
# The University of Melbourne

## COMP90024 Cluster and Cloud Computing

### Greater Melbourne Gluttony Analytics Report

Team 42


| | |
|---|---|
| Minjian Chen | 813534 |
| Shijie Liu | 813277 |
| Weizhi Xu | 752454 |
| Wenqing Xue | 813044 |
| Zijun Chen | 813190 |


May 14, 2019

# Introduction

With the development of internet, social media plays an necessary role in our lives. For example, in Twitter, a variety of tweets are generated every day from users to express their thoughts and emotions. The focus of this project is to explore the Seven Deadly Sins through social media Twitter data analytics. Based on this large amount of data increased continuously, it is essential to build an software architecture that supports handling big data. We choose the Microservice architectural style and built a distributed data harvest import process and visualization system. Finally, we combined the processed data with statistical data from Australian Urban Research Infrastructure Network (AURIN) platform, in order to discover certain scenarios which tell interesting stories related to Gluttony, as one of seven deadly sins.

# Software Implementation

## Software Architecture

On the whole, we use the Microservice architectural style to structuring this project. Our whole system is combined by a collection of services: nginx gateway, helper backend, couchdb, improter, harvester and preprocessor. Each service can be considered as an individual software and all communications between different services are done by HTTP request (RESTful API) through the nginx gateway. We use docker to host all the services and most of tasks are distributed using the Publish–Subscribe pattern. Therefore, the overall system is very easy to scale out. For most of services, the only step is to start a new docker container for the given instance on an arbitrary machine without modifying other existing one. Other services are also scalable but require some configurations. Furthermore, each service can be taken down and we can deploy a new rollout individually without affecting the overall availability.

We currently have 4 VMs that are running Nginx Gateway with Frontend x2, Helper Backend x2, CouchDB x4, Importer x4, Harvester x5 and Preprocessor x4. Those are distributed as follows:

| | Nginx | Backend | CouchDB | Importer | Harvester | Preprocessor |
|---|---|---|---|---|---|---|
| VM alfa | No. 1 | No. 1 | No. 1 | No. 1 | No. 1 | No. 1 |
| VM bravo | No. 2 | No. 2 | No. 2 | No. 2 | No. 2 | No. 2 |
| VM charlie | | | No. 3 | No. 3 | No. 3 | No. 3 |
| VM delta | | | No. 4 | No. 4 | No. 4 & 5 | No. 4 |

