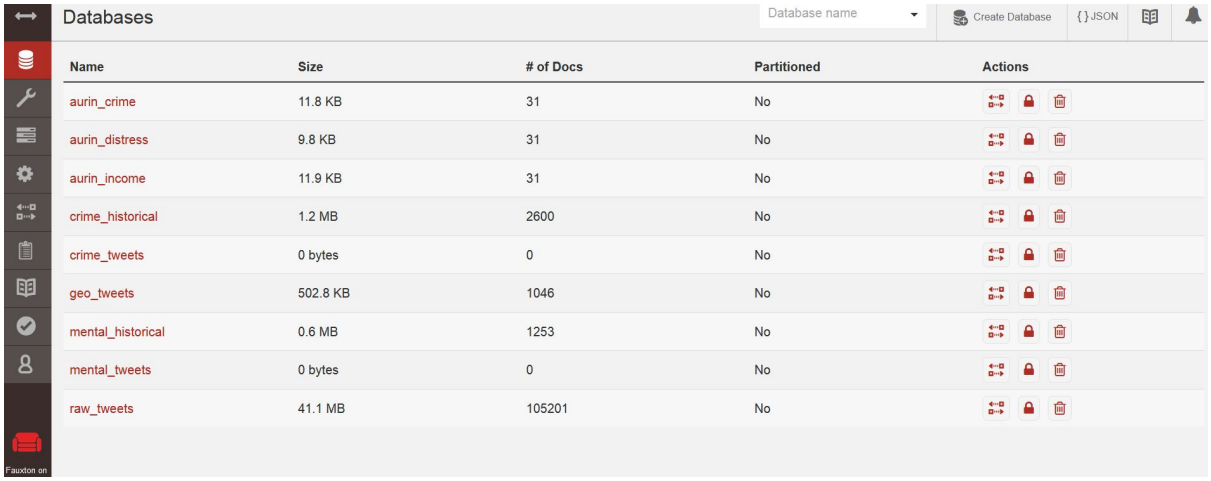

Figure 7: CouchDB info

3.3.1 Query LGA and the number of tweets in that LGA

For this query, the _count reduce function is used. The tweets are first grouped by the keys LGA_NAME and LGA_CODE. The map function is

```
function (doc) {
  emit({LGA_NAME: doc.LGA_NAME, LGA_CODE: doc.LGA_CODE}, 1);
}
```

Figure 8: CouchDB Views LGA

The built-in _count reduce functionality was then used to count how many tweets were in each LGA for both the crimes and health data.

3.3.2 Query the exact coordinates and the LGA of each tweet

For this query, no reduce function was required. The tweets were first grouped by the keys LGA_NAME and LGA_CODE. The map function is

```
function (doc) {
  emit({LGA_NAME: doc.LGA_NAME, LGA_CODE: doc.LGA_CODE}, doc.coordinates.coordinates);
}
```

Figure 9: CouchDB Views LGA Coordinates

3.4 Melbourne Historical Tweet Data LGA

3.4.1 Data Filtering

The historical tweet data file is a large (10 GB+) file containing tweets that were made in Melbourne in the past. This data contained many tweets that did not have location information (tweet coordinates). To deal with this issue, we removed all these tweets since we wouldn't be able to extract the local government authority (LGA) that the tweet belonged to. Based on a list of predetermined keywords for both crime and mental health, only tweets that contained these keywords (under the text field) were kept. For simplicity and ease of analysis, many irrelevant fields (such as a user's screen name) were also removed. The resulting files were named crimes.json and health.json. These files would be used later on to determine which LGA a tweet belonged to.

3.4.2 Extracting LGAs from Geojson

The geojson file contains the coordinates (as a Polygon object) and name of each of the LGAs in Victoria. A total of 91 LGAs were identified. For this task, we only considered the 31 LGA's in greater Melbourne, so all the other 60 LGA's were removed. The LGA codes were also appended to each LGA. This information was taken from Wikipedia and the official travel Victoria website.

