COMP90024 Cluster and Cloud Computing Group 1 Assignment 2 Report

Qianjun Ding
1080391 Melbourne
qianjund@student.unimelb.edu.au

Zhiyuan Gao
1068184 Melbourne
zhigao@student.unimelb.edu.au

Jiachen Li
1068299 Melbourne
jiachen3@student.unimelb.edu.au

Yanting Mu
1068314 Melbourne
ymmu1@student.unimelb.edu.au

Chi Zhang
1067750 Melbourne
chzhang1@student.unimelb.edu.au

Abstract

This report is for illustrating the constructed system deployed on Melbourne Research Cloud's (MRC) virtual machines for investigating the topic of 'Melbourne the Most Liveable City…?' in COMP90024 Assignment 2. We will first present the deployment of instance and instruction to start the system in MRC with use of Ansible in Section 1 as an user guide followed by the illustration of advantages and disadvantages brought by the application for MRC in Section 2. Then in Section 3, the description of the system architecture is presented along with the illustration of the fault-tolerance design. In Section 4, the front-end (UI) design logic is presented. Data resource, preprocessing and the application of CouchDB are discussed in Section 5 based on the characteristics of the collected data and the in-build features from CouchDB. In Section 6, the logic of each data harvester is interpreted and shows how the summarised data on each desired topic from users are generated. Our selected scenarios are presented in Section 7 and 8 for showing the system can be applied for researching on the provided topic. Limitations of the system are discussed and potential solutions are proposed in Section 9, followed by the team member's workload table in Section 10 and the appendix list in Section 11.

Keywords: MRC, Ansible, CouchDB, Twitter, Aurin, Fault-tolerant, Liveable City

Contents

1 Ansible User Guide

1.1 Preparation

The entire Ansible automation setup is operated by 4 main yaml playbooks and each contains several roles to do in corresponding to different MRC instances. Before running the first step, we have to make sure that Ansible is installed locally and some other perquisites are needed to be followed as mentioned in the workshop.

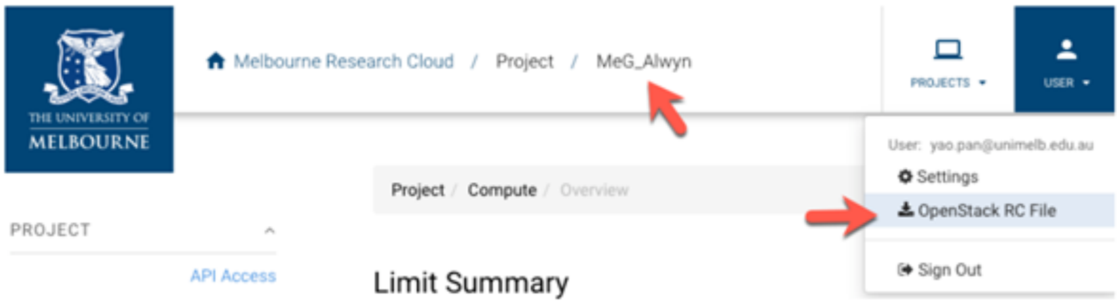
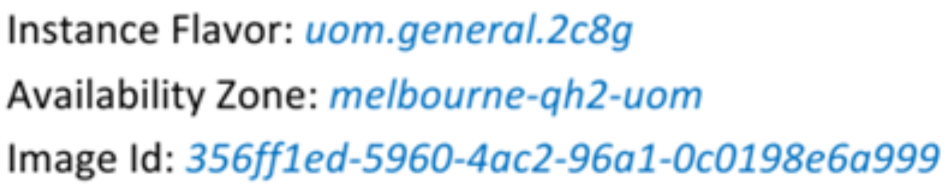

THE UNIVERSITY OF MELBOURNE
Melbourne Research Cloud / Project / MeG_Alwyn
PROJECTS
USER
User: yao.pan@unimelb.edu.au
Settings
OpenStack RC File
Sign Out
Project / Compute / Overview
PROJECT
API Access
Limit Summary

Figure 1: Melbourne Research Cloud GUI

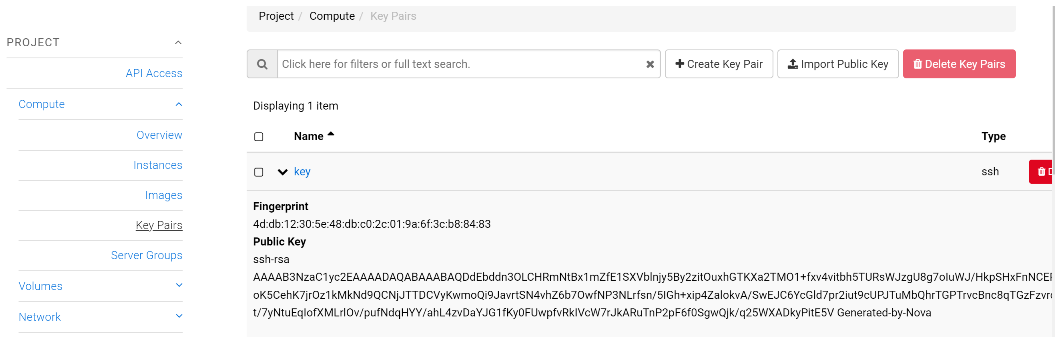
After logging into the MRC, the Project unimelb-COMP90024-2022-gp1 needs to be selected and the OpenStack RC File has to be downloaded as shown in Figure 1, dragging it to the Ansible folder as openrc.sh. The command line of running this file is required to be placed before operating any yaml playbooks for the purpose of verifying the OpenStack authentication. To have a password for passing the authentication, the user needs to select the "Reset Password" button under the "Settings" section.



Instance Flavor: *uom.general.2c8g*
Availability Zone: *melbourne-qh2-uom*
Image Id: *356ff1ed-5960-4ac2-96a1-0c0198e6a999*

Figure 2: Required information for Setup the MRC Instances

In figure 2, the final step is to record the above common variables and the information of the Key Pairs as host variables to the initialization.yaml file. Notably, the instance flavour should be changed to uom.general 2c9g as it is only the available option on MRC now, indicating each instance will have 2 cores and 9 rams. In addition, the private key for our Key Pair is only automatically downloaded once immediately after the creation as shown in Figure 3, which is saved as key.pem in the keys folder for later convenience. This is a crucial step to do for accessing MRC instances via local machines.



Displaying 1 item

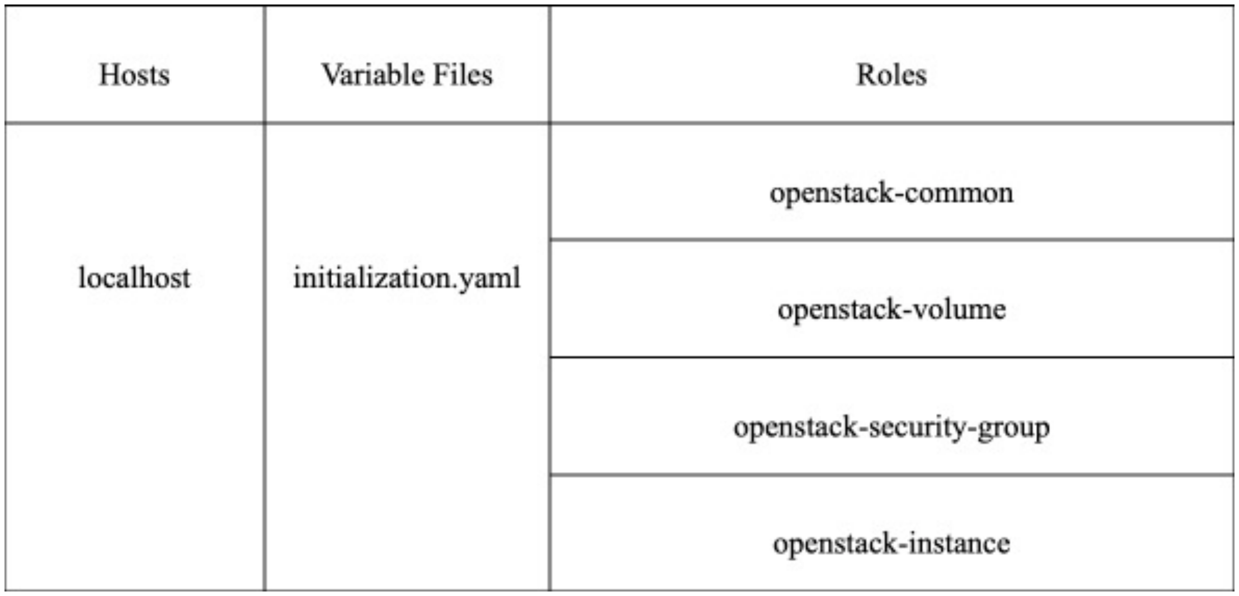
Name	Type
key	ssh

```
Fingerprint
4d:db:12:30:5e:48:db:c0:2c:01:9a:6f:3c:b8:84:83
Public Key
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDdEbddn3OLCHRmNtBx1mZfE1SXVblnjy5By2zitOuxhGTKXa2TMO1+fxv4vitbh5TURsWJzgU8g7oluWJ/HkpSHxFnNCE
oK5CehK7jrOz1kMkNd9QCNjJTTDCVyKwmoQi9JavrtSN4vhZ6b7OwfNP3NLrfsn/5IGh+xip4ZalokvA/SwEJC6YcGld7pr2iut9cUPJTuMbQhrTGPTrvcBnc8qTGzFzvr
t/7yNtuEqIofXMLrlOv/pufNdqHYY/ahL4zvDaYJG1fKy0FUwpfvRkIVcW7rJkARuTnP2pF6f0SgwQjk/q25WXADkyPitE5V Generated-by-Nova
```

Figure 3: Keypair Information

1.2 Initialisation

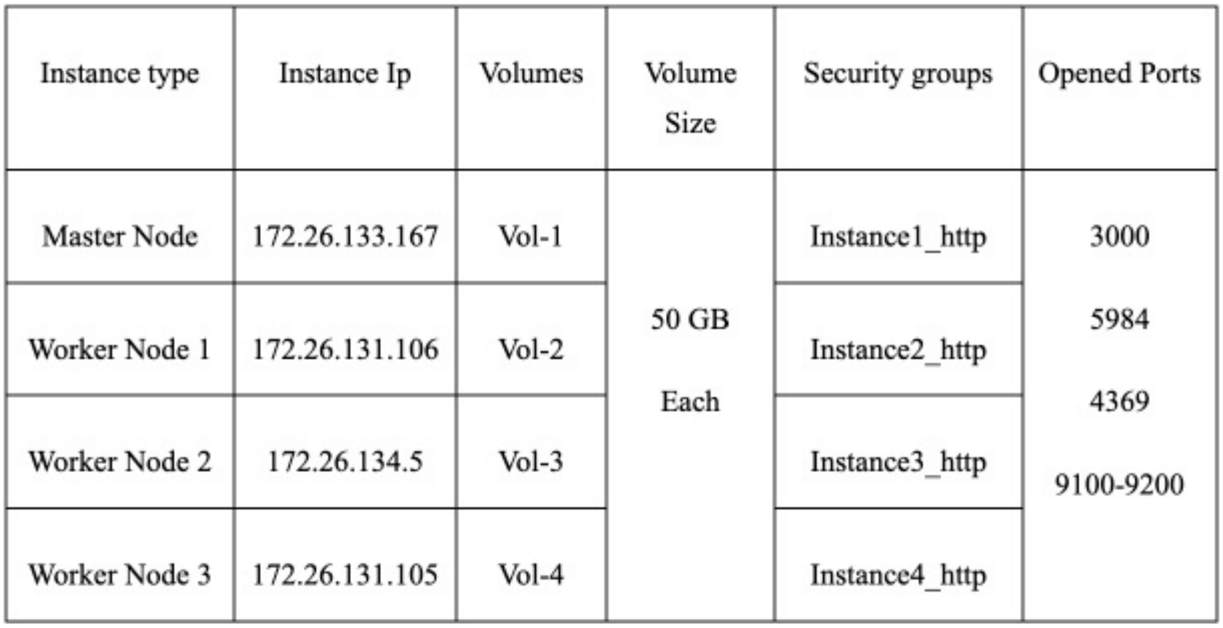
The initialization step is modified from the workshop demo which is to create general settings in MRC, such as volumes, security groups and instances as shown in Table 1 below. All tasks are run by the local host and attached with the initialization.yaml file that contains all required local variables.



