COMP90024 CLUSTER AND CLOUD COMPUTING: City Analytics on the Cloud

Group 17

Jeanelle Abanto
Student ID: 1133815
School of Computing and Information Systems
University of Melbourne
Melbourne, Australia
jabanto@student.unimelb.edu.au

Radhimas Djan
Student ID: 1146240
School of Computing and Information Systems
University of Melbourne
Melbourne, Australia
djanr@student.unimelb.edu.au

Zi Jin
Student ID: 987771
School of Mathematics and Statistics
University of Melbourne
Melbourne, Australia
zjjin@student.unimelb.edu.au

Kartika Waluyo
Student ID: 1000555
School of Mathematics and Statistics
University of Melbourne
Melbourne, Australia
kwaluyo@student.unimelb.edu.au

May 19, 2021

ABSTRACT

This is a report to accompany our cloud-based solution to analyse Twitter data collected using Twitter API to the CouchDB cluster deployed in Melbourne Research Cloud. The data collected from Twitter is used to find a relationship between tweet sentiments and AURIN data on income, social support and unemployment rate across Sydney, Melbourne, Brisbane, and Adelaide. We discuss in detail the methodologies implemented to collect tweets from Twitter API, the data we collected and processed from AURIN, the architecture of the cloud infrastructure hosting the backend and frontend of our web application, the features of our web application, and the findings from our analysis.

Table of Contents

1 Student Roles

1.1 Jeanelle Abanto

Worked on MRC instances and Ansible scripts, CouchDB cluster deployment, tweet streaming and searching, Docker swarm, containers, and scalable service deployment.

1.2 Radhimas Djan

Worked on tweet collection, making the backend, collaborative effort on making the frontend with Kartika Waluyo, connecting frontend and the backend, making the mapreduce on the couchdb.

1.3 Zi Jin

Worked on AURIN and past Tweets collection, feature engineering and scenario analysis.

1.4 Kartika Waluyo

Worked on the frontend together with Radhimas Djan.

2 Introduction

Nowadays, with the popularity of social media, information can be shared across the world more and more easily. Statistics have shown that Twitter, as one of the most popular social media platforms, has 192 million daily active users along with 500 million daily tweets. Through the use of Twitter, people can post anything about their daily lives and express their sentiments and thoughts. Thus, the understanding of the sentiment in each tweet may provide extra information regarding the daily life condition and living environment for a population, such as which factor may bring people happiness and which factor the population is suffering from.

The objective of this project is to explore some social features that may correlate with the sentiment of the tweets. In particular, this project would mainly focus on three factors namely, income, social support, and unemployment rate with respect to the tweets located in the four major cities in Australia – Adelaide, Brisbane, Melbourne and Sydney. The detailed observations and results are discussed in Section 5.

To achieve the objective of this project, a cloud-based application is developed using the resources provided by The University of Melbourne on the Melbourne Research Cloud. The main technologies and tools used for the deployment of the applications are Ansible and Docker. Ansible is an automatic operating technology that is used to deploy the cloud instance and control the remote hosts. A container technology, Docker is also used in this system to package the application and its dependencies, hence ease the process of shipping and running of the application. The system architecture, technologies, along with the deployment of various components are described in Section 3.

For the data collection of this system, two sets of data, i.e., tweets and socioeconomic data have been collected using Twitter API and AURIN respectively. The collected data were stored in CouchDB after a series filtering and preprocessing. The collection and method of processing data is discussed in Sections 3.2 to 3.4.

The result of the exploration of relationships between some socioeconomic features and tweet sentiments is presented and can be observed on the web-based application. The web page is designed with the backend components used to fetch and create views of the required data from CouchDB, along with the frontend components to visualize the result in pie graphs. The techniques used for the frontend design are React and nodejs. A discussion on the details of the web application can be found in Section 3.5.