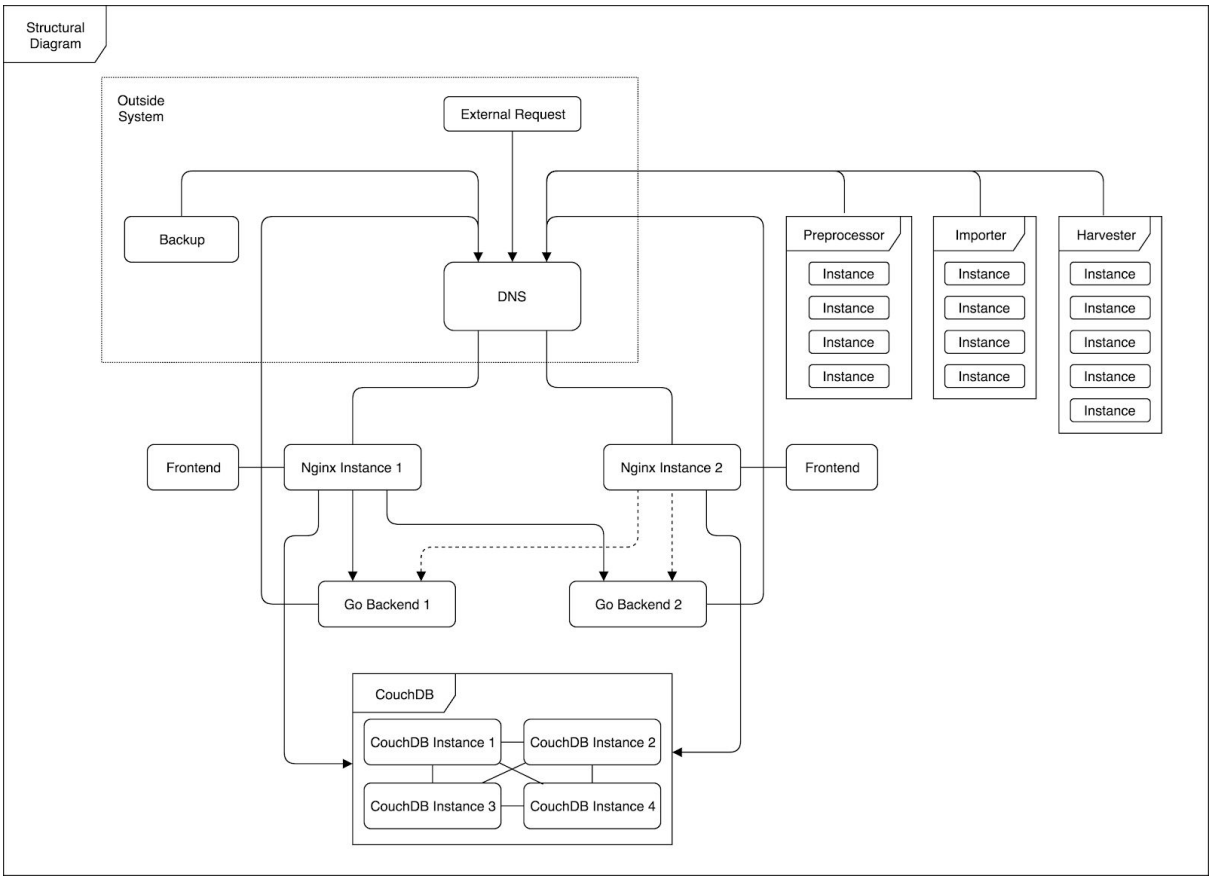



Figure 1: Structural diagram of system

Note that only CouchDB instances will communicate with each other directly. The Backup is a VM that is outside of our whole system. We use it to continuous backup our system (The reason for this can be found in the Fault Tolerance section) and run a port forwarder so that we can access our system outside the school's network in a nice way.

## Data Flow

We are gathering tweets from two sources, one is harvestering tweets using Twitter official API in real time and the other is fetching pre-collected early tweets from UniMelb Research Cloud. The cut-off date is 2 April 2018, so we only keep tweets that are created after this date from the first source and fetch pre-collected tweets before this date. The two diagrams below display the data flow in detail. The reason of introducing each step as well as the responsibilities of each step can be found under the sections of the related services.