3.4.3 Attaching LGA Information

The shapely.geometry library was used to find out whether or not a tweet belonged in a specific LGA. since the tweet coordinate was given as a point object and the LGA coordinates as a polygon object, the contains method was used. The crimes.json and health.json file containing the extracted data was used. For each tweet, if the tweet was inside any of the 31 LGA's, the LGA name and code were appended to the end of the tweet as a new key. For tweets that had coordinates outside of these LGAs, they were given a default value of NO_LGA. The tweets with LGA's attached were then written into a two new JSON files, separated by whether they had an LGA_NAME that wasn't the default value. The files were named crimes-updated-lga.json and crimes-updated-without-lga.json.

3.5 Front-end Data Fetching

The data fetching process in the front-end is done by using REST API with fetch function from "node-fetch" command. Fetch creates an HTTP GET request to the CouchDB server hosted on the MRC at http://172.26.134.126 . The HTTP link for GET request is generated from CouchDB with Reduce function on Fauxton interface to only show desired data within the database. AURIN data is also stored in the database and can be fetched in the same manner.

3.6 Back-end Data Visualisation

After data is fetched from the back-end server, the data can be visualised in the form of a bar chart and map representation as shown in Figure 10.

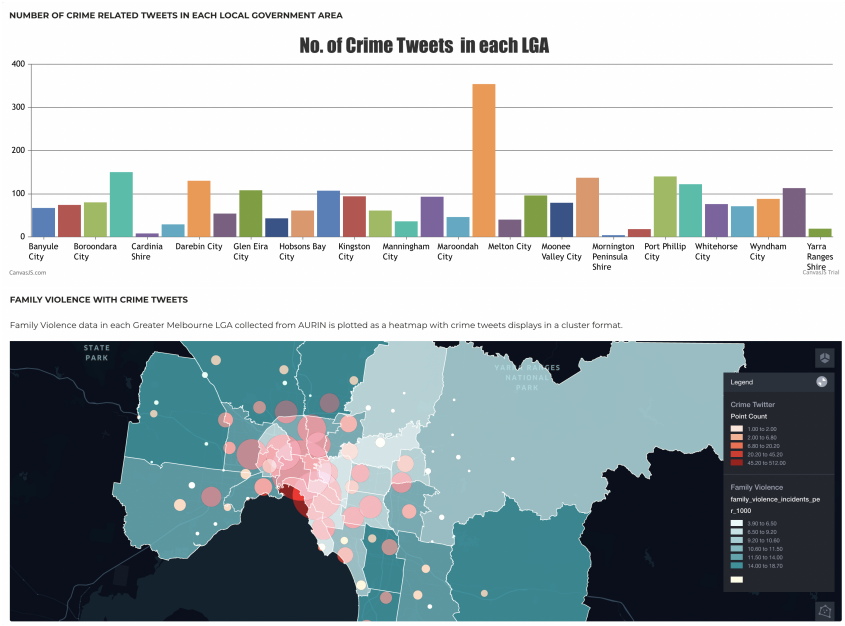

Figure 10: Back-end Data Visualisation

4 System functionality and Analysis

4.1 System Functions

1. Develop and deploy cloud infrastructure to collect data from Twitter
2. Process and analyse collected data
3. Use MapReduce from CouchDB for twitter analytic in order to compare with AURIN data
4. Create visualisation of data from back-end server and display it on a webapp

4.2 Scenario

We decided to investigate into 2 scenarios as follows:

1. Overall statistics of incidents mentioned on Twitter in the suburbs of Melbourne. Are there likely to be more crime in the area where the rate of drug use and family violence is high?
2. Overall statistics of mental health related tweets on Twitter in the suburbs of Melbourne. Does the number of tweets relating to mental health have correlation with the type of people living there. Are rich or pool people more prone to mental illness?

4.2.1 Overall statistics of incidents mentioned on Twitter

4.2.1.1 Description

In this scenario, we explored the data in AURIN related to drug use and family violence and compared the data collected with Twitter API. Gathered data from twitter is processed and filtered out unwanted tweets without crime keywords.

We decided to explore into the crime category of the livability, since it is one of the most important factor that makes a livable city . We believe that a city with low crime rate can make people feel safer and more secure while doing their daily routines. Moreover, there are many students residing in Greater Melbourne because there are many universities within the area, therefore we would like to dive deeper into the this aspect of the livability of a city.