3 System Architecture

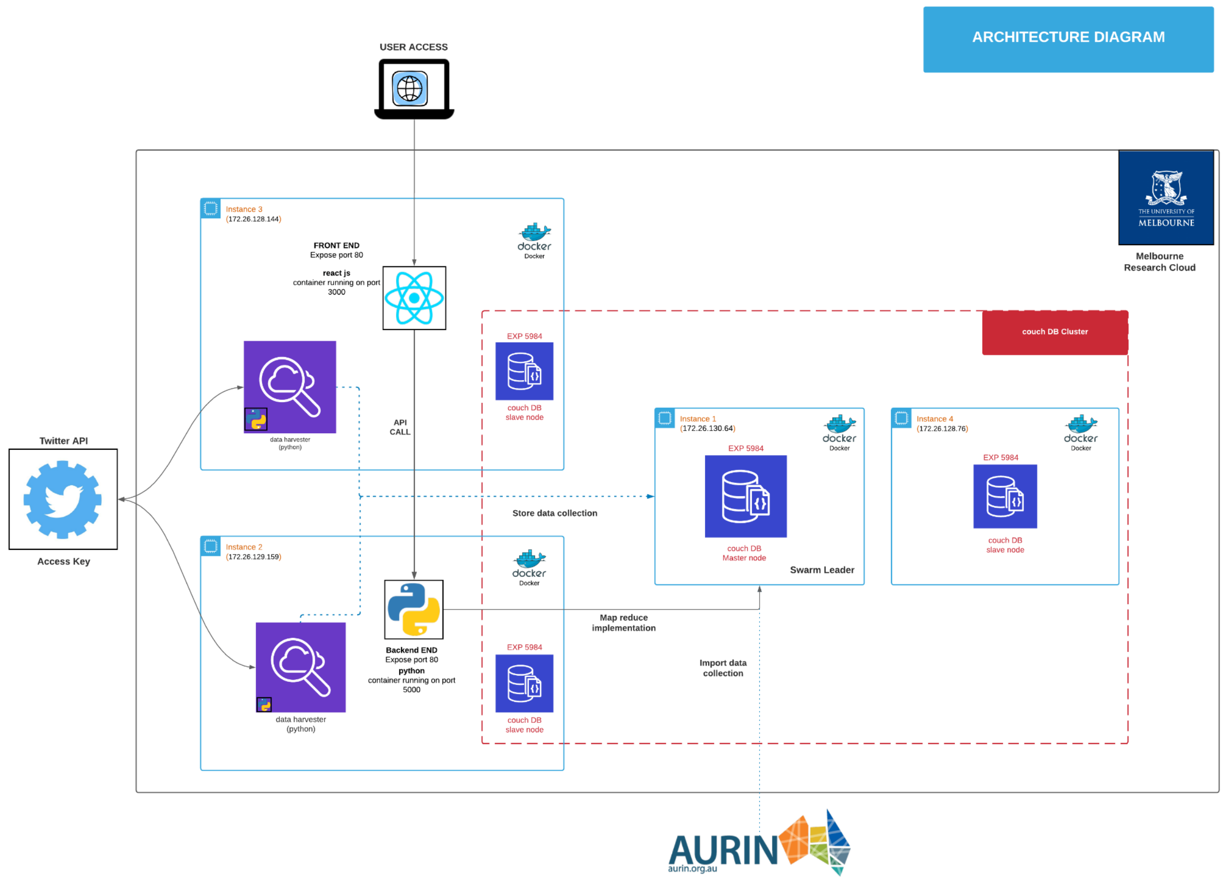


Figure 1. System Architecture Diagram

The system architecture of our project can be seen in Figure 1. We use a docker swarm to manage and schedule the harvester service on any up to all four of the instances on the Melbourne Research Cloud. Instance 1 acts as the swarm leader as well as the master node of the CouchDB cluster, whereas instances 2, 3 and 4 are worker nodes on the swarm and on the db cluster. The backend is deployed on instance 2, and the frontend on instance 3 where the user can connect to access the web application. The twitter harvester, as a service deployed on the swarm, is scalable to run on any subset of the four instances, if necessary.

Our cloud-based solution is deployed on the Melbourne Research Cloud (MRC) which offers free on-demand computing resources to researchers at the University of Melbourne (and affiliated institutions). It provides similar functionality to commercial cloud providers such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform. Using this service allows us access to scalable computational power without the overhead of spending precious time and money setting up our own compute environment in commercial cloud providers.

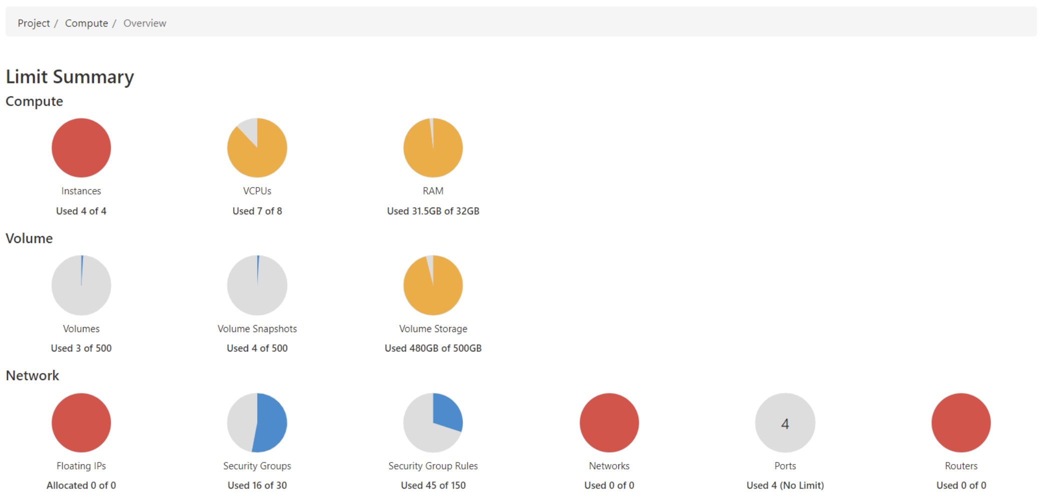
Our solution is deployed on instances with:

- **Availability zone:** melbourne-qh2-uom
- **Network:** qh2-uom-internal
- **Flavor:** uom.mse.2c9g (instances 1-3) which has 2 virtual CPUs and 9GB of RAM, and uom.mse.1c4g (instance 4) which has 1 virtual CPU and 4.5GB of RAM
- **Image:** NeCTAR Ubuntu 20.04 LTS (Focal) amd64

3.1 Melbourne Research Cloud

The Melbourne Research Cloud (MRC) provides Infrastructure-as-a-Service (IaaS) cloud computing to the University of Melbourne researchers, providing access to a robust set of on-demand virtualized computing resources (such as servers and storage).

3.1.1 Resource Allocation



Compute

Instances
Used 4 of 4

VCPUs
Used 7 of 8

RAM
Used 31.5GB of 32GB

Volume

Volumes
Used 3 of 500

Volume Snapshots
Used 4 of 500

Volume Storage
Used 480GB of 500GB

Network

Floating IPs
Allocated 0 of 0

Security Groups
Used 16 of 30

Security Group Rules
Used 45 of 150

Networks
Used 0 of 0

Ports
Used 4 (No Limit)

Routers
Used 0 of 0

Figure 2. Resources allocated on MRC

We are provided access to create up to 4 instances, and 8 virtual CPUs to utilise with a total of 32GB of RAM on the Melbourne Research Cloud. We are also provided access to create up to 500 volumes for the instances with a total storage capacity of 500GB. We allocate these resources to build our solution as shown in Figure 2.

Using 4 instances on the MRC, we deploy a Twitter harvester that collects tweets using Tweepy to connect to Twitter API, a CouchDB cluster to store the tweets and support MapReduce functionality for data analytics, and the frontend web application that shows the different analytical scenarios we implemented on the tweets along with the data collected from AURIN.

3.1.2 Advantages

In this section, we will be listing down and discussing some of the advantages of using the Melbourne Research Cloud.

3.1.2.1 Cost

The main advantage of using the Melbourne Research Cloud for our application is that it is free to use for students and researchers of The University of Melbourne. Since it provides usage of cloud infrastructure for free, we do not have to worry about the costs of using commercial cloud providers such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform for the purposes of building and deploying our not for profit application. Moreover, having the compute power on a free cloud service provider saves us the overhead cost of procurement of applications and infrastructure.

3.1.2.2 Accessibility

It provides on-demand access to instantly build servers and gain compute power without the hassle of planning, purchasing, and maintaining our own hardware to act as servers. It is ideal to use for our purposes which is a test configuration and deployment of a simple cloud-based solution. Moreover, it is free to access 24/7 anywhere in the world thus, it provides a free and easy way to collaborate among developers in the team.

3.1.2.3 Scalability

Similar to commercial cloud providers, the Melbourne Research Cloud provides flexibility to create and manage our own dynamic and scalable cloud infrastructure. It allows users to spin up or tear down instances with scripts such as Ansible.