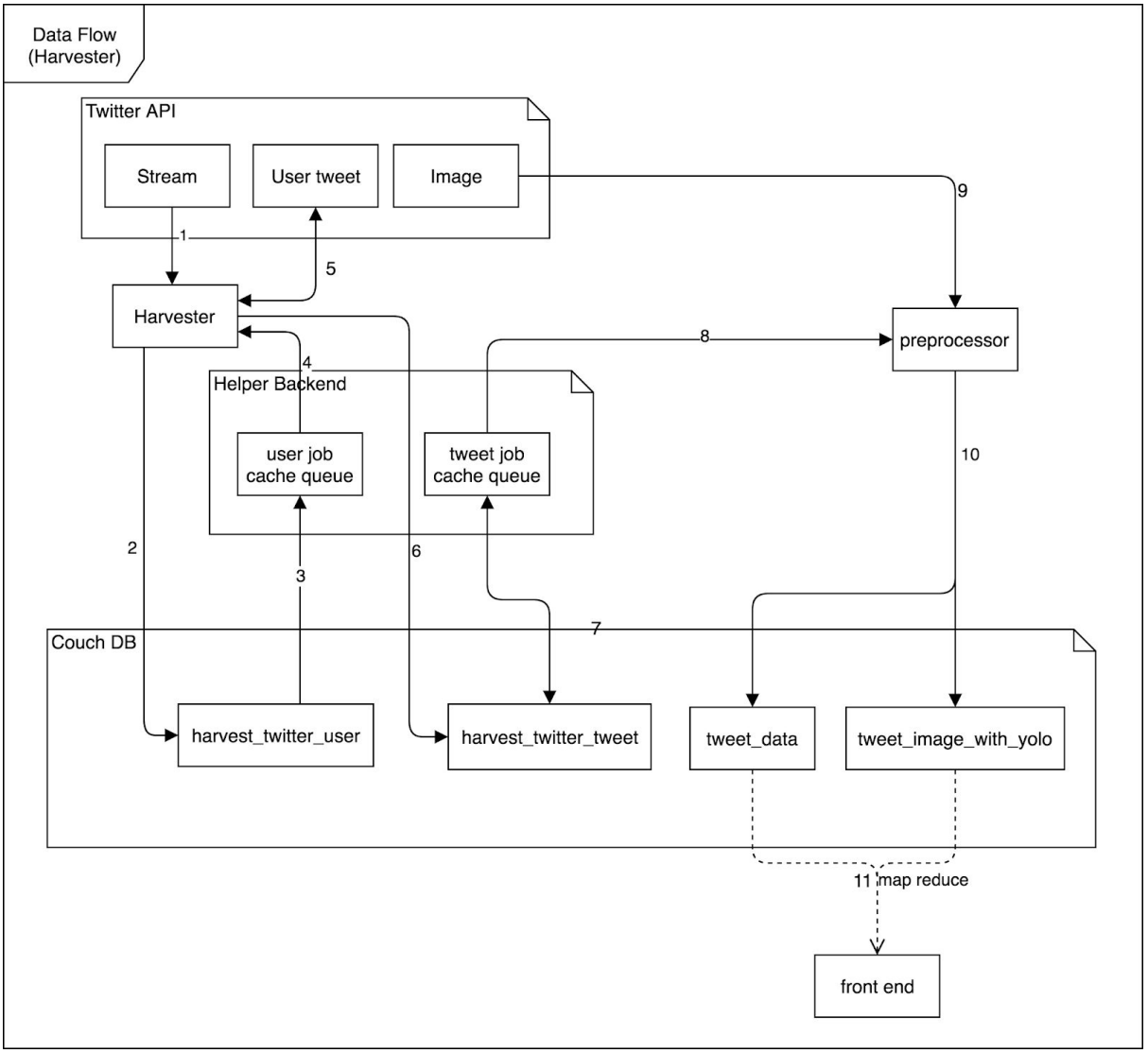



Figure 2: Data flow diagram for harvester

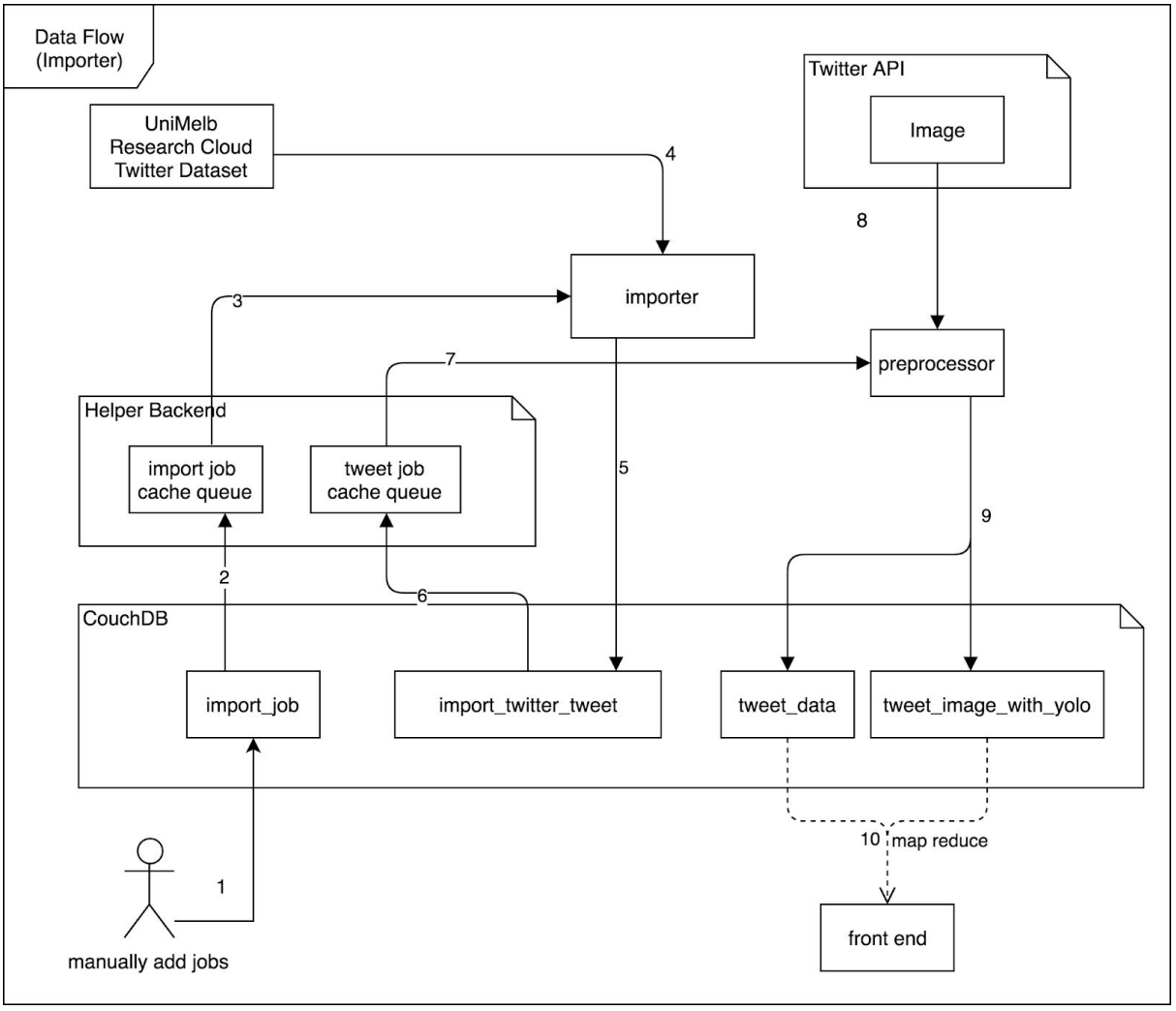



Figure 3: Data flow diagram for importer

## Publish–Subscribe pattern

Almost all kinds of jobs in our system are distributed using the publish-subscribe pattern. Basically each worker will pick (and lock) a pending job from the queue, process it and then submit the result as well as mark the job as done before picking the next job. To achieve this, we need a locking and message queuing mechanisms and here is how we implemented.

### Locking

We simply using CouchDB as our distributing locking system instead of adding new service such as Redis. The general idea is to keep it simple and we are aware that the CouchDB does not provide strong consistency, it only achieves eventual consistency. Due to the type of our jobs, this won't cause any trouble since all the work can be considered as an idempotent operation. The worse case is just some jobs have been done twice. There is a natural unique key come with each tweet ("id_str") and we use this one as the document key in our database to avoid any duplicated results. The locking system we used here is based on Optimistic Lock and the "_rev" attribute of a couchdb document is very useful here. Two more custom attributes are added which are "lock_timestamp" and "finished". The lock process is first to check if "finished" is false and "lock_timestamp" is set a "timeout" period

ago and then set the “lock_timestamp” to the current timestamp. If the job is done then just set “finished” to true and nothing need to be done if the job failed. Other worker will pick this job again after timeout. The “_rev” is used to make sure noone change the document during the whole process (If some did, just abort the current job).

### Message Queue

Our initial idea was also using CouchDB as message queue. However, the problem is that even with appropriate indexes, the mango query in CouchDB still has a terrible performance in some cases. Also unlike MongoDB, there is no way to search and update a document within one request, which increases the chance of having a conflict. In order to solve this issue, we added a helper backend (implement in Golang and more details can be found under the helper backend section). It simply caches the query and distributes the results to all workers. After received the job id from the backend, the worker will try to lock the job and proceed.