We filtered tweets according to the keywords in Table 2. A total of 2600 tweets related to crime was collected with the relevant keywords and tweet based in Greater Melbourne. The data is then cleaned and process and can be seen in Figure 11 in a form of bar chart.

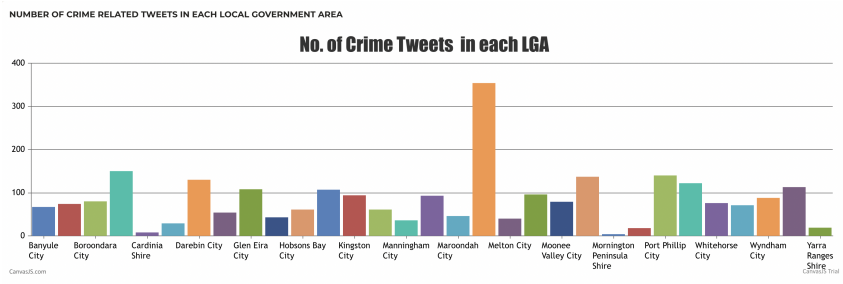

Figure 11: Bar chart of No.Crime Tweets in each LGA

4.2.1.2 Family Violence

Melbourne city has the highest concentration of tweets relating to crime, as well as one of the highest rate of family incidences. This result was expected since Melbourne city is by far the area in Greater Melbourne with the highest population, so we can assume that it is more likely for people to be constantly updating social media. There is a high positive correlation between the number of tweets and the rate of family incidences. Some other LGAs that exhibit similar patterns are Nillumbik Shire and Manningham City, with a low number of tweets and family incidences.

However, there are also many more areas that do not have this trend. The Surrounding areas around Melbourne city such as Stonnington, Boroondara and Glen Eira have a moderate amount of tweets about crimes, but a very low rate of family incidences. One explanation could be because these areas are close to Melbourne city, so people entering from Melbourne city, perhaps at the end of a work day, are tweeting about crime.

Outer LGA's like Hume City, Milton City, Cardinia Shire and Whittlesea City all have high family violence incidents, but very little tweets that are crime related. This is likely due to lower population in these areas, resulting in less of a tendency to use social media.

Overall, there doesn't seem to be a correlation between the number of tweets mentioning crime and the number of family incidents. There is no clear pattern in the LGA's in Greater Melbourne, and it seems that a deeper look into the population in each LGA and its correlation with the number of tweets relating to crimes might be more interesting. Thus, we conclude that there is no correlation between crime tweets and the number of family incidences.

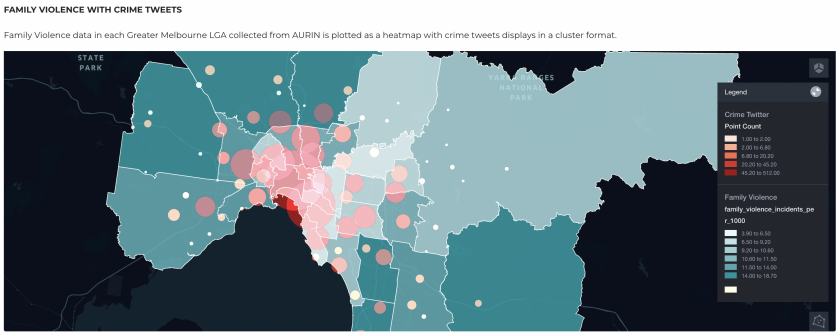

Figure 12: Heatmap Representation of Family Violence

4.2.1.3 Drug Use

The the data collected from AURIN, we can see that there is a high drug use present in Frankston, Greater Dandenong and Brimbank compared to other LGA in Greater Melbourne. Tweets related to crime are heavily concentrated near Melbourne.