3.1.2.4 Security

It offers a high level of security wherein only authorized users can access the tenancy allocated to students and research teams. It allows users to choose their own key pair to attach to the servers or instances. Similar to commercial cloud providers, users have the ability to set the security groups of the servers to limit or open access to the servers as needed.

3.1.2.5 Collaboration

As a secure environment to test and deploy applications, it allows users to access other researchers’ work to save valuable time on their own work. Users can also publish their research on the cloud to allow other researchers to use their work which in turn can help grow their own portfolio. It also allows users to create a backup of their work which they can share by creating an image of the instance(s).

3.1.3 Disadvantages

In this section, we will be listing down and discussing some of the disadvantages or issues we encountered when using the Melbourne Research Cloud.

3.1.3.1 Management

Since the Melbourne Research Cloud is an Infrastructure as a Service (IaaS) provider, the team has to learn how to manage the infrastructure effectively. Considering that none of the members in the team is an experienced system administrator, setting up, configuring instances, and deploying applications on the Melbourne Research Cloud is no trivial task.

3.1.3.2 Accessibility

The accessibility advantage is mostly applicable to users accessing the cloud from The University of Melbourne. For students or researchers accessing the cloud resources from outside, e.g., outside Australia, especially during this time of the COVID-19 pandemic, it is necessary to connect to a Virtual Private Network (VPN). Accessing resources through a VPN is not an issue in itself, however using the provided VPN causes issues e.g., internet connectivity becoming unstable thus hindering research, difficulty connecting to VPN, or getting disconnected from the VPN. Getting disconnected from the instances during configuration or deployment would result in errors which will require repeating a whole process. The Melbourne Research Cloud may be a free option to access anywhere in the world, however it is necessary to consider the risks and issues associated with VPN connectivity.

3.1.3.3 Security

Although users are provided with a high level of security wherein only authorized users can access the tenancy allocated to them, there is no guarantee that the resource allocated is exclusive for their use alone. It is possible that one resource is allocated multi-tenancy thus sharing the resource among users. If one or more users on the same resource are using an inordinate amount of computing resources, this could affect and slow down performance for other users. There are other possible security risks associated with this architecture, and users can only trust that the Melbourne Research Cloud is configured correctly and securely to mitigate these security risks.

3.2 Twitter API

Using a standard developer account in Twitter, we collect tweets by streaming for live tweets and searching past tweets in Melbourne, Sydney, Brisbane, and Adelaide. With Tweepy for Python 3, which is an easy-to-use Python library for accessing the Twitter API, creating an API connection to Twitter, streaming, and searching tweets is simplified. This library allows us to handle and catch common errors with ease such as exceeding the search and stream rate limit and setting a sleep time enough to recover and reconnect to Twitter API to collect tweets continuously.

3.2.1 Data Collection

Since most tweets now do not have geolocation information e.g., coordinates and other location data, we stream live tweets using the bounding boxes of Melbourne, Sydney, Brisbane and Adelaide. To differentiate among the cities, we take advantage of multiprocessing i.e., each process is tasked to collect tweets from a city thus, we have four processes - one process streaming tweets from Melbourne, one from Sydney, one from Brisbane, and another from Adelaide. Similarly, for searching tweets, we also create four processes - each searching for tweets from one city. However, due to the difference in input parameters of the stream and search APIs, we do not use bounding boxes for the search. Instead, we search for tweets within a radius from the coordinates of each city.

Spawning processes to stream and search one city each allows us to identify which city a tweet is from when we preprocess the tweets before saving them to CouchDB even without geolocation information. This helps to simplify our preprocessing since we can tag the location of the tweet as the city information from the process that collected the tweet. Moreover, we do not have to discard tweets without geolocation which could affect the number of tweets we could use for our analysis.

3.2.2 Data Processing

Once a tweet is collected by a process, we preprocess the tweet before saving to CouchDB. First we remove mentions, hyperlinks, and hashtags from the text tweet as these are not necessary when we do sentiment analysis. We chose to use VADER (**V**alence **A**ware **D**ictionary for s**E**ntiment **R**easoner) to compute the sentiment scores of each tweet - a sentiment analysis tool that is specifically attuned to sentiments expressed in social media. After preprocessing and computing the sentiment scores of the text tweets, we extract the tweet ID, the date and time the tweet was created, the preprocessed text tweet, the city information from the process, the sentiment scores, and the sentiment of the tweet - whether it is positive, negative or neutral.

3.2.3 Data Storage

The data extracted from preprocessing the tweet are the only data we save to CouchDB. In order to handle duplicate tweets collected by the harvesters, we use the tweet ID as the unique identifier of an entry in CouchDB. We use the tweet ID as the unique identifier because tweet IDs created by the Twitter API are unique, hence we can query the database if the tweet ID is existing before saving new tweets to the database.

3.3 CouchDB

To store the data we collected from Twitter and AURIN, we use CouchDB - an open-source NoSQL database. It uses JSON format to store documents which is suitable for our application since the data we are collecting from Twitter is also in JSON format. This allows us to store raw data collected from Twitter directly to CouchDB if needed.

To retrieve data stored in CouchDB, we create views using MapReduce. Views contain rows of data that is sorted by a key e.g., sorting our data using city as a key to sort tweets based on city. This feature is not specific to CouchDB, however it is a powerful feature to process stored data for data analytics. Using MapReduce, we were able to process over 2.5 million tweets stored on our database for the different data analytic scenarios implemented on the web application.

We deploy CouchDB in a cluster on the cloud to allow us to replicate our database on different nodes. Deploying a cluster also allows us to save and access the same data from different servers due to its redundant data storage. Thus, even if a node fails, we can still access our data saved on other nodes and employ other replication measures to keep our data safe until we restore the node in the cluster that failed. Moreover, this allows us to distribute the load among applications, e.g., the Twitter harvester can continuously save tweets to an instance of CouchDB different from the instance that the backend API is retrieving data from.

3.4 AURIN

In order to relate the information gained from the tweets with the real world data, a collection of data is selected from the AURIN database. AURIN is an infrastructure with thousands of real data stored, and can help the researchers to make evidence-based decisions quickly and confidently. Since one of the main objectives of this project is to explore a series of factors that would have correlation with the tweet sentiments, including income, social support and unemployment rate, the corresponding data is collected from AURIN.

For income, the “income” dataset provided by ABS is used, it consists of all the related information about income from 2011 to 2019. However, considering the objective of our report and the information interested such as the median and mean income per year is only available up to 2017, the features could only be extracted from 2014 to 2017. After processing the data, we have included the following features for each city in our project: the median and mean income per year in AUD, the number of earners and the median age of the earner.