## Fault Tolerance

In general, all jobs and requests are distributed to corresponding services evenly. If the gateway detects a service is no longer available then no more request will pass to it. The same for all workers, a stopped worker is definitely not able to fetch any more jobs. As for jobs that the worker locked but not finished, those jobs will be fetch by other workers after timeout. The helper backend is an exception but it has a failover. The failure of the gateway itself can be solved by the DNS server outside. There are more detail on this under the Service section.

Since we are using the Microservice, in the scenario of all instances of a specific service failed (for example a buggy update), other service will remain operational.

The single point of failure here is that all our VMs are running on the same cloud and in the same availability zone. There was actually one time that all of our VMs stop working at the same time. Due to this reason, we add a backup server that sync with our main CouchDB (The other reason is that the volume snapshot of nectar is not an atomic operation). More details on this can be found in the Melbourne Research Cloud section.

## Service

### Nginx Gateway

We use Nginx as our gateway and load balancer since it is a very common choice.

It serves the following things:

- Frontend: from static files mounted to it on port 80 with path “/”.
- CouchDB: pass to one of the CouchDB instance on port 8080 with path “/”.
- Helper Backend: pass to the first backend instance (and switch to the second one if the first one is down) on port 8080 with path “go_backend/”

As for the Nginx Gateway itself, we launched two instances and used the DNS to handle the load balance and Failover. However, since all four of our VMs do not have public IP and there is no way for the DNS part to check the availability of our Nginx Gateway. Therefore, this part is disabled right now.

### Helper Backend

This backend has two purposes, including providing a helper method for the frontend and messaging queue cache for other works.