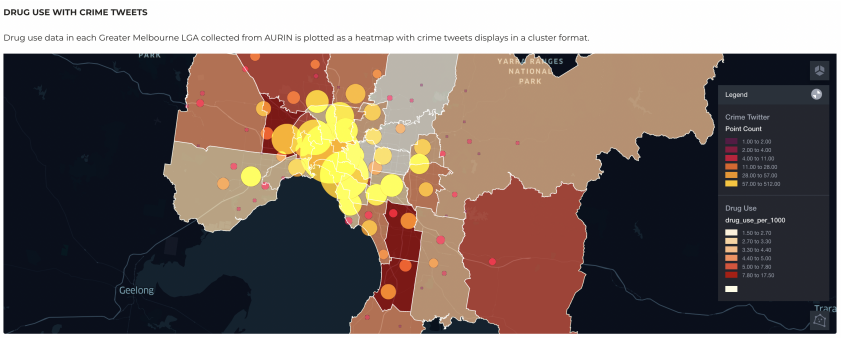

Figure 13: Heatmap Representation of Drug Use

According to Figure 13 AURIN and Twitter data are plotted in the form of heatmap and cluster format respectively. The darker the shade of red of heatmap the higher the drug use in that particular LGA grid. Similarly with the number of tweets, the larger the radius the higher the number of tweets in that area.

We cannot see a strong correlation between the actual drug use compared to crime tweet in all areas of Greater Melbourne. This is because for some area, such as Yarra Ranges Shire and Wyndham, AURIN shows a medium drug use, but there are insufficient data from Twitter to support that there are medium level of drug use in that particular area. Nevertheless, there is a postive correlation of crime tweets and drug use in Melbourne, Port Philip and Yarra.

From this we concluded that, there are some correlation between drug use and number of crime tweets present in certain area of Greater Melbourne mainly near Melbourne. We assumed that people living near the Melbourne

are more likely to post their activities on social media. Moreover, Melbourne is more populated compare to other areas of Greater Melbourne, hence Twitter data are more present in Melbourne and nearby LGA.

4.2.2 Overall statistics of mental health related tweets on Twitter

4.2.2.1 Description

This scenario investigates the underlying association between tweets that contain mental health related content and some potential factors of people living in regions of Greater Melbourne. It is believed that various elements of an individual are associated with the level of mental health in a certain way and may even form a causal relationship. Specifically, we looked into factors including distress rate and individual income where the data is provided from AURIN to support our hypothesis.

We classified a tweet as interest when it contains one of the keywords that are related to mental health or psychological well-being. A total of 1253 tweets are collected with such keywords and also located inside Greater Melbourne based on their coordinates when posting the tweets. The collection of tweets is cleaned and processed to attach standard LGA names. Therefore, a summary of tweets related to mental health in each LGA is illustrated in Figure 14. The bar chart shows the number of related tweets posted in each LGA and can be analysed later with data from AURIN.

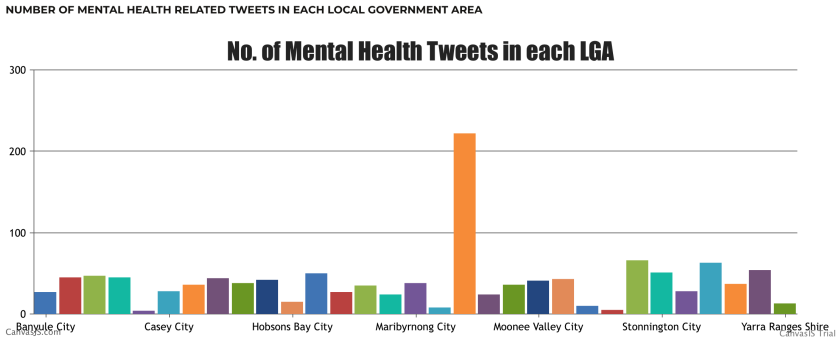

Figure 14: Bar chart of No.Mental Health Tweet in each LGA

4.2.2.2 Distress Rate

It is believed that distress is considered an indicator of mental health for individuals. We pulled the data from AURIN published by "TUA_PHIDU" about psychological distress targeting individuals aged 18 years or older from 2011 to 2013. This data set contains the distress count and distress rate for each LAG based on the local population base. The statistics collected from Twitter are integrated with the data from AURIN about psychological distress so that a heat map that represents the level of correlation is generated and presented below 15.