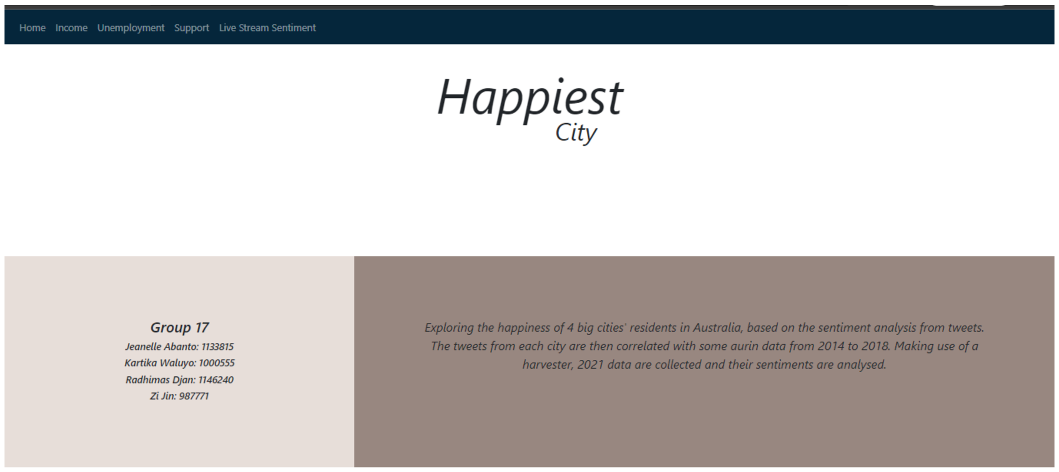
For the factor of social support, we have focused on the four kinds of support provided from the state government, that are age pension, benefit for sole family, disabled people and unemployed people. With the use of the “income support recipients” provided by PHIDU, the following features are extracted: 1) the percentage of old people receiving age pension (i.e. age pension receiver / people age 65+); 2) the percentage of female sole family receiving benefits (benefit receiver / total female population aged 16 to 64); 3) the percentage of disabled people receiving benefits (benefit receiver / total population aged 16 to 64); 4) the percentage of unemployed people receiving unemployment benefits (benefit receiver / total population aged 16 to 64).

For the unemployment rate, with the use "smoothed unemployment" dataset provided by DESE, we have extracted the average unemployment number, as well as the average unemployment rate of each quarter of the year for each city from 2014 to 2018.

On the other hand, one of the limitations of our investigation can be found, which is the lack of aurin data to support our result. Since we have included the tweets from 2014 to 2021, the real world data should be in the same time range as well, in order to give a more confident and evident conclusion of our findings. However, due to the limitation of the AURIN database, it is almost impossible to find the related data for all 2014 to 2021, hence the improvement can be made in the future with more real world data given.

3.5 Web Application

The server side including API handlers are created using Node.js Express, applying ReSTful designs. The client side is created using React framework.



Happiest
City

Group 17
Jeanelle Abanto: 1133815
Kartika Waluyo: 1000555
Radhimas Djan: 1146240
Zi Jin: 987771

Exploring the happiness of 4 big cities' residents in Australia, based on the sentiment analysis from tweets. The tweets from each city are then correlated with some aurin data from 2014 to 2018. Making use of a harvester, 2021 data are collected and their sentiments are analysed.

Figure 3. Home page

Figure 3 is a screenshot of the home page. Zooming into the navigation bar (Figure 4), there are 4 different page options to be explored.

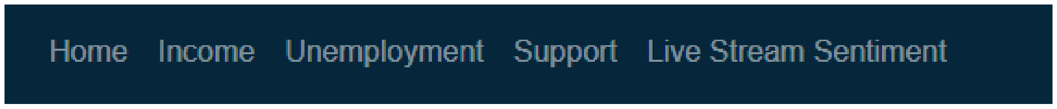


Figure 4. Navigation bar

The options are income, unemployment, support, and live stream sentiment.

Figure 5 is the map visualisation, which is the same for every page.

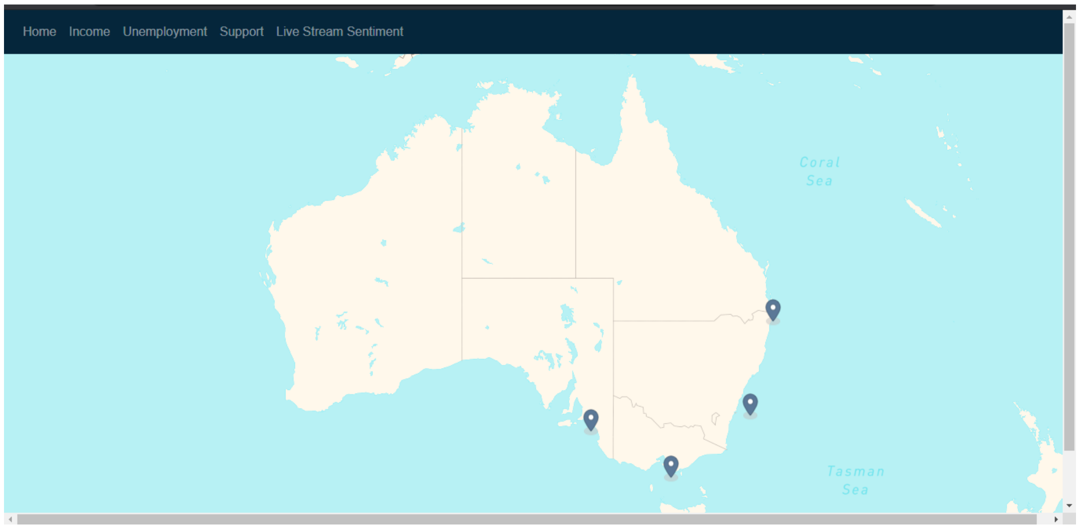


Figure 5. Map visualization

When one of the markers is clicked, some information regarding the chosen city will be displayed (Figure 6), and if the mouse is hovered over the pie chart, further information will be shown. The analysis that made up the pie charts will be discussed in Section 5.