Our system will run YOLO v3 object detection on the tweet images. The frontend needs to display images with object annotations (bounding box and label for each detected object). This backend provides an API for generation that kind of images.

Recall that there is a performance issue if we use CouchDB directly as a message queue. So we built this backend and it queries the CouchDB to get a batch of unprocessed jobs and cached those into the memory. When there is a worker asking for job then it retrieves one directly from the memory and will query for the next batch if the current batch is finished (The actual implementation is a bit more complex in order to achieve better performance but the idea is the same). In practice, there is no more noticeable loads on the CouchDB as well as lock conflict between workers.

However, the backend is the only part that we cannot split the traffic to all instances since it caches the queue to its local memory. So on the one hand, our gateway only sends request to the first instance and will switch to the second one only if the first one no longer available. On the other hand, we don't guarantee this backend to preserve any kind of consistency. When the worker got a job id from the backend, it still needs to lock the job to make sure it is valid.

### CouchDB

We have 4 CouchDB instances in cluster mode on 4 different VMs and we use the default parameters which are 8 shards and 3 replicas. As for the authentication, we disabled the admin part and use the CouchDB built-in user system. One admin user for all internal services and one read-only user for the frontend.

### Importer

The task of the importer is fairly simple, it is pretty much a data ETL tool. It will fetch a job and follow the configuration of the job to download, filter and transfer tweets form the UniMelb Research Cloud pre-collected Twitter dataset. In our practice, we split the whole work based on the date and each job is to process tweets of a single day.

### Harvester

The task of harverster is to crawl real-time tweets using Twitter official API.

Each harvester is given different Twitter developer tokens and uses streaming API to monitor events from a given area. Afterwards, it extracts the user account information and put it into user account database (harvest_twitter_user) since the user has activity inside our interested area. The other part of

the harvester will scan each user's timeline to get their historic tweets every 10 hours. All tweets that contain geographic location and within Greater Melbourne area will be stored. Additionally, We store the newest tweet id of each user and the tweet id is actually time ordered and we use it to reduce duplicated request in the next scan.

Harvester is one of the service that can not be scaled out directly by adding a new instance. The extra step is to add a new document in the "config" database with a new set of twitter developer tokens and the specific monitoring area. Each harvester will find the corresponding configuration document and lock it before crawling tweets. It will stand by if the document has been locked or restart itself if the document changed.

#### Preprocessor

As the term suggests, preprocessor is used to preprocess the raw tweet data. It is actually very similar to the map stage of the CouchDB MapReduce, we use this service to handle things that can be tricky. Two main things that will be handled by this service are:

It will download all images of the given tweet, then run YOLO v3 object detection algorithm and save the results. Even though YOLO v3 is a fairly fast algorithm, it is CNN based and we only have CPU instances. So the preprocessing is actually a very heavy task.

The "created_at" attribute of a tweet is a UTC time and we need to convert it to melbourne local time in order to analyse the tweets distribution over day time of each suburb. It is a tricky task, especially cause Melbourne has Daylight Saving Time. Also, it is worth noting that the JavaScript engine of CouchDB is kind of old, which is also a bit hard to use external library. Thus we process this timezone thing in the preprocessor.

### Frontend

In our implemented website application, there are three sections for demonstration purpose, including Project, Visualization and Statistics. Project page includes a brief description of this project and group member information. Statistics page presents the real-time statistical data of the Greater Melbourne area, including total number of tweets and time distributions of imported and harvested tweets. For visualization, a customized map was developed based on the service provided by Mapbox GL JS using JavaScript React library, in order to illustrate the statistics of different areas as well as display sample tweets. The geometric data is accessed by making HTTP requests to CouchDB API. All visualization objects and statistics charts shown in the website are drawn using React libraries, which will also do asynchronized requests to the backend. To be mentioned, the areas are divided by the Statistical Areas Level 2 (SA2s) 2016.

## Software Integration & Deployment

### Ansible

To set up instances and orchestrate the necessary software environment, Ansible is used in our system. There are several stages - create instances, run common configuration, configure CouchDB cluster,

deploy front-end web app, configure Nginx and run dockers. They are illustrated in the following diagram.

Video link: https://youtu.be/HiE8ropg83Q?t=19

Note: We have used a dynamic inventory approach. OpenStack provides a python script called 'openstack_inventory.py' which is listed under 'automation' folder in our Github repository. This script has an error about 'cloud' attribute in OpenStack. However, this error is not fatal and can be ignored during automation.