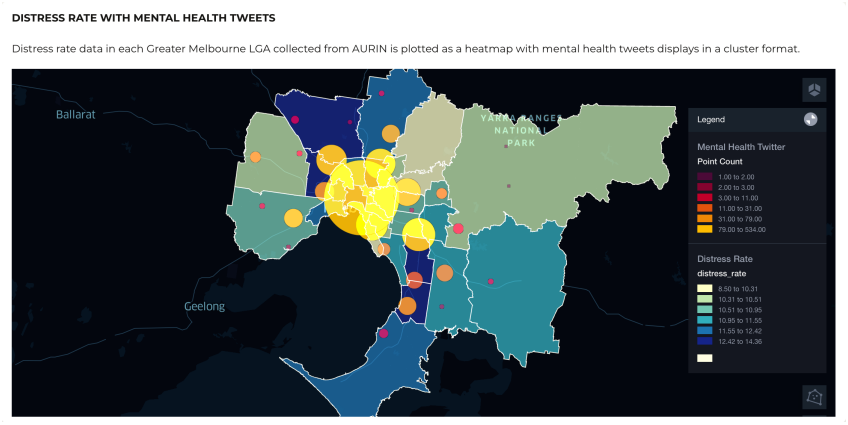

Figure 15: Heatmap Representation of Distress Rate

The level of distress rate is mapped to each LGA on the heat map represented with different colours. The darker blue indicates a higher distress rate. Greater Dandenong is the suburb that has the highest distress rate at 14.4 according to the data set from ARUIN. The suburb with the lowest distress rate is Boroondara, with a number of 8.6.

As for the number of related tweets, Melbourne City has the largest count of 222. The rest of the LGAs vary between 4 (Casey City) and 66 (Port Phillip Island). The distribution is highly biased by the data from Melbourne City since most of the tweets containing the keywords are posted from the CBD area. In this case, Melbourne City will be considered an outlier in the data set and can not represent Greater Melbourne. The size of circles on the heat map indicates the count of relevant tweets in that region.

By examining the generated heat map, it illustrates the correlation between the number of tweets with mental health keywords and the distress rate in regions of Melbourne. Apart from the CBD area with an extremely large number of tweets, the LGAs with darker blue colour are covered with larger circles on the map. This indicates a positive correlation is identified between these two factors and more tweets mention mental health is generally associated with a higher level of psychological distress.

The positive correlation suggests the number of tweets related to mental health topics could be an indicator of the average level of distress in that region. By using our Twitter harvesters and data analysis techniques, it is possible to make predictions of potential mental health issues for certain groups of people.

This could be applied in real settings to evaluate the psychological well-being of an individual or recognise the signs of mental health issues at an early stage. For example, a person who keeps complaining about the persistent stress from the study in school could be a warning for potential psychological problems.

Furthermore, such interventions could be set in place for those who exhibit similar characteristics.

4.2.2.3 Income

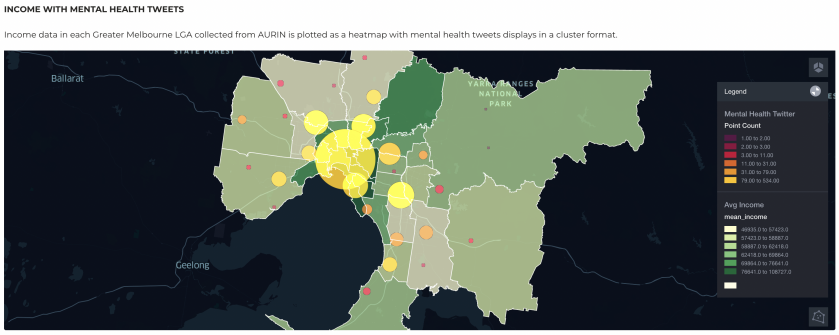

Figure 16: Heatmap Representation of Income

The Figure 16 above displays the number and the location of the mental relative tweets with size of circles and the income levels of each LGA with the heatmap. The larger size of circles indicates the larger number of tweets and the darker colour of the heatmap means the higher income level.