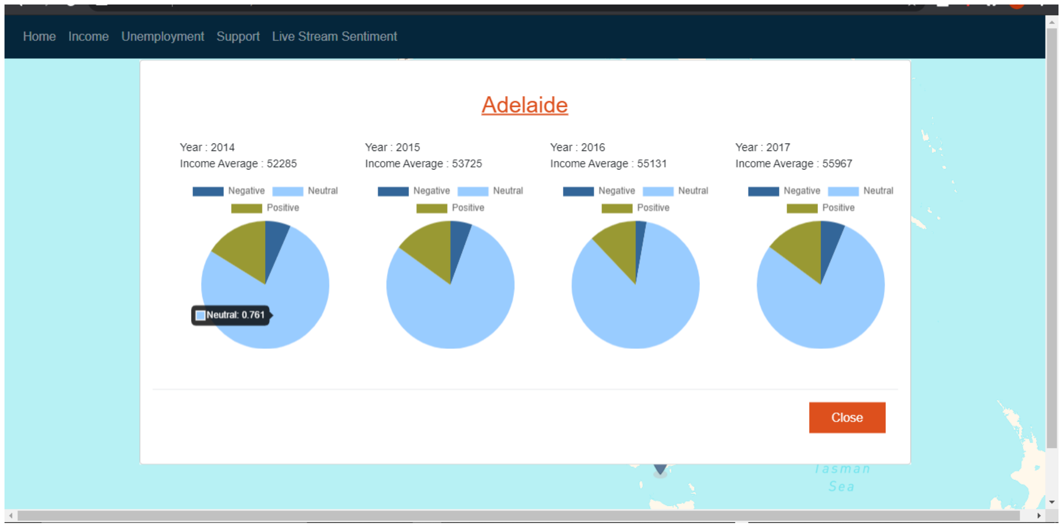


Figure 6. Data visualization

3.6 Docker

We deploy our applications on the cloud using a container to allow us to package our application and its dependencies into one succinct manifest. Deploying our applications in a container allows us to replicate and share the application across the team and machines in the cloud. No matter the flavor our machines are using, e.g., Windows OS, Ubuntu, or Mac we can replicate and run the application by simply executing the runnable instance of an image of the application - the container. For our solution, we use Docker which is an open platform for developing, shipping, and running applications.

Docker is highly portable and lightweight which makes it easy to dynamically manage workloads, scale up or tear down applications and services as necessary in real time. Since Docker is lightweight, it is a cost-effective alternative to hypervisor-based virtual machines which allows us to use more of the compute capacity for other applications or services. Moreover, it allows us to deploy our application in a cluster of nodes using Docker Swarm - a tool for clustering and scheduling Docker containers to provide redundancy and faster scaling on demand.

The CouchDB cluster is deployed on the cloud in Docker containers which allow for easy deployment of CouchDB by simply starting a Docker container with the latest CouchDB image. Similarly, the Twitter harvester is deployed as a Docker service on a Swarm which can be scaled up or down as needed with one command, instead of manually going through each node on the cloud and deploying or tearing down the harvester when the need arises. Furthermore, since the harvester is deployed as a service on the swarm, should an error occur the swarm leader can manage the service and attempt to deploy the service on another node in order to meet the number of services required.

The backend and frontend web applications are also deployed in Docker containers. However, since the Docker images of both applications were built and deployed separately, the applications were not deployed as scalable services on the swarm. The deployment of the CouchDB, the harvester, backend, and frontend as one application using Docker stack can be an improvement on the system to make the whole system scalable on the swarm. Moreover, setting up multiple swarm leaders is another improvement that can be employed so that if a swarm manager node fails, other swarm leaders can lead and manage the services deployed on the swarm.

3.7 Ansible

In order to create and configure instances and deploy our applications in the Melbourne Research cloud, there are various tedious tasks involved such as installing dependencies or configuring the proxy settings on each instance. Doing this manually for a small set of hosts may be doable in a short amount of time, however configuring and deploying applications on a large set of hosts could take an enormous amount of time for a system administrator. To automate these tedious tasks of cloud provisioning, configuration management, application deployment, intra-service orchestration, and more, we use Ansible - a radically simple IT automation engine.

It uses no agents and no additional custom security infrastructure, so it's easy to deploy - and most importantly, it uses a very simple language (YAML, in the form of Ansible Playbooks) that allows us to describe automation jobs in a way that approaches plain English. The Ansible playbooks we used for this project are described in more detail in Section 6. These playbooks are used to connect and deploy all the components of the system architecture to the Melbourne Research Cloud.

4 System Functionalities

The system functionalities achieved by our cloud-based solution include:

1. Automatic and continuous harvesting of tweets from Twitter API for social media analytics.
2. Storing collected tweets to a CouchDB cluster.
3. Using MapReduce functionality of CouchDB to support data analytics along with collected socioeconomic data from AURIN.
4. Visualize three scenarios in an attempt to find correlation between sentiments and socioeconomic factors namely, income, unemployment rate, and social support in Melbourne, Sydney, Brisbane, and Adelaide.

We chose to explore data on the major cities Melbourne, Sydney, Brisbane, and Adelaide as these cities have the highest population across Australia. Adelaide was chosen instead of Perth, which has a slightly higher population (but both considered regional centres), since it sits closer to the other three major cities and thus, could have more similar economic status and be representative enough to draw some conclusions.

5 Scenario analysis

5.1 Scenario 1: Income & sentiment

For this scenario, to distinctly observe the correlation between income and sentiment, we used the average income as the only feature. After observation, we can see that Sydney has the highest average income among the four cities from 2014 to 2017, with over 60,000 AUD per year. Whereas Adelaide has the lowest average income per year around 50,000 to 55,000 AUD. When comparing the average sentiment score for the two cities, the highest positive score of 0.167 can be found in Sydney, 2014, whereas the lowest positive score of 0.11 can also be found in Sydney, 2016 (shown in figure 7&8). On the other hand, Adelaide has a relatively stable and uniform distributed positive score, which is around 0.14 for most of the year. From the pattern observed, we may conclude that income may reflect the happiness of the population to some extent, but it is not very reliable to consider the income as the only variable. Since higher average income (i.e. Sydney) can not always lead to a higher positive sentiment, but lower income may lead to a relatively stable sentiment.