Hosts	Variable Files	Roles
localhost	initialization.yaml	openstack-common
		openstack-volume
		openstack-security-group
		openstack-instance

Figure 4: Table 1 - Initialisation Steps

Firstly, the openstack-common task is deployed, which will help the localhost to install all required python packages in case that its operating system is ubuntu. These prerequisites are in preparation for mounting openstack, which contains useful Ansible commands for most of the following tasks.



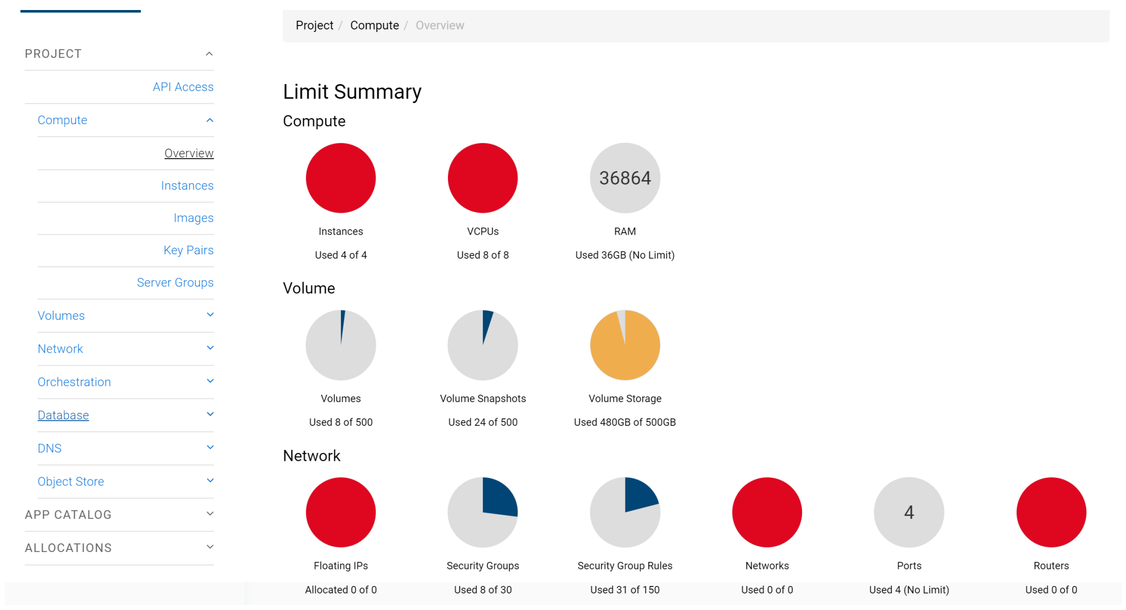
Instance type	Instance Ip	Volumes	Volume Size	Security groups	Opened Ports
Master Node	172.26.133.167	Vol-1	50 GB Each	Instance1_http	3000 5984 4369 9100-9200
Worker Node 1	172.26.131.106	Vol-2		Instance2_http	
Worker Node 2	172.26.134.5	Vol-3		Instance3_http	
Worker Node 3	172.26.131.105	Vol-4		Instance4_http	

Figure 5: Table 2 - Instances Setting in MRC

The above table summarises the generated results after running openstack-volume, openstack-security-group and openstack-instance roles. We start up with establishing four separate volumes that include 50 Gigabytes capacity individually. Each volume will be attached to one instance. Subsequently, 4 security groups are also created and each one opens necessary ingress ports for later usage. Port 3000 is only for master nodes so that we will be able to access our frontend web page via gateway. On the other hand, port 5984, 4369 and 9100-9200 are allocated to all instances, which allows us to access the couchdb system corresponding to the IP of each instance.

The final step of initialization is to create 4 instances in MRC. Most of the tasks are similar to the Ansible demo and all required variable inputs are from the initialization.yaml file under the host_vars folder. However, in order to record all instances' Ips locally and access them via ssh later, host.ini file is generated under the inventory folder which includes an "instances" block that contains Ips of 4 instances, as well as 4 instance blocks that correspond to 4 instance Ips separately. Through this way, the later commands can be chosen to either access all 4 MRC instances via a loop or access only

1 instance individually. Moreover, after creating all instances, we consequently classify instance 1 as our master node, and the rest instances are treated as slave nodes for later backend purpose. As a result, the output of our initialization is represented as follows:



Limit Summary

Compute

Instances	VCPUs	RAM
Used 4 of 4	Used 8 of 8	36864 Used 36GB (No Limit)

Volume

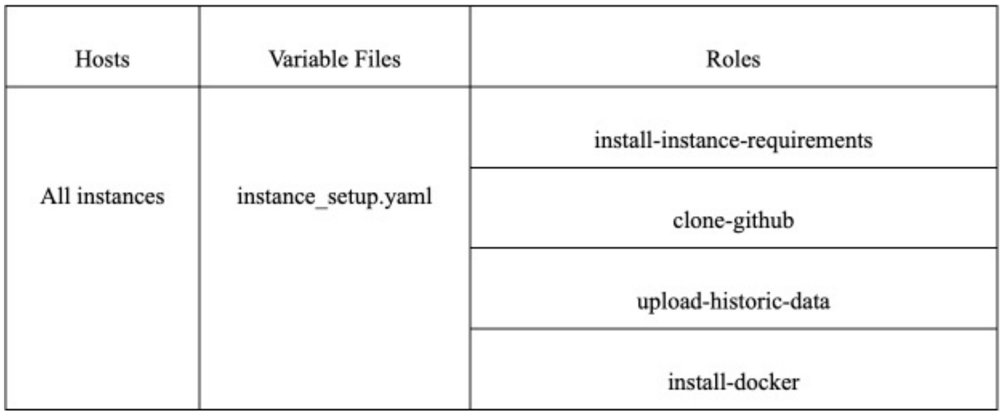
Volumes	Volume Snapshots	Volume Storage
Used 8 of 500	Used 24 of 500	Used 480GB of 500GB

Network

Floating IPs	Security Groups	Security Group Rules	Networks	Ports	Routers
Allocated 0 of 0	Used 8 of 30	Used 31 of 150	Used 0 of 0	4 Used 4 (No Limit)	Used 0 of 0

Figure 6: Allocated Resource in Melbourne Research Cloud

1.3 Instance Setup



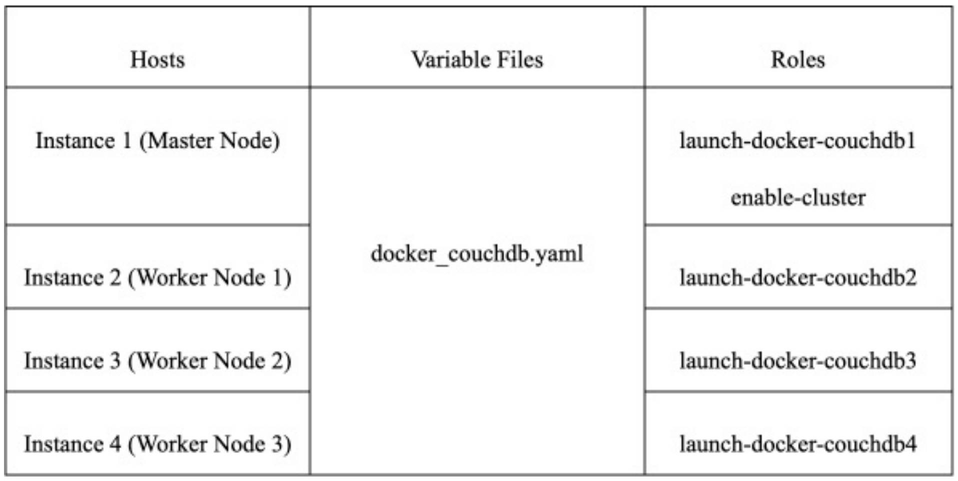
Hosts	Variable Files	Roles
All instances	instance_setup.yaml	install-instance-requirements
		clone-github
		upload-historic-data
		install-docker

Figure 7: Table 3 - Instance Setup Requirement

We divide the instance setup process into 2 parts. According to Table 3 in figure 7, the first section is to install all pre-required dependencies as well as upload compulsory data and files to those 4 MRC instances. According to the table, apart from downloading python packages just like the previous openstack-common step on the local machine, both install-instance-requirements and install-docker tasks help the instances to maintain extra packages for future functionalities, such as “git” for cloning our github repository, “docker” for creating docker-CouchDB systems and “tweepy” for deploying tweet API in attempt to crawl twitter data via backend.

The following clone-github role is not a simple task just like cloning files on local machines. Since our git repository is private, we have to first manually generate an SSH key on github and type “ssh-keygen -t rsa -b 4096 -C your@email.com” on the local machine’s terminal. For convenience, our private key is attached under the keys folder named “id_rsa” and it will be uploaded to the “.ssh” folder with a “ssh_config.j2” file as a template in each instance. Consequently, our repository will be successfully cloned and the corresponding private key will be deleted for security purposes.

We also need to upload the given historic tweet data towards each instance so that the worker nodes will be able to read the json file locally. Since it wastes much time to transmit a 11 GB file on MRC via the network, we use Ansible script to upload the 1.1 GB compressed file and extract it afterwards.