The centre of Greater Melbourne aggregates the most tweets related to mental health, and it is also the highest income level region compared to the surrounding regions. There are four cities with the low income level but large numbers of tweets recorded: Brimbank City, Hume City, Whittlesea City and Greater Dandenong City. The city with high income level but low number of tweets recorded also exists, which is Nillumbik Shire.

It can be deduced that the income level has a weak correlation to mental health. The regions with the higher income level aggregates the most mental relative tweets, but the distribution of the tweets may be affected by the population distribution. The higher income will attract citizens to locate in that area and contribute mental relative tweets. The change of the number of tweets from low to high income level is also not regular. The situation that areas with the low income level contribute larger numbers of tweets than the area with the high level also exists. Therefore, it is hard to conclude that the income level affects mental health from the existing dataset.

5 Issue and Challenge

In this section, we discussed and provided solutions to issues and challenges that we encountered during the development process.

5.1 Twitter Harvester Issues

The biggest challenge is our tokens. The restrictions come from 3 aspects: rate limit, functionality limit and information limit. For each token, the rate limit is 1 tweet/sec, which significantly restricts the total number of tweets we can get. Due to the limitations of the basic account, functionalities such as, geolocated search, are not available, which means that we can't decide if the tweets have geo information when making requests. Two scenarios in this project both require tweets with coordinates, while most tweets we get from Twitter don't have them. The only functional filter we can use in the request is keyword, which makes the second filter in the tweet harvester very strong.The most unexpected is that there are parts of tweets attached geo-information that do not contain the coordinates, which are place_id instead. We can't get the details from place id due to the token limitation. Therefore we have to give up tweets which only have place id as geo information. What's more, we can only get tweets from the last 7 days due to the token limitation.

In other words, the number of raw tweets we get from Twitter are limited by rate and date. Also the raw tweets only contain keywords we require. What we need is tweets with coordinates in the great Melbourne, which is a very strong filter. But the tweets we get are too little too general, it's basically searching for a needle in the ocean. That's why in both scenarios most tweets we use are historic data. To solve this problem, the best way is to get an academic account and a token with higher authority.

When running tweet harvester instances in MRC, we encountered an unexpected issue. Sometimes some instances would keep reporting "429 too many requests" while other instances could still run normally. We checked our tokens to make sure that each token was only used in a single instance and no one else used that token. And we also made sure that the requests were under the rate limit. But this issue kept happening. After a couple of hours, it would go back to normal. Our guess is that this is a Twitter side problem. Perhaps there were too many requests at the same time, and then Twitter will block some tokens for some time to ease the traffic. But this remains an unsolved issue and if our guess is right is still unknown.

5.2 Ansible

5.2.1 In-memory Inventory

The main challenge of writing Ansible scripts is we are expected to complete all tasks in a single playbook. Usually, an inventory file will be prepared for specifying a list of hosts and the tasks will be executed on those hosts. However, the instances on Melbourne Research Cloud need to be created using Ansible as well so it is impossible to know the IP addresses for those instances before execution. Therefore we can not use an inventory file to inform Ansible of the IP addresses of hosts.

A workaround solution would be the in-memory inventory. Unlike the inventory file, *in-memory inventory* is similar to a global variable and can be defined during the execution. We added the instances just created with the OpenStack module into a group called `MRC_NODES` in the first play of the playbook. Those instances now can be referenced while running the second play and deploying services on those instances. Besides, a particular host can be specified for running certain tasks by indexing the `MRC_NODES` group. This is particularly useful for deploying frontend only on instance 4 and other services only on instances 1-3.

```
# Add hosts to Ansible in-memory inventory
- name: Add host
  ansible.builtin.add_host:
    name: '{{ item.openstack.public_v4 }}'
    groups: MRC_NODES
  loop: '{{ os_instance.results }}'
  when: item.openstack is defined
```

Figure 17: In-memory Inventory

5.3 CouchDB

5.3.1 Error Handling

To ensure that CouchDB can handle errors such as database crashes or one of the VM instances that the CouchDB server is on failing, we used a CouchDB cluster on three separate nodes. A cluster-based setup allows the CouchDB server to continue running even if two of them fail. This results in better reliability overall, and failure of one database does not mean that the application can't be used.