Sydney

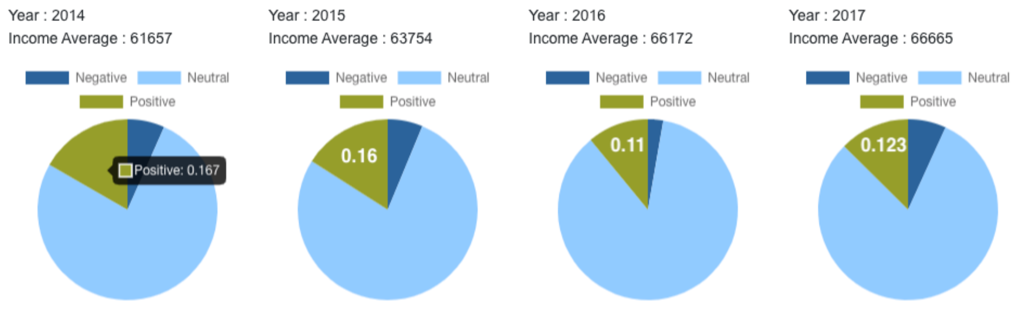


Figure 7. Income & Sentiment for Sydney

Adelaide

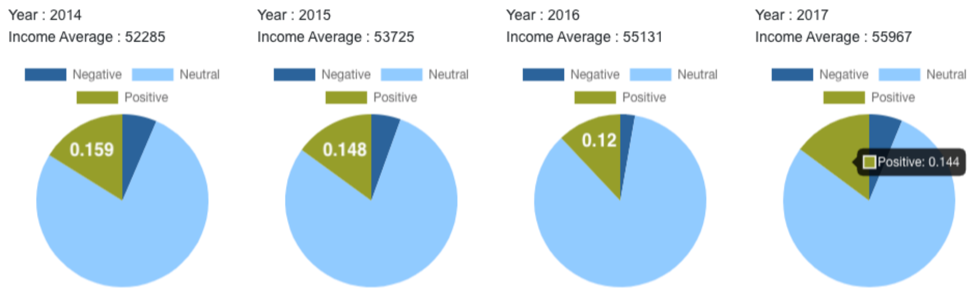


Figure 8. Income & Sentiment for Adelaide

5.2 Scenario 2: Social support & sentiment

After observing the statistics regarding the social support received for each city, we have found that in general, Brisbane has provided relatively higher quality of the social benefit services especially for age pension and sole family support, that is, a higher proportion from these special communities is able to receive the social support. However, Adelaide can be found to provide relatively lacking social support with only approximately 33% of older people having an age pension. However, when integrating with the sentiment score of tweets, surprisingly, the positive sentiment score of Adelaide is higher than Brisbane for all years (shown in figure 9&10). One possible reason may be that these special communities receiving social support is only a small proportion of the tweets creator, hence the sentiment score obtained may not be a very supportive and confident evidence to prove the correlation between social support between the overall happiness for the whole population.

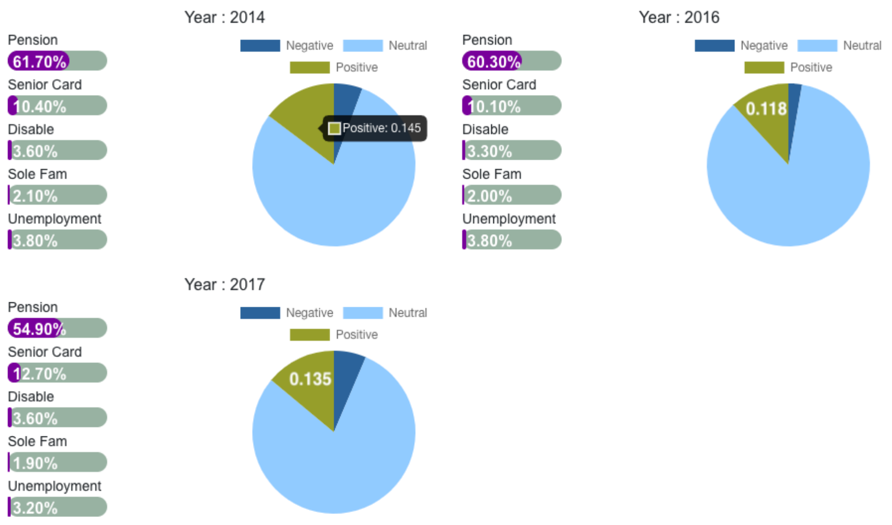


Figure 9. Social Support & Sentiment for Brisbane

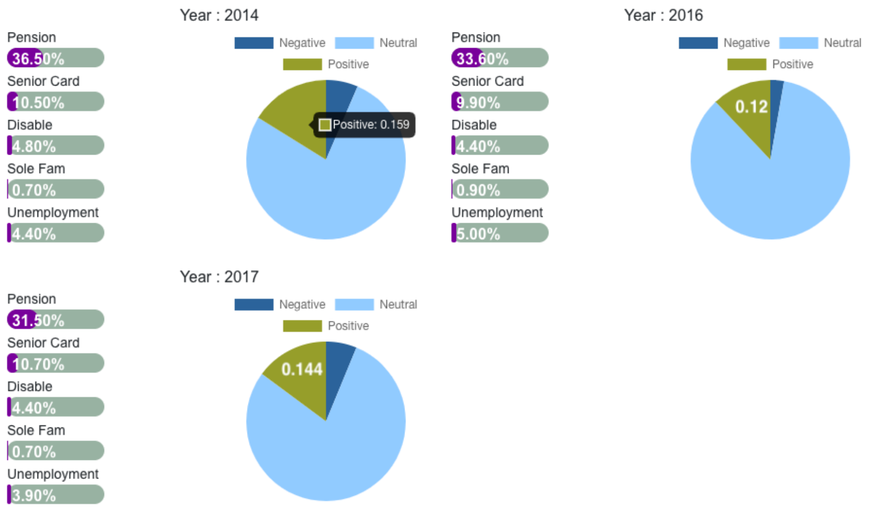


Figure 10. Social Support & Sentiment for Adelaide

5.3 Scenario 3: Unemployment rate and sentiment

For the unemployment rate, it can be found that Melbourne has relatively the lowest unemployment rates around 4% from 2014 to 2018 among all cities. And the highest unemployment rate can be found in Adelaide, which has the range form 6.5 - 9.52%. However, when comparing with the positive sentiment scores for the two cities, Melbourne is found to have a lower positive sentiment score than Adelaide for all years (shown in figure 11&12). However, considering the real world situation, unemployment rate is not likely leading to positive sentiment. Thus, the possible explanation for our result is that the tweets we collected may only include a very small amount that is actually sent from the unemployed people, therefore the connection between the sentiment calculated and the unemployment rate may be weak. However, when considering the results from all the scenarios, another explanation could also be found, that is people in Adelaide are generally happier than the rest of the cities, regardless of the external factors such as income or job.