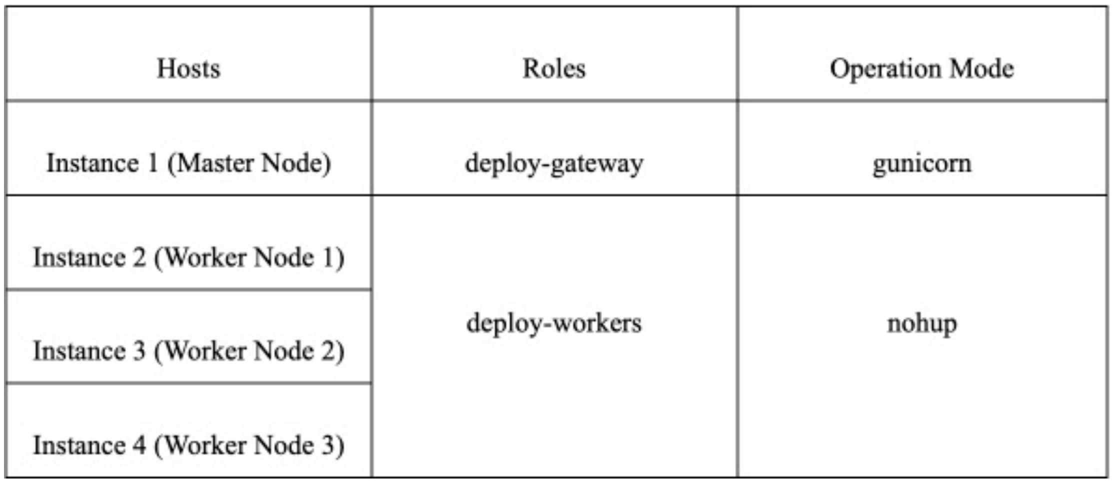


Hosts	Variable Files	Roles
Instance 1 (Master Node)	docker_couchdb.yaml	launch-docker-couchdb1 enable-cluster
Instance 2 (Worker Node 1)		launch-docker-couchdb2
Instance 3 (Worker Node 2)		launch-docker-couchdb3
Instance 4 (Worker Node 3)		launch-docker-couchdb4

Figure 8: Table 4 - Launch Docker CouchDB in MRC Instances

Referring to Table 4 in figure 8, the second section is to launch docker-couchdb via all instances. Along with opening all required ports, we also add user, password, corresponding node name, secret cookies and erl flags to the environment as shown in the demo. After successfully creating 4 couchdb systems, we have to enable clustering via our master node and allow it to connect with other 3 worker nodes. The final relationship is represented as below:

1.4 Deploy Gateway and Workers



Hosts	Roles	Operation Mode
Instance 1 (Master Node)	deploy-gateway	gunicorn
Instance 2 (Worker Node 1)	deploy-workers	nohup
Instance 3 (Worker Node 2)		
Instance 4 (Worker Node 3)		

Figure 9: Table 5 - Deployment of System (Gateway Worker Nodes)

Our final step is to deploy one gateway and three worker nodes automatically via Ansible yaml book as shown in Table 5 in figure 9. Since we can access the frontend web page from the gateway, gunicorn allows us to open the webpage on the backstage so that the webpage can always be accessed. Similarly, the python scripts from workers are operated via nohup, which helps the system to finish the Ansible task immediately and allow background processes rather than running them continuously via terminal. As a result, the gateway will successfully allocate missions to those workers and workers will pass corresponding data to the CouchDB system.

2 Melbourne Research Cloud (MRC)

2.1 MRC Advantages

2.1.1 Availability

The MRC instances are available 24 hours and 7 days per week which allows the users to operate their scripts backstage continuously. They can be set up immediately without any hardware costs and many popular images that include different operating systems can be chosen. Meanwhile, the users are able to have 500 volumes and 500 Gigabytes of ram totally, as well as having the ability to open all required ports. In addition, MRC owns a fancy online interface, helping users to change settings manually as quickly as possible for testing purposes. In order to prevent data loss issues, we can make snapshots of the volumes via the interface as well. Moreover, unlike Spartan, the user would not share its own MRC with members of other teams, so there is no pending time for running jobs on the cloud.

2.1.2 Scalability

Since ansible scripts can be used directly for MRCs, the users are able to deploy tasks and install dependencies for all instances once via automation deployment instead of accessing each instance via ssh separately. As a result, no matter how many instances we have, the resource allocations will always be configured for deemed applications on the cloud remotely, resulting in a "repeatable process".

2.1.3 Security

MRC has a very strict standard to protect user's security. Firstly, in order to pass the cloud authentication, the user has to own a private key from the key pair as well as providing a user unique password every time when making changes to instances. These two steps can only be done by logging the user's unimelb account only on the online MRC webpage. On the other hand, MRC allows the user to open and adjust particular ports under the security group rules rather than providing all the ports straightforwardly, which ensures frontend security safety while accessing the project's webpage.

2.1.4 Concurrency

Since one MRC is shared among all members from one group, other group members will receive an immediate update when accessing instances once one member commits a change. For example, if one user installs python dependency on a particular instance, then another user will be able to run the python script successfully that requires the corresponding dependency without any lock in issues. On another perspective, one user can also access 4 instances at the same time for testing purposes, such as checking any differences between them as well as keeping monitoring all the logs while running the gateway and workers in the backstage together.

2.1.5 Efficiency

One major advantage of MRC is to have multiple instances which would significantly improve the efficiency, especially speeding up the couch db clustering task in this case. When there are several tasks to be allocated, each worker node would be assigned by a particular task and run in parallel rather than doing tasks in a queue. Furthermore, the MRC would have better internet access than localhost, which allows users to download pre-required files and dependencies in a shorter time.

2.2 MRC Disadvantages

2.2.1 Connectivity

Unless running the project near the University, all users have to connect Cisco unimelb vpn in order to access instances via their ips. Subsequently, it takes more time to enter the MRC due to vpn connection and the connectivity would sometimes be unstable which might stop the Ansible task halfway, especially for the overseas users.

2.2.2 Authority

The MRC authority would result in inconvenience when having instant change for testing on a particular MRC, especially printing temporary results for debug purposes. In other words, users cannot change the codes or install dependencies directly via MRC, so that they can only run Ansible scripts or use “sudo nano” to change files remotely.

2.2.3 Hardware Obsolescence

Since MRC is classified as a private cloud, its corresponding resources and hardware capacity are very limited. As it is possible to take countless requests from frontend in our project, the 500 Gigabytes volume might not be capable of containing everything without deleting any less important data.

3 System Architecture

In this section, the architecture of the whole system is illustrated with visualisation of flow charts followed by the interpretation of the benefits brought by this structure and illustration of the fault-tolerant methods deployed in the system.