5.3.2 Connection Issues from Windows OS

One of the biggest issues faced were the connection issues. On Windows OS, when connecting with the CISCO AnyConnect VPN to access the server instances on the MRC, some team members were unable to connect to the CouchDB server after setup. Using the LINUX distribution on windows (Ubuntu 20.04 accessed via WSL2), team members were getting connection lost and connection refused errors when running HTTP commands. This was a known issue with WSL2, and several solutions found online were used to resolve this.

5.3.3 Broken pipe error

While uploading the historical tweet data into CouchDB, a broken pipe error was encountered. On a localhost with a local CouchDB instance, the data could be uploaded without issue. However, when connecting to the CouchDB database deployed on the MRC instances, the broken pipe error persisted. This was due to potential networking issues. Ultimately, SCP was used to move the files onto the MRC instance first before it was uploaded to CouchDB and the issue was solved.

5.3.4 Docker Swarm

It was initially designed to run the CouchDB cluster in Docker Swarm mode. Similar to Docker Compose, Docker Swarm manages to host multiple containers as a single seamless service but across multiple servers. Services running on Swarm mode can provide high availability without having to worry about the failure of several servers.

However, we encountered troubles when configuring CouchDB in Swarm mode from different aspects. Firstly, these containers need to communicate over an isolated network so the outside can't disturb internal message exchange. This is relatively easy by manually creating a Docker network. Secondly, the hard part is how to consider them as a single service while setting up the cluster over multiple CouchDB. We tried with Docker compose file to pass environment configurations as well as preparing a CouchDB `config.ini` to initialise every CouchDB as recommended in official documentation.

The challenge is that the user password in `config.ini` has to be the hash of the password rather than plain text. We followed an article [2] to generate the hashed password and saved it in `config.ini` but it didn't work. It also mentioned that by replacing the `config.ini`, the entry point of the Docker image has to be defined again for proper functioning.

While three CouchDB containers are finally running, we failed to set up the cluster joining all containers. The usual way to set up a cluster is through the *Fauxton* GUI provided by CouchDB but this page is corrupted if we try to set up a cluster using Ansible. Another challenge is containers running in Swarm mode will be assigned with special node names rather than their normal IP addresses. This increases the difficulty for the following API calls as well.

We decided to deploy the CouchDB cluster without using Docker Swarm. Also the CouchDB will return an error at the end when finalising the cluster saying "unable to sync admin passwords", this does not affect the functioning of the cluster.

Figure 18: CouchDB Cluster Error

5.4 Front-end

5.4.1 Finding and Adjusting React Template

As we are unfamiliar with React, the processing of altering and adapting React template downloaded from an open-source site was very difficult and complicated. Nevertheless, after watching online tutorials on how to adapt React template. We managed to remove unnecessary pages and React components from the template and created a website that is suitable for our project.

5.4.2 HTTP GET request

The main challenge during the development of the front-end was fetching data from the backend server. As we were unfamiliar with the Rest API, thus we kept getting errors such as 404, 401 and CORS. It took us a while to solve this problem, but we were able to fix it in the end. The CORS error was fixed by enabling CORS from the CouchDB server. With authentication error, it was easily solved by adding an authentication header to the GET request. Moreover, there are 2 libraries available to fetch data, which were `node-fetch` and `fetch-with-proxy`. We initially thought that we were supposed to use `fetch-with-proxy` as the web app will be hosted on the MRC, hence the front-end must require a proxy in order to fetch the data from CouchDB server. However, it turned out that we do not need that and `node-fetch` was a better option as it did not produce error 404 when we ran the fetch command.

5.5 Docker Issues

5.5.1 Anonymous Volume Mount

The volumes mounted to Docker containers were not properly configured at first. It was assumed that bind mount has been defined when launching containers but then it was noticed that data generated with CouchDB can't be seen on external volumes as it should be.

Then it was found that Docker will create an anonymous volume which is managed by Docker itself and assigned to each container by default. This will lead to the issue that every time the ansible script is executed, the Docker container will be initialised again with a new anonymous volume attached to it. This becomes serious in terms of CouchDB because the data collected before will be "removed" (actually unlinked) and everything will be set to default.