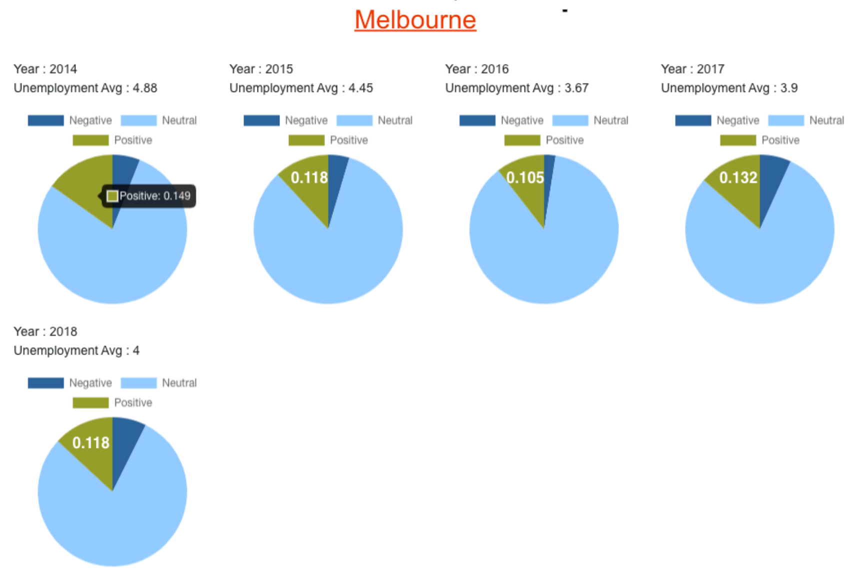


Figure 11. Unemployment rate & Sentiment for Melbourne

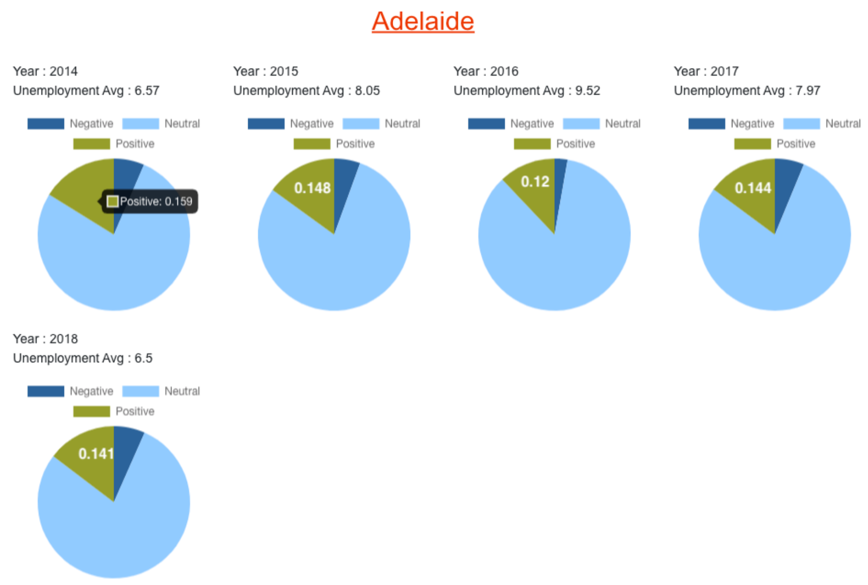


Figure 12. Unemployment rate & Sentiment for Adelaide

6 User Guide

In this section, we discuss the procedure to run the application on Melbourne Research Cloud. All scripts are run from the Ansible directory.

6.1 Deploy Instances

In this step, we define the volumes, security groups, and the instances we would like to create on the Melbourne Research Cloud using Ansible scripts. To do this we first install some dependencies such as Python-pip installation and update which is necessary for later installation of Python libraries, and installation of OpenStack SDK which is a client library used for building applications to work with OpenStack clouds such as the Melbourne Research Cloud. Next we define the volumes that we will attach to each instance, e.g., the volume size of each instance. We also define security groups which will control the inbound and outbound traffic on each instance such as allowing ssh, http, and https access from the outside world to each instance, and defining ports that all instances can use to communicate with each other. In order to create instances, we also need to define the image we would like to use for each instance, for this project we used Ubuntu 20.04 LTS (Focal) amd64 image for all instances.

To automate the above procedure, we used Ansible which is an IT orchestration engine that automates configuration management, application deployment and many other IT needs, suited for setting up and deploying our cloud-based solution on Melbourne Research Cloud.

To create our instances on the cloud, we run the shell script *run-nectar.sh* and choose option *1. Create instances*. This will run the Ansible playbook *deploy_instances.yaml* which automates the procedures previously discussed to deploy instances on Melbourne Research Cloud. A summary of the playbook to deploy instances on the cloud can be found in Table 1.

Hosts	**Roles**	**Description**
Localhost	Common	Install Python and OpenStack dependencies on localhost
	Volume	Create volumes on MRC from vars
	Security group	Create security groups and security group rules
	Keypair	Add the keypair on OpenStack
	Instance	Create instances on MRC Add hosts to Ansible in-memory inventory Create an inventory file containing host IP addresses

Table 1. Playbook to deploy instances

6.2 Allocate Hosts

We generate an inventory file with the host IP addresses labelled as either master node, worker nodes, harvester, backend, or frontend. To do this dynamically, we go to the *inventory* directory and run the Python script *allocate_hosts.py*. This reads and allocates the host IP addresses saved on the inventory file generated after the instances were created in the first step to a new inventory file *hosts.ini*.

To run the script, we use the following command, where *n* is the number of hosts we want to allocate for [-*t*] the Twitter harvester, [-*b*] the backend api, and [-*f*] the frontend web application:

```
python[3] allocate_hosts.py -t n -b n -f n
```

This script allocates unused resources first before re-allocating the same host to another application to prevent overloading one host with all applications when other hosts are free or unused.

6.3 Configure Instances

In this step, we configure the instances we created from the previous step by configuring proxy settings, installing dependencies, cloning the git repository, installing docker, and file systems on the volumes.

Before configuring the instances, we first create a copy of our keypair to *~/.ssh* path on localhost to enable us to ssh into the Melbourne Research Cloud using this keypair. Next is we set up the proxy settings on all instances we created on the cloud to enable internet access. We reboot the host machines afterwards to make sure that the proxy settings we configured on the environment are applied and working. After rebooting the host machines, we start installing the dependencies such as Python-pip, git, and other Python3 toolkits necessary to deploy our applications on Python.