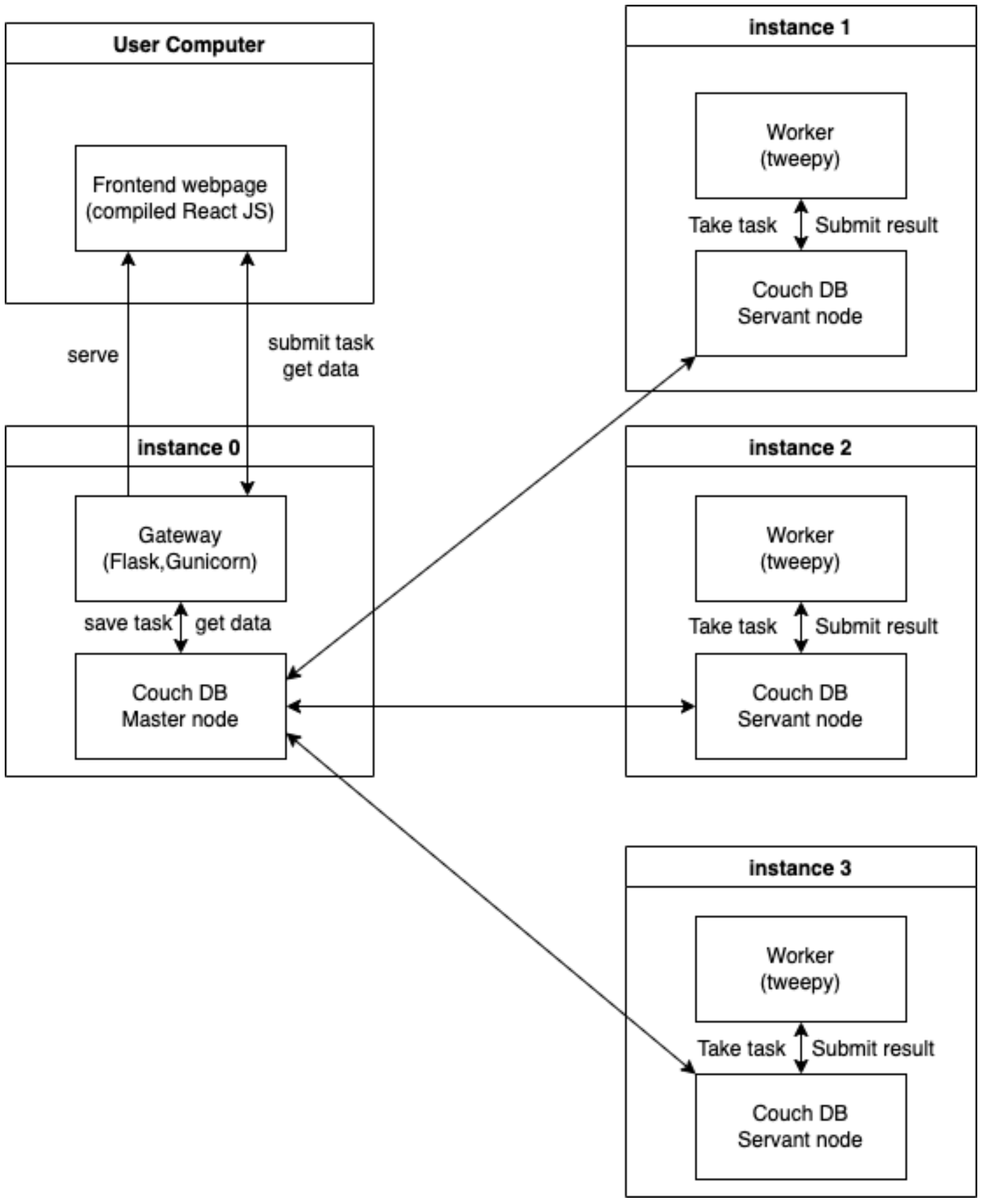


Figure 10: System Architecture Deployment Diagram

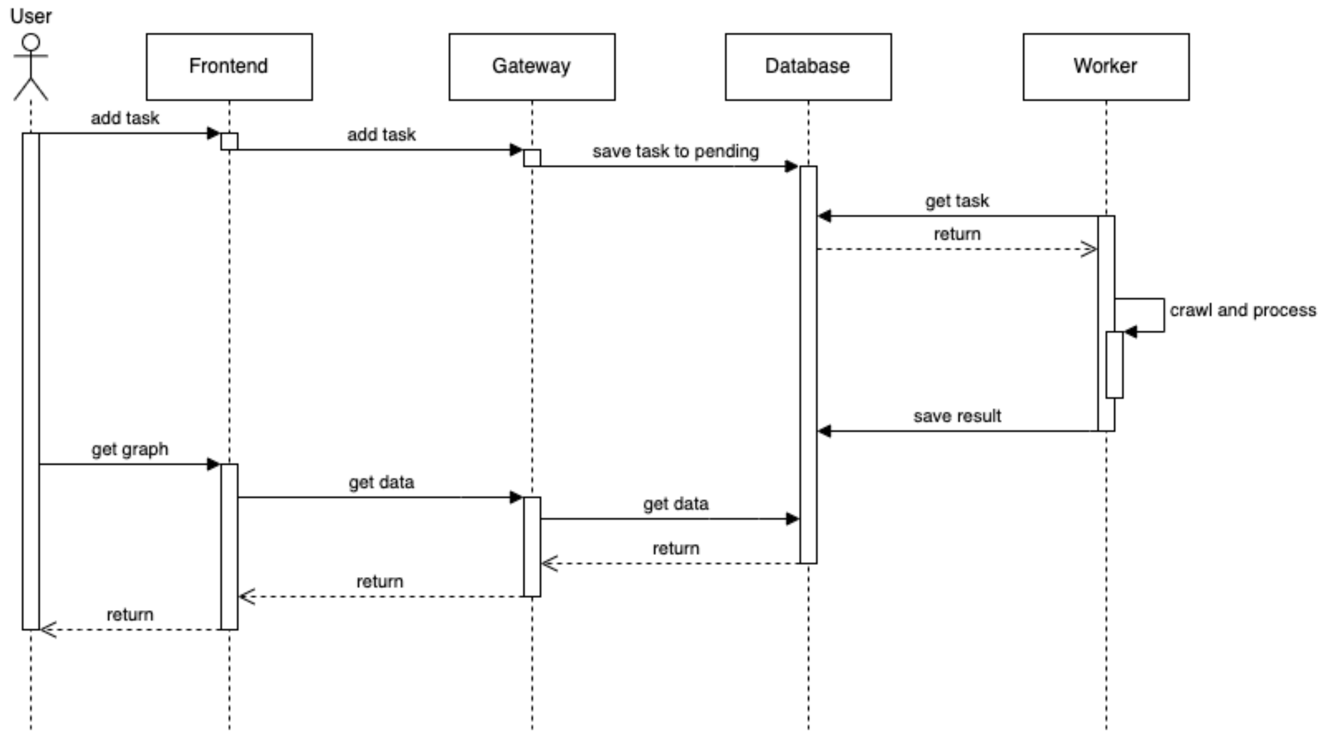


Figure 11: Sequence Diagram

3.1 System Structure

We designed our system based on the standard of cluster and cloud computing system similar to Spartan but specifically for analysis of Twitter and AURIN data. We used a worker pool design to solve the problem in distributed and parallel computing ways to increase efficiency. We modularized our system to not only solve preset scenarios but also function as a general multi-purpose cloud data processing system allowing users to request any additional graph or data such as search on keywords from twitter through API calls.

We used multi-node clustering functionality of CouchDB as the backbone of our system to communicate across instances. Backend programs running on all instances will only need to connect to its local database node. We have one instance saving received tasks from the frontend API to the database called "Gateway". And three instances take tasks from pending documents in the database, crawl from twitter or other data source and process it according to instruction. The summary of task results will be saved in the database for the user to use after the task finished. The benefit of such design is the system is easily scalable, simply connecting more instance running Worker programs to the database without modify existing data and scale can be performed during the system running.

3.2 Fault Tolerance

We have kept fault tolerance in mind while designing the system. All points of failure have been carefully considered and handled to ensure the robustness of the system. All the data and progression information including pending, working and finished tasks list will be saved in the multi-node clustered database with replication enabled. So the crush of any instance or even all the instances will not cause the loss of progress. And DBMS will also solve potential concurrence issues for use.

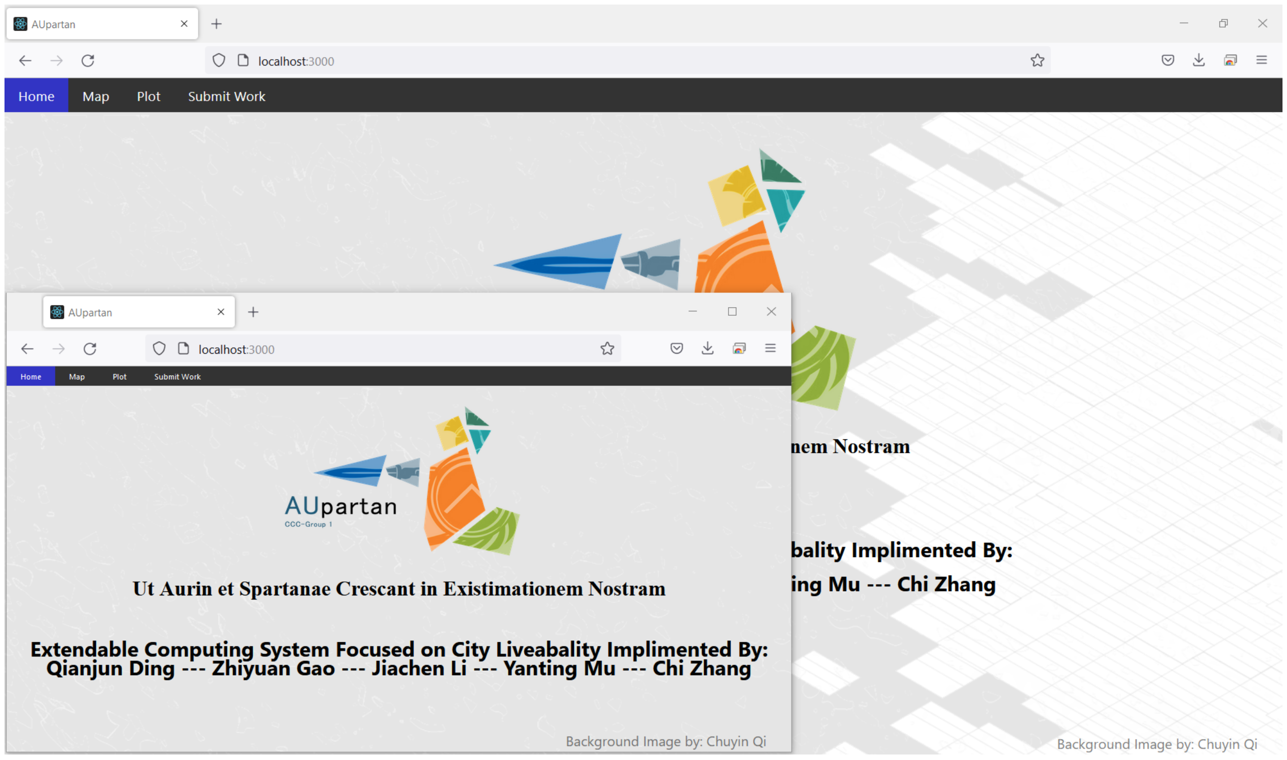
Because the chance of Worker programming failing will inevitably increase as we scale up the system, we have designed the Worker program in a decoupled way to minimise the effect on the rest of the system in the situation of failing. Besides, to ensure the task will not be lost if the Worker working on the task has failed, we have implemented a timeout re-submit system. The Worker will re-accept the task in the working list if the task has runned too long that has reached timeout so insure the task will not be lost. Thanks to the duplicate removal function in the database saving code, even two workers running the same task will not change the result nor break the system.

4 Front-end

The front-end was implemented to fulfil the design patterns in subsections below.

4.1 Responsive Design

A responsive design means the front-end should be able to work on most screens or windows of different (reasonable) sizes, that is to say, all components must react to the given size of the window, and change their width and height accordingly. Figure 12 is a case of our homepage rendered on different windows



Ut Aurin et Spartanae Crescant in Existimationem Nostram

Extendable Computing System Focused on City Liveabality Implimented By:
Qianjun Ding --- Zhiyuan Gao --- Jiachen Li --- Yanting Mu --- Chi Zhang

Figure 12: Front-end Main Page

4.2 Scannable UI

Unlike our backend, human users of the front-end have a very limited speed for processing and remembering incoming information. Therefore, our front end is designed to clearly highlight the components our users need to be focusing on. For example, the detail of each point in the figure 13 will not be displayed, so we can prevent the plot from being too messy, instead, the detailed information will only be shown when the users have their mouse focused on a certain data point.

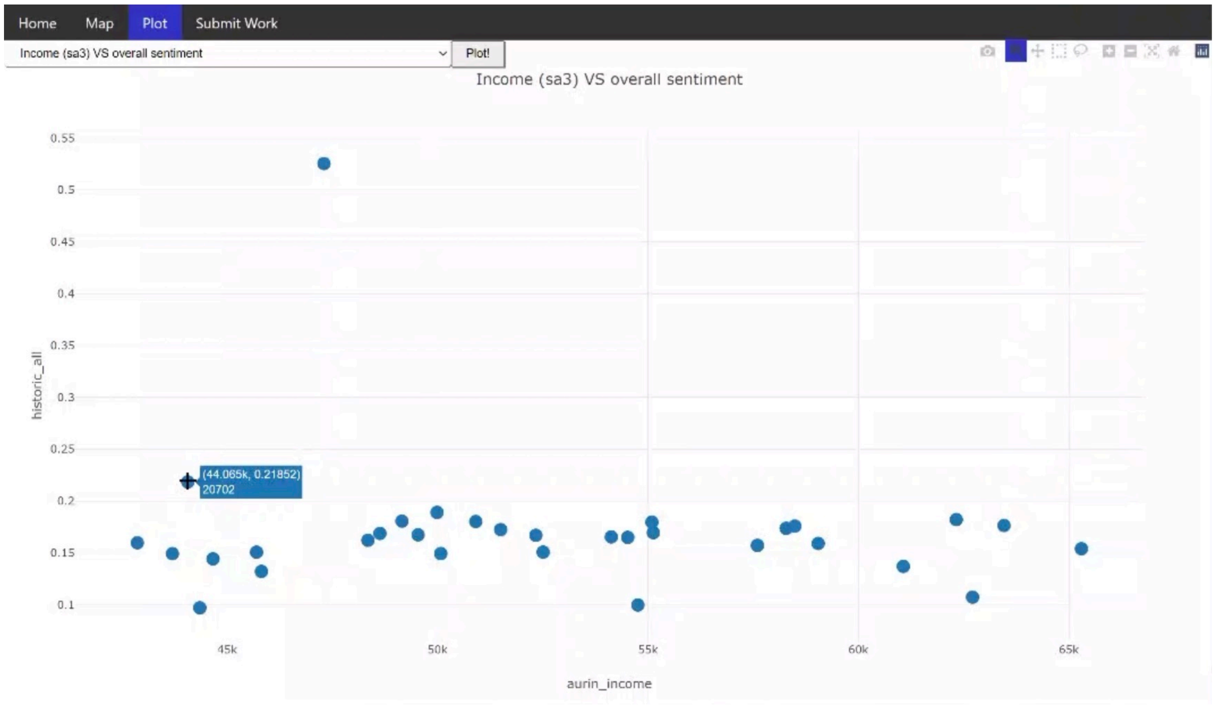
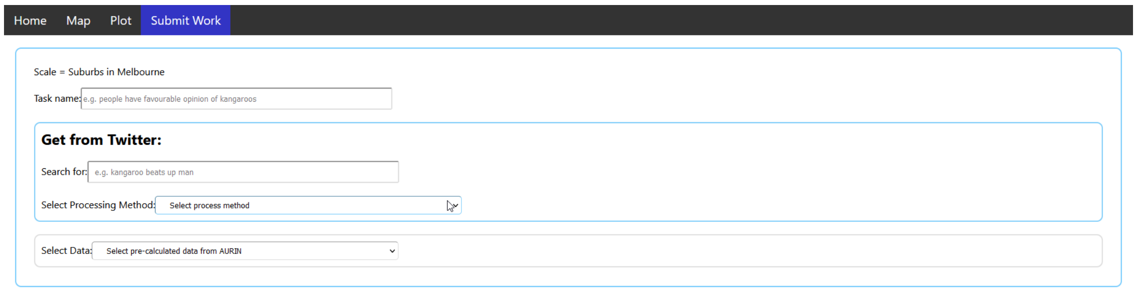