5.5.2 Bind Mount

By looking up the Ansible document and modules for the Docker container, it is finally properly configured to mount a volume with a bind mount. However, the error returned from CouchDB reminds us that the bind mount would not automatically copy the existing files inside the container to newly attached volumes. Instead, the bind mounted volume will overwrite everything that already exists inside containers. The configuration files are overwritten so that CouchDB won't even start running.

Then we proposed a solution to copy the files from containers just initialised to the external volumes we intend to mount. After that, we stop the containers and re-launch them with proper configurations to bind mount volumes. Eventually, the CouchDB is running with all data generated correctly stored on volumes of the host machine. We can now easily SSH to the host machine and check the log files directly without having to execute commands with Docker interactive mode anymore.

5.6 AURIN

5.6.1 Data Collecting

AURIN system contains many data related to the scenarios of our research, which are crime and mental health. There are many different levels of information differentiated according to postcode, SA2, SA3 and LGA. It was challenging to find the right datasets that match our project research area of Greater Melbourne only. As there was not any dataset that only focus on Greater Melbourne. Thus, we decided to categorise Greater Melbourne according to the Local Government Areas Code (LGA) instead, as the data at the LGA level is available in AURIN for us to download.

5.6.2 Data Processing

After we found the right dataset, the raw data contained many different types of unnecessary data from other LGAs that are not in Greater Melbourne. Therefore, these data have to be filtered out before we use them in our analysis and visualisation. This took quite a long time, we need to find and record each LGA name and code manually in order to filter out the unwanted data.

6 Video Demo Link

Playlist

https://youtube.com/playlist?list=PLNH1Fhe-gYxM0mh9OAEuU9f57RHeB2Llb

Individual video

Twitter Harvester Demo

https://youtu.be/N6S-jMLzoKY

Ansible Demo

https://youtu.be/-zO3tLMP2BE

Attach LGA Historical Twitter Demo

https://youtu.be/Bj3L0NLXEwQ

CouchDB Views Demo

https://youtu.be/RD6homKl7M4

Front-end Demo

https://youtu.be/ISGNG8cmOcA

7 GitHub Link

https://github.com/jackson-hu1279/Livability-of-Melbourne

8 Front-end Link

The front-end webapp is accessible through The University of Melbourne Internet or Anyconnect VPN at:

http://172.26.134.255:3000

9 Individual Contributions

Name	Contributions
Renwei Hu	Collect list of keywords for scenarios Manage and monitor Docker containers Write Ansible scripts MRC instance configuration Set up CouchDB cluster Deploy harvesters Deploy frontend Record demo video of Ansible Create system architecture graph Prepare presentation slides Write the report
Siwat Chairattanamanokorn	Develop React Front-end Webapp Create front-end visualisation of data from CouchDB Setup HTTP GET request from backend to frontend Collect and process data from AURIN Collect and process Greater Melbourne LGA geojson Record Demo video for Front-end Prepare presentation slides Report Latex editor Write the report
Chi Yin Wong	Open and extract info from geojson file Filter and extract historical tweets Attach LGA to historical tweets HTTP requests to upload data to CouchDB (bulk_docs) CouchDB Views Record demo of attaching LGAs Record demo of view functions Prepare presentation slides Write the report
Renkai Liao	Design and test tweet harvester Design harvester instances for scenarios Record harvester demo video Prepare presentation slides Write the report
Kaiquan Lin	Design and test the tweet harvester Prepare presentation slides Write the report

References

[1] T. U. o. M. The Universtiy of Melbourne. [Online]. Available: https://docs.cloud.unimelb.edu.au/.

[2] A. Hsu, *Setup couchdb cluster using docker swarm*, Jan. 2021. [Online]. Available: https://ssarcandy.tw/2021/01/26/setup-couchdb-using-docker-swarm/?fbclid=IwAR30Xn-lw4IQ-66bw6rNq5GQjAgT6m%20CsE9Mja83gVmbGAlTygPzCMnhvWYg.