Next, we configure our instances in order to clone our GitHub repository. To do this, it is a prerequisite that our GitHub account is SSH enabled, i.e., we have an SSH key associated with our GitHub account. We create a copy of this SSH key on *~/.ssh* path on the instances and create our working directory where we will clone the git repository. Once we have cloned the git repository, we delete the SSH key from *~/.ssh* path for security purposes, e.g., it is not advisable to have a private key exposed on the cloud.

Once we have cloned the git repository, we next install and configure Docker which we will use as the container of our application and the CouchDB cluster. We first uninstall any old versions of Docker before installing the dependencies and then installing Docker and Docker-compose. Once installed, we configure the proxy settings on Docker to enable internet access for the containers. To complete the configuration of instances, we also install a file system on each volume.

Before we deploy the CouchDB cluster and applications in succeeding steps, we also set up the Docker Swarm in this step. We first configure the Swarm on the master node by initiating a swarm and defining corresponding commands for the master and worker nodes to join the swarm. Next, we define the master node as the host manager or Swarm leader and advertise the master node's IP address to the worker nodes to allow worker nodes to join the swarm.

To configure the instances on the cloud, we run the shell script *run-nectar.sh* and choose option *2. Configure instances*. This will run the Ansible playbook *config_instances.yaml* which automates the procedures previously discussed for configuring the instances on the Melbourne Research Cloud. A summary of the playbook to configure the instances on the cloud can be found in Table 2.

Hosts	**Roles**	**Description**
Localhost	Groupkey	Create a copy of MRC keypair on ~/.ssh path
Instances	Proxy	Configure proxy settings on all hosts to enable internet access
	Common	Install dependencies
	Clone Git	Clone GitHub repository containing our application
	Docker	Install Docker dependencies, Docker, and Docker-compose
	Volumes	Install filesystems on the volumes
Masternode	Initiate Swarm	Set up and initiate Docker Swarm
Worker nodes	Join Swarm	Advertise master node IP address to worker nodes Join worker nodes to Swarm

Table 2. Playbook to configure instances

6.4 Deploy CouchDB and Twitter Harvester

Since we have already configured the instances on the cloud and we have cloned the git repository, we are now ready to deploy the CouchDB cluster which will store the tweets that our application will be harvesting, and the Twitter harvester. First, we deploy Docker containers which will run the CouchDB cluster in all the hosts we have defined as part of the Swarm, e.g., the master and worker nodes. Here, we define the container name, ports that CouchDB will use to communicate with each other in the cluster, and the CouchDB credentials. Once we have set up the container, we then set up the cluster on the master node and let the worker nodes join the cluster. To make sure that the cluster has been successfully created, we login to the any node (master or worker node) and run the membership check command:

```
for node in "${nodes[@]}";
do  curl -X GET "http://${user}:${pass}@${node}:5984/_membership";
done
```

Once we have confirmed that cluster membership is correct, i.e., all master and worker nodes are listed, we then proceed to pull the project's git repository on all hosts to make sure that we are running the latest version of the application. Since we deleted the SSH key in the previous step when we cloned the repository, we created a copy of the SSH key to the *~/.ssh* path of the harvester server to pull from the repository. After pulling the latest version of the application on the repository, we then delete the SSH key again for the same security reason as before. Then, we create a copy of the config file which will be used by the Twitter harvester to connect to Twitter API. The config file is a JSON file containing the Twitter developer credentials, stream and search keywords, location, and a since ID which allow us to search for Twitter IDs created after the since ID.

Next, we configure the Python file responsible for establishing connection to CouchDB with any of the host IP addresses of the CouchDB cluster. Once we have all the necessary configuration files setup on the cloud, we build the Docker image for the harvester which we will use to deploy the harvester application as a service. To deploy the application as a service on the swarm, we run the Docker-compose file which will build and use the harvester image to create a replicated service on the swarm to complete this step.

To deploy the CouchDB cluster and the Twitter harvester on the cloud, we run the shell script *run-nectar.sh* and choose option *3. Deploy CouchDB cluster and Twitter harvester*. This will run the Ansible playbook *deploy_harvester.yaml* which automates the procedures previously discussed for deploying the CouchDB cluster and the Twitter harvester on the Melbourne Research Cloud. A summary of the playbook to configure the instances on the cloud can be found in Table 3.

Hosts	**Roles**	**Description**

Database	Deploy CouchDB	Pull CouchDB image and deploy CouchDB Docker containers
Master node	Deploy cluster	Set up the CouchDB cluster
All	Pull git repository	Pull latest version of application from git repository, create config files, and build harvester image on the hosts
Harvester= Master node	Deploy harvester	Run Docker-compose to create harvester service on Docker Swarm

Table 3. Playbook to deploy CouchDB cluster and Twitter harvester

Here, we deploy the harvester on the master node since we want to deploy the image as a service on Docker swarm. Deploying the service on the master node allows the service to be deployed dynamically by the swarm leader. As a service, the harvester becomes a scalable application on the swarm.

6.5 Deploy Backend API

In this step we deploy the backend api which will host the necessary data to the frontend web application. First, we make sure that we have the latest version of source codes by pulling from the git repository. Then, we configure the python file responsible to establish connection to our database. Once configured, we build the docker container of the backend api to complete deployment.

To deploy the backend API on the cloud, we run the shell script *run-nectar.sh* and choose option *4. Deploy backend*. This will run the Ansible playbook *deploy_backend.yaml* which automates the procedures previously discussed for deploying the backend on the Melbourne Research Cloud.

6.6 Deploy Frontend Web Application

Lastly, we deploy the frontend web application on the cloud. We start by pulling from the git repository to make sure that we have the latest version of source codes. Then, we configure the address of the backend api on the *package.json* file. Once configured, we build the docker container of the frontend web application to complete deployment.

To deploy the frontend web application on the cloud, we run the shell script *run-nectar.sh* and choose option *5. Deploy frontend*. This will run the Ansible playbook *deploy_frontend.yaml* which automates the procedures previously discussed for deploying the web application on the Melbourne Research Cloud.

7 Links

7.1 GitHub Link

The version-control system we have chosen to use for collaborating is GitHub as it is the tool all members of the team are familiar with. Below is the link to our GitHub repository:
https://github.com/r4dhiDj/ccc_assg2

7.2 Youtube Link

Following is the youtube link for the project:
https://www.youtube.com/watch?v=riuTqRpOonY