Figure 13: Scannable UI Example

4.3 Clear Call-to-action

Call-to-action means using one or a combination of several design techniques to show users where they should click on, fill in or select items. With such a pattern, our front-end will help the users to avoid as much mis-operations as possible, and here are some examples:



Scale = Suburbs in Melbourne

Task name: e.g. people have favourable opinion of kangaroos

Get from Twitter:

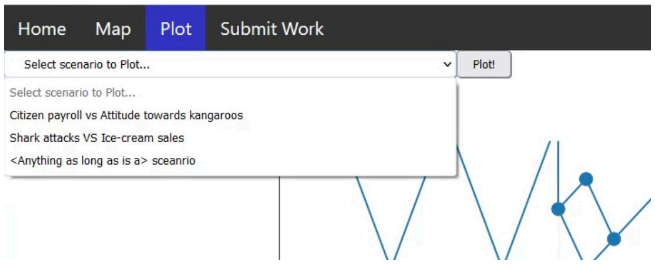
Search for: e.g. kangaroo beats up man

Select Processing Method: Select process method

Select Data: Select pre-calculated data from AURIN

Figure 14: Submit Work Page

Figure 14 shows what users will see in the page “Submit work”. Our front-end will use a bright blue frame to remind the users which layer of config they are editing.



Home
Map
Plot
Submit Work
Select scenario to Plot...
Plot!
Select scenario to Plot...
Citizen payroll vs Attitude towards kangaroos
Shark attacks VS Ice-cream sales
<Anything as long as is a> sceanrio

Figure 15: Plot Visualisation Page

Figure 15 is captured when we were testing the selection for a scenario to plot. The selection bar and plot button is clear, eye-catching and easy to follow.

5 Data and CouchDB

5.1 Data Source

5.1.1 Aurin Data

In order to have a direct standard for quantifying the livable statistics of people living in Melbourne, data from Aurin is applied. In this project, we have collected the average income data within Melbourne in each SA3 region and in each main city (Melbourne, Sydney, Brisbane, Adelaide, Darwin, Perth, Hobart and Canberra) for comparison with the processed harvested data which are considered as relating to people's wellbeing index. Besides, data of immigration rate in each city is taken into consideration as an representation of cities' attractiveness to people, which also indicates the livability of the city in one aspect. By using the Aurin Portal to select the desired data by plotting the preferred range from the map, dataset are then downloaded as a JSON or CSV file and stored in the Data backend data section. After preprocessing the raw data, the Aurin data are then uploaded to the CouchDB as documents and stored into two databases with respect to the city level and SA3 level for further usage.

5.1.2 Historical Twitter Data

Due to the new privacy policy announced by Twitter Developer, geo-location of each tweet has no longer been accessible through the 'tweepy' (Twitter API V2) except the enterprise premium version. In this case, to enrich the based data for further analysis, historic data of tweets within Melbourn in each SA3 region is applied. Since the scenario in revealing the wellbeing index of each user in each region in Melbourne relies on its geo-location for grouping data and sentimental analysis from NLP method based on "English" contents, preprocessing of the historic data is required. By removing all tweets without coordinates' information and written in other languages despite of 'en' (ISO639-1 Standard), further limitations are considered. Since the retweet could represent users' opinion but not the wellbeing index, retweets from the historic dataset are removed. Besides, as public accounts do not reveal an individual's opinion toward a specific topic, we pre-set a upper limit of each user's follower amount of 3000 followers to exclude the effects brought by the Official Public Account.

5.1.3 Search Twitter Data

With the Twitter API Version 2 search function, we are allowed to collect the latest 7-day tweets related to the selected keyword(s) and hashtag(s). All team members have applied the tokens and upgraded to ‘Elevated’ level which enables us to harvest two million tweets in each month from each account. By setting the keyword(s) including “Election”, “Crime”, we could collect the recent tweets which contain these keywords and store the related information into CouchDB after data preprocessing. Notice that in this project, both retweets and tweets from users that have more than 3000 followers would also be collected since the related tweets’ count amount in this scenario are significant for representing the livable of each city while sentimental statistics won’t be diluted due to the repeated contents.

5.2 Data Preprocessing

In this project, we assume that twitter user’s is a general representation of residents in each region / city. Using the tweet_id from each tweet from the historic dataset or search API from tweepy, we obtain the tweet’s content. Before applying the sentimental analysis model, we applied the function to convert the tweet’s content to purely text since emojis, which cannot be analysed by the NLP model, are widely used. Besides, regular expression is applied to remove the url link attached to each tweet text to further help improve the sentimental quantifying. Then, the sentimental analysis of each user’s tweet’s text with the NLP model is applied and calculates the emotional statistic (constructed by three levels including ‘positive’, ‘negative’, ‘compound’ values) of each tweet. For illustrating the well-being statistic from each tweet (user), the ‘compound’ value, which is calculated based on the ‘positive’ and ‘negative’, will be applied as a summarised statistic for simplifying the illustration of target scenario(s).

5.3 CouchDB Storage

5.3.1 CouchDB Introduction

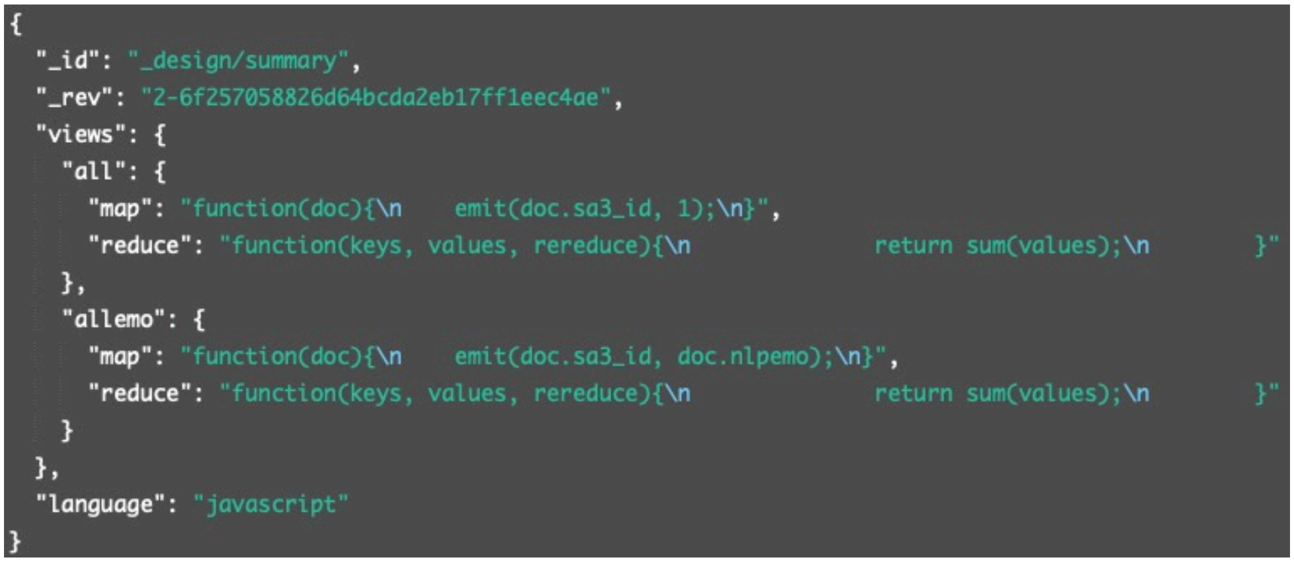
In this project, we applied CouchDB for collecting data for scenario(s) analysis. With its convenience to set up a cluster and the in-build MapReduce queries algorithm, CouchDB performs well in processing this project. CouchDB is firstly applied for storing both raw data collected from Aurin or Twitter dataset and preprocessed data required by the target user (clients) for presenting the selected scenario(s). Besides, with the in-build MapReduce queries algorithm from CouchDB, the efficiency of the extracting data for further analysis has been improved.

5.3.2 Cluster CouchDB

A cluster of CouchDB is set in this project on Melbourne Research Cloud’s with its nodes set across the four provided instances. With the cluster architecture of CouchDB, the replication enables the database to be fault-tolerant by storing the same document onto different nodes which reduces the probability of losing data (system failure) as long as one node remains working and supports the queries from the users. Besides, with the cluster CouchDB, the efficiency in processing MapReduce is improved. Having distributed nodes in each instance provides the platform for distributing the raw data from the select database into different machines and summarise them after the results are obtained from each machine rather than processing data within a single node.

5.3.3 CouchDB MapReduce

The in-build MapReduce query logic in CouchDB is applied for extracting data from raw databases and calculating the summarised statistics in this project. By first uploading the design documents to the raw data database, we then have the logic in extracting the data following the conditions and rules in processing and grouping the target data. Present in figure 16, CouchDB will first group the data from the target database based on the selected key (e.g. sa3_id), then process the logic of summing up the required value from each document (e.g. the nlpemo value from all grouped documents) and finally return the result which is similar to the python dictionary format for further processing.



```
{
  "_id": "_design/summary",
  "_rev": "2-6f257058826d64bcda2eb17ff1eec4ae",
  "views": {
    "all": {
      "map": "function(doc){\n    emit(doc.sa3_id, 1);\n}",
      "reduce": "function(keys, values, rereduce){\n            return sum(values);\n        }"
    },
    "allemo": {
      "map": "function(doc){\n    emit(doc.sa3_id, doc.nlpemo);\n}",
      "reduce": "function(keys, values, rereduce){\n            return sum(values);\n        }"
    }
  },
  "language": "javascript"
}
```

Figure 16: Design Document for Processing MapReduce in CouchDB

5.3.4 Data Anti-Duplication

Since the system is designed for allowing front-end users to insert the desired research perspective with the preferred harvester (Describe in Section 4) and keyword(s), duplicate tweets in the same topic will be collected. In order to reduce the redundancy brought by the duplicate tweets and avoid the unnecessary bias created by the duplicate data, code for avoiding storing the same data is designed based on the in-build CouchDB documents id searching logic. In this project, we applied SA3 id as keys in databases related to Aurin and tweet's id as key in databases related to historical data and tweepy search data. By checking the existence of the designed record's id in the target database, we then decide whether the new data should be stored or not.