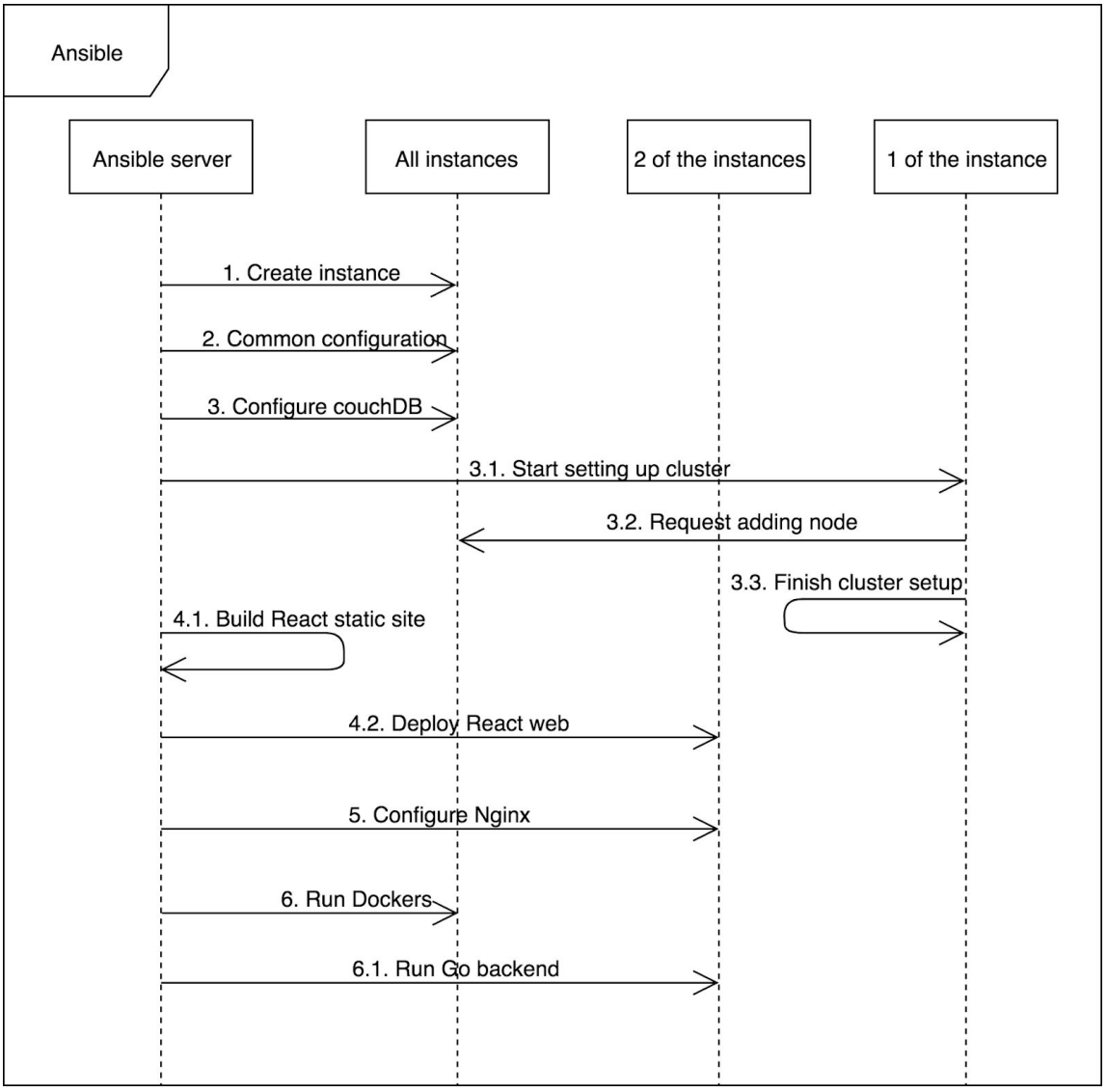



Figure 4: Communication diagram for Ansible

The detailed steps are listed below:

1. Create instances using OpenStack API provided by Melbourne Research Cloud.
    a. On the deployment machine (localhost)
        i. Install and update pip and install OpenStack SDK.
        ii. Create a security group with rules that
            - Within this security group, all instances can communicate each other

- Open port 22 for SSH
- Open port 80 for HTTP
- Open port 8080 for CouchDB cluster and Helper Backend

iii. Create volumes for each of the instance, two of them have 63 Gigabytes, other two have 62 Gigabytes.

iv. Create key pairs so that the ansible deployment host can connect to all instances.

v. Create four instances with one security group, one key pair, one volume respectively

**Note**: the image id might be expired over time. It is dangerous to search for the image name and then use that as the image. The administrator should change the image ID manually in the configuration file.