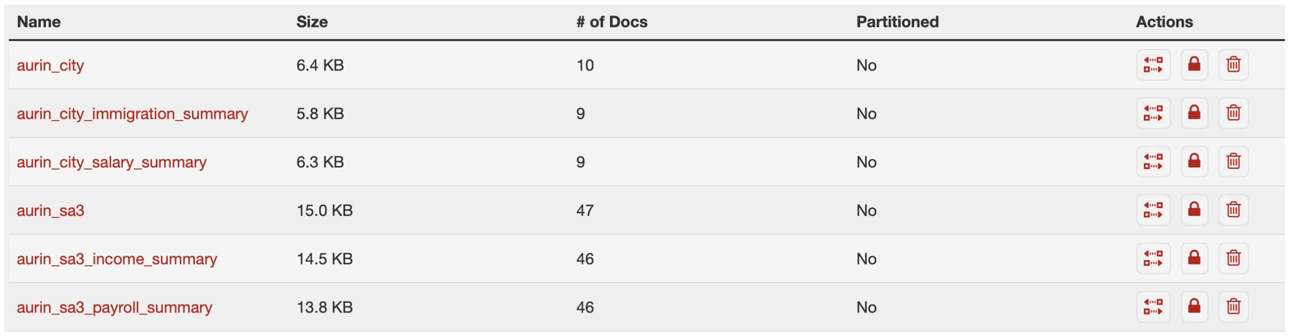
5.3.5 Summarised Data's Retrieval

Once the harvester(s) has finished the task(s), the summarised database of this requirement is prepared. Following the new task sent from the front-end (UI), the Gateway in our system will then call the function to find the target summarised database and extract the data (counts / sentimental statistics) with MapReduce algorithm for creating visualisation on the user interface.

6 Harvester

6.1 Aurin Harvester in both SA3 & City Scales (aurin.py)

The Aurin harvester is designed for collecting the target data with the given requirement of region's level (SA3 / City) and data keyword. Notice that in this project, we only download the datafiles including median value of personal income and personal weekly payroll for the selected SA3 regions in Melbourne, and the mean value of personal salary and the latest immigration rate in each major city in Australia from Aurin Portal. We first store these data into the raw database and calling the designed function to extract the desired data with input parameters (e.g. level = 'city', keyword = 'salary') enables us to collect personal salary in each city and store them into the summarised database. Presented in figure 17, the Aurin data are pre-stored into aurin_city and aurin_sa3 database and the summary database related to specific keywords after the query from user(s).

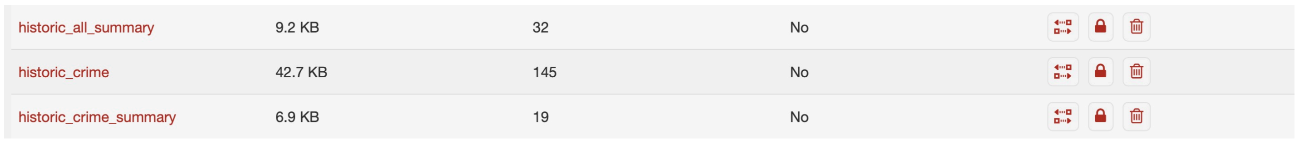


Name	Size	# of Docs	Partitioned	Actions
aurin_city	6.4 KB	10	No	
aurin_city_immigration_summary	5.8 KB	9	No	
aurin_city_salary_summary	6.3 KB	9	No	
aurin_sa3	15.0 KB	47	No	
aurin_sa3_income_summary	14.5 KB	46	No	
aurin_sa3_payroll_summary	13.8 KB	46	No	

Figure 17: Aurin Databases

6.2 Historic Harvester in SA3 Scale (historic.py)

Similar to the preserving logic of Aurin data, the historic data is selected and preprocessed from the provided historic json. Tweets which satisfy the requirement, are stored into the raw database in a total amount of 348302 documents for further analysing based on the inserted keyword from user(s). By selecting the harvest method in the UI and typing in the desired topic, the task is then sent to the worker node which could activate the historic data harvester. Presented in figure 18 below, when the front-end user wants to investigate the 'crime' rate information from the historic dataset, the Worker node will activate the harvester and use the MapReduce to extract documents from the 'historic_raw' database before aggregating the tweet's count and sentimental value of each SA3. With the summarised data obtained above, the 'historic_crime_summary' database is created for user's usage.

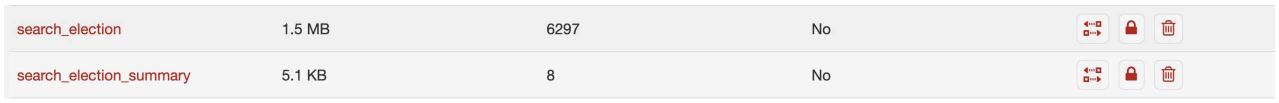


historic_all_summary	9.2 KB	32	No
historic_crime	42.7 KB	145	No
historic_crime_summary	6.9 KB	19	No

Figure 18: Raw database, Target Database and Target Summary Database

6.3 Search Harvester in City Scale (search.py)

Different from the Aurin and Historic harvesters, the Search harvester does not rely on the prepared data but only requires the input keywords from the user. Due to the policy of twitter developers, we are only allowed to crawl the tweets from the past 7 days, which has limited the available data amount which is planned to be used for data analysis. Presented in the Figure below, after the user has sent a task via the UI by specifying the keyword they want to investigate, the harvester will then activate the 'search_recent_tweets' method to collect all related tweets. After storing all these related tweets into CouchDB documents, summarised data is automatically computed with MapReduce to create the summarised database. Shown in figure 19, after activating the Search harvester with keyword = 'Election', it will collect the latest 7 days' related data from each major city and generate the summarised database 'search_election_summary' for further usage.



search_election	1.5 MB	6297	No
search_election_summary	5.1 KB	8	No

Figure 19: Target and Summary Database Generated by 'Search' Harvester with Keyword 'Election'

7 Functionality

In this section, the functionality of the system will be illustrated with respect to the scenarios which are designed for investigating the livability of regions in Melbourne and comparison between Melbourne and other cities in Australia. Moreover, the system is not limited in the preset tasks but allows users to process their analysis with the desired topics based on different harvesters mentioned in Section 6.

7.1 Functionality Process

(i) Execute the dynamic automation deployment of the system onto Melbourne Research Cloud
(ii) Pre-record the required dataset (Aurin and Historic Data) into CouchDB once deployment is done
(iii) Load in required scenarios' tasks into Gateway node and Worker nodes get work(s) for execution
(iv) Collect data with desired keyword and in-build MapReduce query
(v) Applied sentimental analysis on the collected raw data if necessary (Aurin does not require NLP)
(vi) Collect all preprocessed data into summarised database for front-end visualisation usage
(vii) Front-end send the query to Gateway and retrieve the desired summarised data

7.2 Functionality Example

(i) Execute the dynamic automation deployment of the system onto Melbourne Research Cloud
(ii) Record the historic dataset into CouchDB once the deployment of system is done
(iii) The Gateway receive the desired task 'historic Election' for collected related tweets for analysis
(iv) Worker 1 request the task from Gateway task-list and start harvesting the 'Election' historic data
(v) After collecting tweets, apply sentimental analysis model onto it and obtains emotional statistic
(vi) Collect preprocessed tweets' information and summarise it into database with MapReduce
(vii) The front-end user request to plot visualisation by querying data from database through Gateway

7.3 Generic Usage

Notice that this system is not limited in operating the preset tasks / scenarios but allows user(s) to put in their desired topic and harvest the related information via the preferred harvester. Once the required topic is sent to Gateway node, it will be placed into the task queue and will be automatically got by the Worker node for harvesting , preprocessing and summarising for visualisation usage which is further requested by the front-end user(s).

8 Designed Scenarios

8.1 Scenario Regions

In this project, two scales of regions in Australia related to Melbourne are studied for showing the analytic information of liveability standard. Presented in figure 20, the selected regions are the investigated SA3 regions bound by the selection criteria box from Aurin.

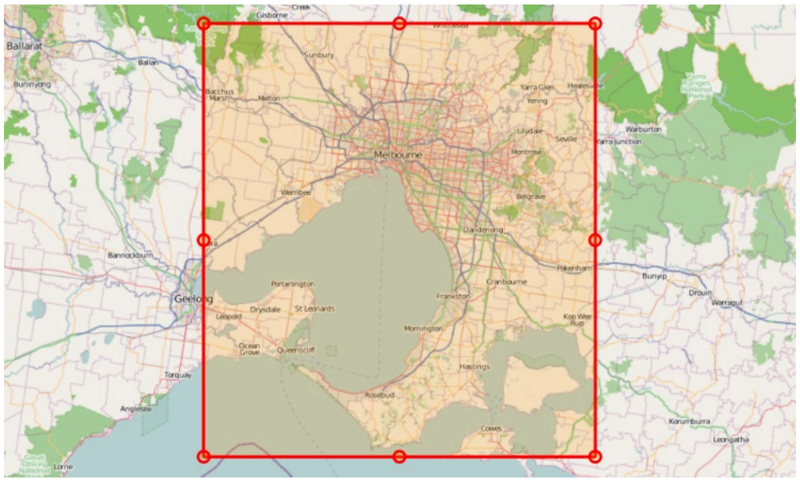


Figure 20: Selected SA3 Regions from Aurin Portal Bounding Box

Besides, comparisons between cities are also considered for analysing the livability of Melbourne. Presented in figure 21, the capital city from each state and federal government are included for investigation and once the valid data is collected.