2. Run common configurations. Some basic configurations, like proxy settings, are needed for each instance.
    a. On each one of the four instances, run:
        i. Configure proxy settings in '/etc/environment'
        ii. Change the time zone to Australia/Melbourne, restart cron so that all tasks are using the same time zone.
        iii. Turn on NTP syncronyzation for more accurate time.
        iv. Install Docker and its dependency
        v. Restart the SSH session so the settings take effects.

3. Configure CouchDB cluster. For fault tolerance, data integrity and stability, a cluster of database is essential for the whole cloud. To set up the CouchDB database:
    a. On all instances, run:
        i. Install docker-py, which is used by Ansible's Docker Module
        ii. Pull the latest image of CouchDB
        iii. Write CouchDB 'ini' and 'vm.args' file, including each node's name, port range and cookie. We also make sure that UUID and HTTPD_AUTH secret is identical across all nodes. Otherwise, "replications may be forced to rewind the changes feed to zero, leading to excessive memory, CPU and network use.", according to the official Apache CouchDB Documentation.
        iv. Start CouchDB docker on each node
        v. Send 'enable_cluster' request to all nodes.
    b. CouchDB does not have the concept of 'master' in the cluster. However, a node is required to act as the centre node to orchestrate the whole cluster when setting up. On one of the instance, run the following commands:
        i. Send two requests to the rest of the instances to add nodes.
        ii. Send a request to itsself to finish setting up the cluster.

    It is worth to note that the schema of the database can be checked with the file named 'common_script/db_dump' folder.

4. Deploy front-end web app. We use React framework and Node.js to build a static website and then hosted by Nginx.

    a. On the deployment machine (localhost)
        i. Install nodejs and npm.
        ii. Build the static site. Note that the heap size and memory size need to be sufficient. We use 'node --max_old_space_size=15000 /usr/bin/npm run build' to build the site.
    b. On two of the instances:
        i. Fetch the whole built site from the deployment machine.
        ii. Update Nginx if needed.
5. Configure Nginx.
    a. On two of the instances:
        i. Pull the newest Nginx image
        ii. Fetch the Nginx configuration file from the deployment machine.
        iii. If the image has been upgraded or if the Nginx configuration has been changed, stop and remove the old Nginx docker and then restart the docker.
6. Run dockers to harvest, import and process Twitter data.
    a. On the deployment machine (localhost)
        i. Login to the private registry with writing access.
        ii. Build Harvester, Importer, Preprocessor and Helper Backend dockers with respect to the latest git repository.
    b. On all of the instances:
        i. Login to the private registry with read-only access.
        ii. Pull and run Harvester, Importer and Preprocessor dockers.
    c. On two of the instances:
        i. Login to the private registry with read-only access.
        ii. Pull and run Helper Backend dockers.

## Docker

In this project, we have utilized Dockers to deploy our applications, such Harvester, Importer, Preprocessor, Helper Backend and Nginx. We have also taken advantage of GitLab's private registry services so that we can push our images into it and pull from the other instances easily.

By leveraging docker technology, it is nearly effortless when deploying them. A single 'docker run' command with pre-configured arguments is adequate for the service to be up and running. In addition, you do not need to worry about the environment you are working in. It would be tedious process to configure the server exact the same as the environment for development and multiple services running on the same server makes this situation even worse. For example, if one of the service is running using Python2 and the other is using Python3, it is extremely painful if Docker or other environment configuration tools like 'virtualenv' are not used. Furthermore, the log can be accessed easily and we are not required to remember all the locations of logs for different services.

On the other hand, the docker images occupy a lot of disk spaces. Although some common layers existing in the private registry can be reused and won't be downloaded again, the image size on the disk can be as large as one Gigabytes. Additionally, the running speed of containers is slower than bare-metal speeds. Despite the fact that the container technology is more efficient than old VM

technology, the cost of bridging network and exchanging data with the operating system via the host system can not be evitated.

## Melbourne Research Cloud (MRC)

Melbourn Research Cloud provides OpenStack API which enables configuration management and application deployment tools, such as Ansible, to automate the process of creating and managing instances on the cloud. By utilizing the dynamic inventory, we can deal with arbitrary number of instances but two or more instances should be a minimum requirement to allow features like load balancing and fault tolerance to take effects.