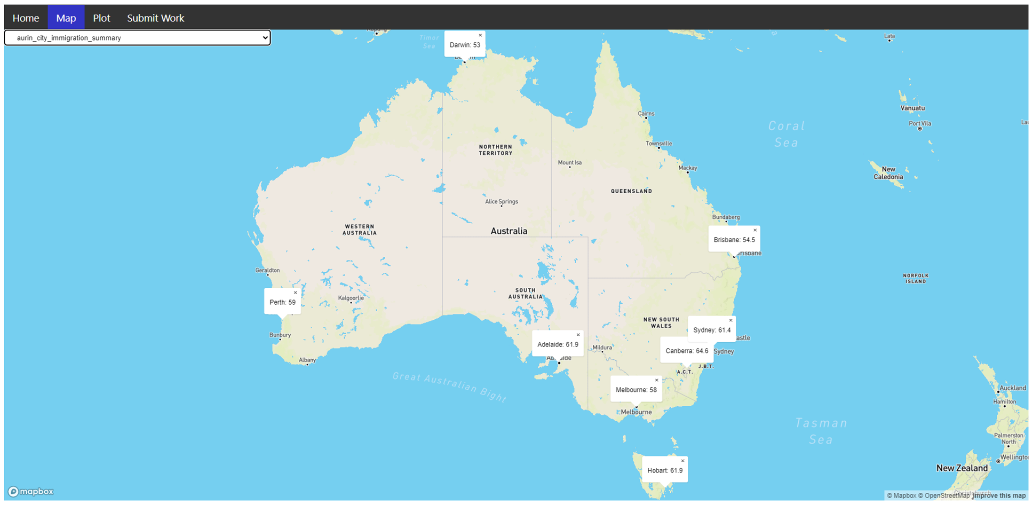


Figure 21: Eight Major Cities from Australia with Immigration Rate

8.2 Scenario

8.2.1 Resident's Average Happiness Index in each SA3 Regions vs. Average Income

In this section, we treat the average sentimental statistic of residence within each SA3 region to represent the general wellbeing level for illustrating the liveability of each region in Melbourne. Supported by Cooper et al., the difference in income level is significant in affecting the individual wellbeing index (life satisfaction) especially for the lowest twenty percent of income distribution compared to the overall group (2013). By plotting the scatter plot of the sentimental result (a compound statistic for representing the positive or negative emotion within range of [-1, 1]) against the average income value collect from Aurin Portal, we expect to visualised an approximate correlation and could further obtain that which zone(s) has a higher index of liveability in Melbourne.

However, presented in the plot below, a uniform distribution is observed between the income and sentimental result from all SA3 regions which have valid data in Melbourne. The outlier from the plot is further proven that only one tweet is collect in that region with SA3_id = 20303 indicating that large bias is involved in generating this point and should not be considered in further analysis. In this case, two potential conclusions are made including: the income value cannot reveal the truth stage of resident's overall life satisfaction level, or the differences of income and overall sentimental result between each SA3 region in Melbourne are not significant indicating that no significant difference is observed from the liveability in each sub-area in Melbourne.

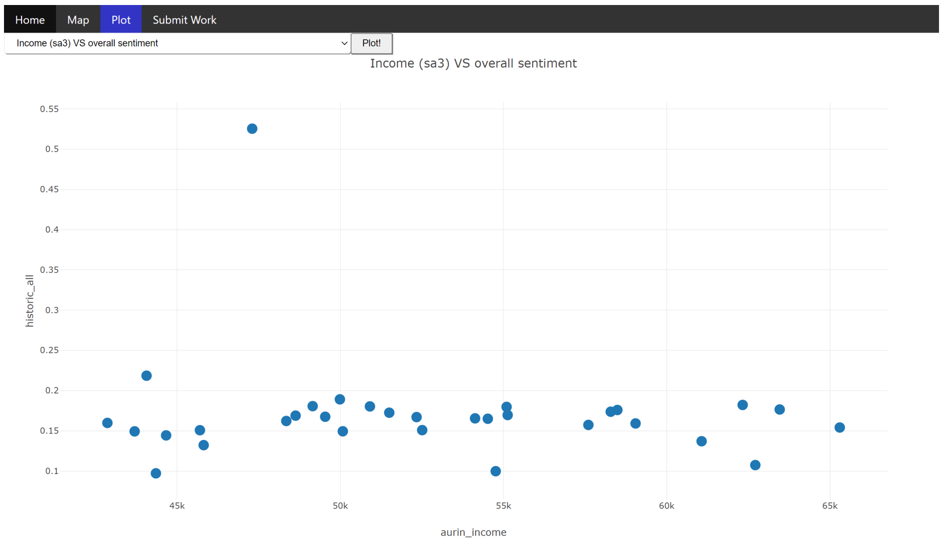


Figure 22: Overall Sentimental Score vs. Average Income in SA3

8.2.2 User's Recent Tweet Count of Mentioning Crime vs. Average Salary

Despite the population difference between each city, the reported crime count from tweets is an absolute number that alerts the issue of security in all cities. By treating the crime count as a negative effect and the average salary as a positive effect towards the livability statistic of each city, we could then create the visualisation to obtain whether these two effects are balanced. If an outlier is observed from the plot in the positive y-axis (crime count), we then conclude that the city is relatively not liveable in this standard.

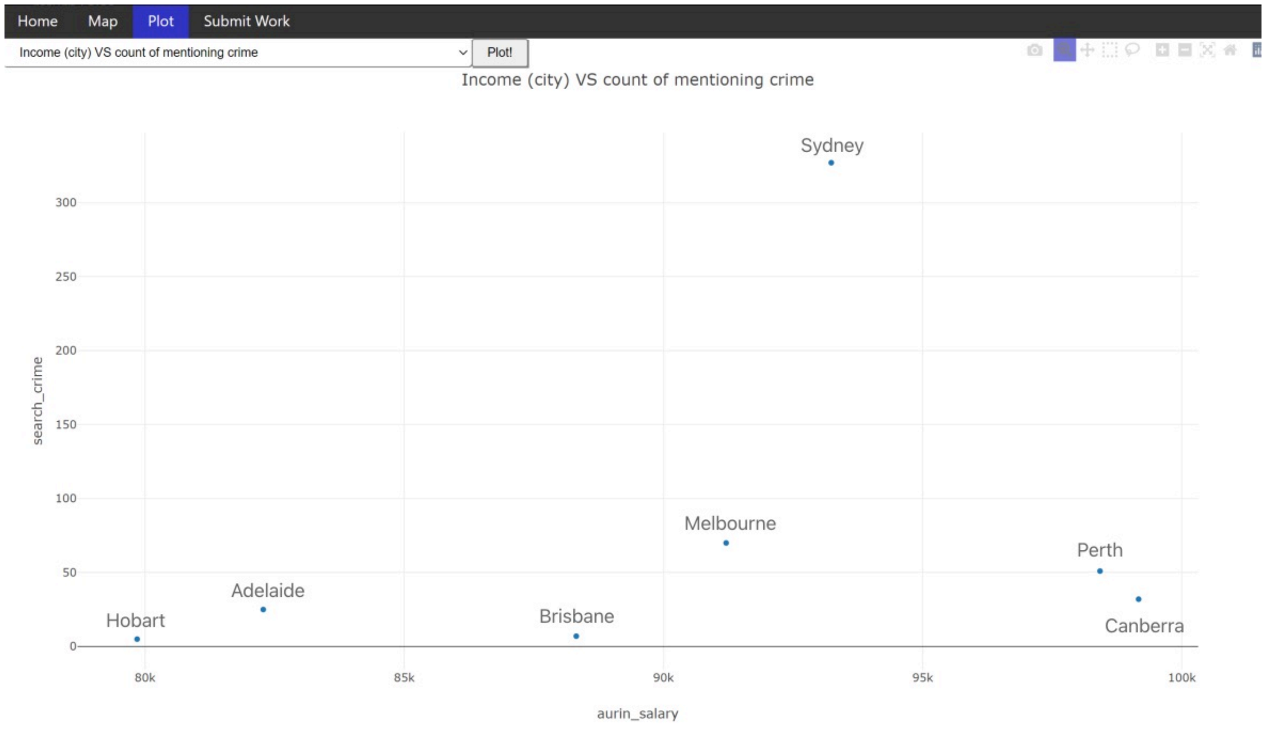


Figure 23: Count of Mentioning Crime vs. Average Salary in City

Presented by figure 23 from the plot in the user interface, all six major cities (Hobart, Adelaide, Brisbane, Melbourne, Perth, and Canberra) have approximately lied along with a line, showing that the trade-off between the positive and the negative effect is considerable. However, Sydney in this plot shows a significantly different characteristic and is appropriate to treat it as an outlier with too many reported crimes on twitter. In this case, by comparing Melbourne with Sydney in this standard, we could conclude that Melbourne is a liveable city while we have not enough evidence to further prove that Melbourne is better than the other five cities with valid data from Twitter.

8.2.3 Resident's Attention and Attitude towards Election

Since it is currently the election period of the federal government, the attention and attitudes from residents are worth investigating. Besides, presented by Randall, participation rates in politics (Election / Voting) are positively correlated with the family income (2019). By collecting the 'Election' keyword related to last-7-day tweets in each city and summing up the total number of tweets to represent the participation enthusiasm subjectively from individual users or passively affected by the official account broadcast in Twitter.

Although presented in figure 24 that no linear correlation is found between the count of mentioning 'Election' in the past-7-day tweets harvested by the 'search' method, the count of 'Election' in Melbourne is the highest indicating that the highest enthusiasm of participating into the political life. In this case, we preliminary reach the conclusion that people living in Melbourne shows a relatively high participation frequency which indirectly the reveal the liveability index of Melbourne compared to other cities with valid data in Australia.

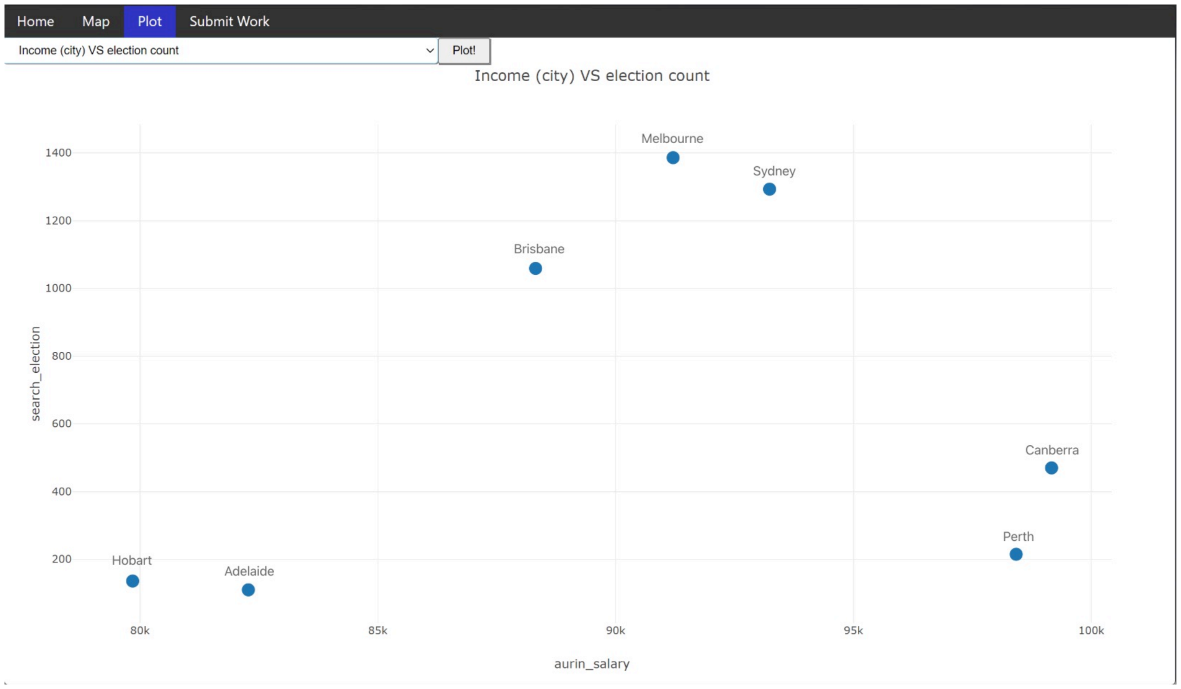


Figure 24: Count of Mentioning Election vs. Average Salary in City

Besides, the sentimental analysis of all tweets related to Election in each city is calculated as a side-product for further digging out information from all residents. Presented in the scatter plot below (figure 25), the fact that only Melbourne lies in the negative y-axis of attitude towards election in this year. Further investigation on whether the spread of tweets with negative emotion to individuals would affect people's happiness / wellbeing index can be conducted which might also be helpful in researching the liveability stage of a city in future study.

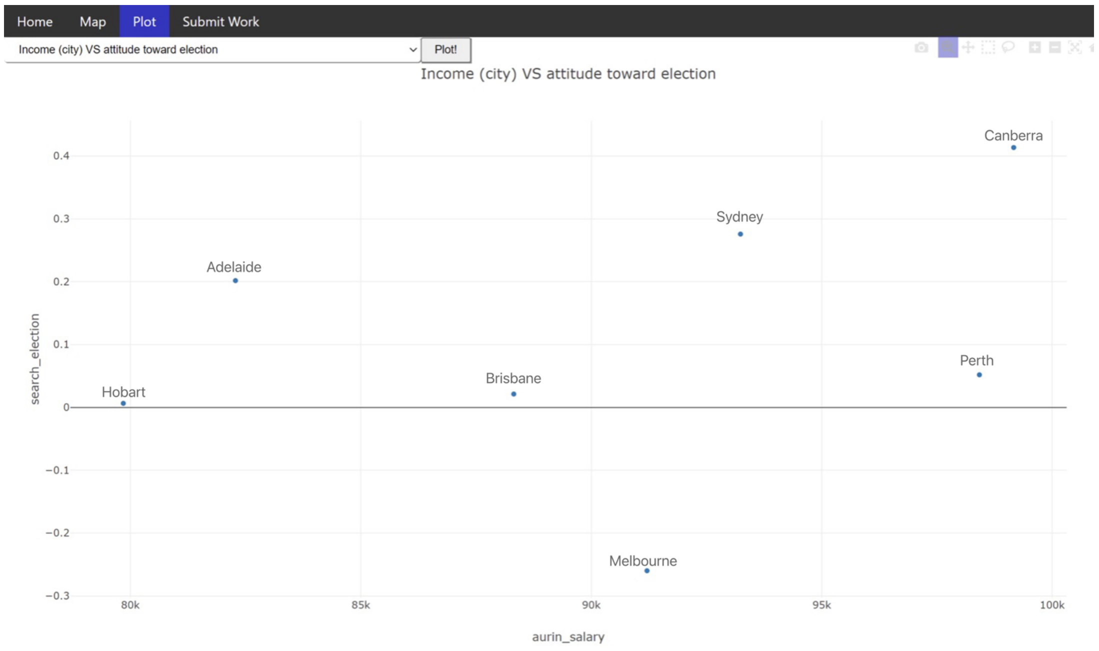


Figure 25: Sentimental Value of Election vs. Average Salary in City

9 Limitation

Due to the limited time and resources we have, there are still a lot of limitations that can somehow be enhanced.

9.1 Twitter API Limitation

As twitter changes their user policy now, our ability to access data is largely restricted..Without the ability to easily acquire some data, like geolocation, there are many difficulties for designing any location-based analysis. Although for now this problem is temporarily handled by using keywords matching, the accuracy of geolocation is very poor, which makes a detailed analysis that needs SA3 area data or more not possible to be done. This can be solved by using an academic or even enterprise account. Another way is to design a complex algorithm/method based on social engineering techniques.

9.2 CouchDB Limitation

As a major ability and advantages of CouchDB, the map-reduce ability within the view function helps us a lot as mentioned in the database part above. However, the limitation of it also reduces the utility of our usage of it. Although within the map function, some filter or search query can be applied to it, there is a probability that the system will return a timeout error. We are not sure what causes this error and never stably trigger it during the test. It might be solved by applying some more efficient query method or even using a different database system. As a result, our keywords search function has some limitations and cannot reach its full potential. It can still work on simple search but might have problems on complex queries. Thanks to the error handling system discussed above, this error should not threaten the overall performance of the entire system.

10 Appendix

10.1 GitHub & GitLab Links

We use a Github repository as a version-control system for storing, sharing and maintaining the code for all teammates to ensure the no loss of code with backup versions. Notice that the current version of our GitHub repository only contains the main branch as we have concluded the code and merged the branches for finalising. Due to the privacy issue, we make our GitHub repository in private mode. If you cannot open the link below, please contact jiachen3@student.unimelb.edu.au for adding you into the contributor list. We also create a GitLab repository which allows user from Unimelb to access it.
The access link to GitHub is: `https://github.com/Valkyriean/CCCA2.git`
The access link to GitLab is: `https://gitlab.unimelb.edu.au/jiachen3/ccca2`

10.2 Front-end Access Link

The access link to the Front-end is: `http://172.26.133.167:3000/`

10.3 YouTube Link

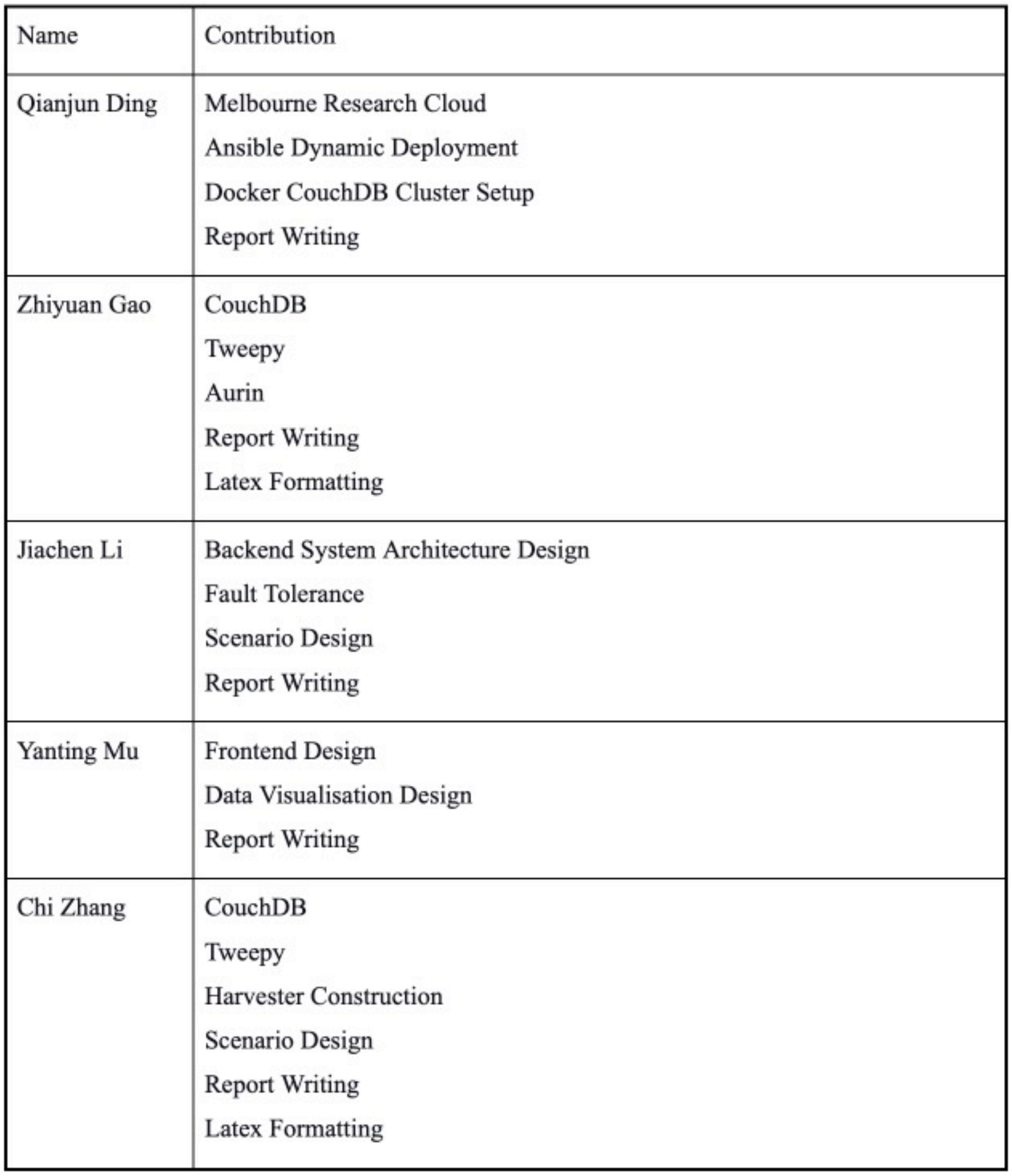
We record a video demonstration of deploying the code onto MRC with Ansible and processing the system to show the designed scenarios and extra functionality in this project.
The access link to Ansible Demo is: `https://youtu.be/rVhQiquWo1g`
The access link to Live Demo is: `https://youtu.be/llylVGgSSEo`
The access link to System Demo is: `https://youtu.be/AtHIEw5ZZ0s`

10.4 Target Regions

In this project, we investigate the information in cities including: Canberra, Sydney, Melbourne, Brisbane, Perth, Adelaide, Hobart. Notice that 'Darwin' is not included in this project's investigation since another city, which has the same name, has provided significant effect on collecting the tweet from twitter API and lead to bias in analysing the data.

In this project, we also investigate the information in SA3(s) within Melbourne City and Greater Melbourne. Notice that some regions might not contain valid information and will be discarded before delivering the visualisation at the user interface.

11 Individual Contribution



Name	Contribution
Qianjun Ding	Melbourne Research Cloud
	Ansible Dynamic Deployment
	Docker CouchDB Cluster Setup
	Report Writing
Zhiyuan Gao	CouchDB
	Tweepy
	Aurin
	Report Writing
	Latex Formatting
Jiachen Li	Backend System Architecture Design
	Fault Tolerance
	Scenario Design
	Report Writing
Yanting Mu	Frontend Design
	Data Visualisation Design
	Report Writing
Chi Zhang	CouchDB
	Tweepy
	Harvester Construction
	Scenario Design
	Report Writing
	Latex Formatting

Figure 26: Individual Contribution

References

[1] Akee, R. (2019). Voting and Income. Econofact, February, 7.

[2] Cooper, D., McCausland, W. D., Theodossiou, I. (2013). Income inequality and wellbeing: The plight of the poor and the curse of permanent inequality. Journal of economic issues, 47(4), 939-958.