All of our Internet traffic goes through MRC proxy by default, which seems to be inconvenient. However, without proxy, all of us can directly access the external internet, which does not ensure any security, as proxy makes it possible for the Melbourne Research Cloud to monitor the traffics between the intranet and internet. Also, the proxy enables caching, which probably increase the access speed. But proxy still brings a shortage that no different IP can be used to harvest the tweets data. As a result, Twitter might ban the account that harvest the tweets too frequently according to its IP address.

Another disadvantage would be that the snapshots of volumes are not atomic. In other words, the snapshot can not restore the data at the point of time when the snapshot is created since MRC website states that "creating a snapshot from an attached volume can result in a corrupted snapshot". If we want to make a snapshot, the only option we have is to unmount the volume, which results in downtime for our database service. In order to provide zero-downtime functionality, atomic snapshot should be available.

On 12th of May, connections to all four instances of our system cannot be established. The states of these instances were showing running. Unfortunately, the only thing action we can take is to delete and re-run our Ansible playbooks to create and config four new instances. These instances are under the same network, availability zone and create within a continuous time interval, they are highly likely to be allocated in the same data center, even on the same rack. If all of these instances are disconnected or having fatal errors which results in losing connections to the visitors, any fault tolerance, load balancing, failover will be useless in this situation. To remediate this situation, multi-region active-active architecture can be used. That is, having servers located in different data centre geographically to decrease the probability of all instances failing at the same time. Additionaly, management tools like Kubenetes can be utilized when services are out of reach. The process of discovering the downtime and create instances and running the services should be automated.

We use docker for every single service of our system, so what we actually need is a Managed Kubernetes as a Service or Platform as a Service (PaaS). Comparing what we are using now, that is a Infrastructure as a Service (IaaS) cloud computing, PaaS has a more simple approach to deploy instances and they service can be dynamically scaled with respect to the demand.

# Dataset

Our project datasets are based on the suburbs in Greater Melbourne area. The division follows the Statistical Areas 2 (SAs2) 2016 provided by Australian Statistical Geography Standard (ASGS). All the data discussed later corresponds to one of the suburbs.

## Crawler

Over 272 thousand pieces of data have been collected, including imported data and harvested data. It is worth noting that in fact, less than one percentage of data are stored in our database, since tweets with geographic location are quite rare through exploration.

## Pre-processing

The crawled raw data will be processed by extracting keywords from text content, recognizing items from YOLO object detection. Meanwhile, only necessary attributes in each piece of Twitter JSON data are kept for saving database spaces.

## Post-processing

Each CouchDB database includes a design document that can contain a list of views implemented by users. Each view has a map function and reduce function, which enables the user to extract the data and present it in a specific order, and also filter the documents in the database to find the relevant ones. In this project, 3 views are used:

1. The view that accumulates the number of gluttony tweets and the number of all tweets in each suburb.
2. The view that counts the number of tweets in different time interval in each suburb.
3. The view that calculates the total number of tweets for each day in Melbourne.

# Sentiment Analysis

For the seven deadly sins (pride, greed, lust, envy, gluttony, wrath and sloth), we initially picked sloth as our theme. However, after several tries, it was found that judging whether a tweet is related to sloth is a hard job. Therefore, we finally chose the sin gluttony. Two method will be used in our project, including keyword analysis on text and object detection on image.

## Keyword analysis

For each tweet data, we tried to look for keywords in the text content that are possibly related to gluttony scenario, such as “bread”, a specific food, “Mcdonald”, a restaurant, and “bowl”, a tablewale. To sum up, the words we are seeking for is not only words of “food”, but any words that might have relation with food.

## Food word dictionary from Wordnet

The Python dictionary Wordnet was imported, which was used to generate a list of words related to the word “ food”. As a result, around 3000 “gluttony” words were achieved. However, the word list

still includes several words that is not highly related to food. For example, the words “course” and “collins”, which have ambiguous meanings, will make the analysis inaccurate. Therefore, the words list is manually modified, with some of the words or phrases deleted.

### Regular expression for word matching

When we tried to determine whether a word is in the tweet text simply using Python method “in”, we found some cases that this method is not sufficient. For instance, the word “scrabble” is matched with “crab”, which leads to a wrong detection. To solve this problem, we decided to use regular expression, which restrict that the word matched should not be following or followed by any other letter immediately.

### Yolo object detection

Yolo is object detection model which was trained using Deep Neural Network (DNN). The model can recognize up to 80 objects occurred in an image, including bottle, wine glass, cup and so on, with around 25 of them related to “gluttony” by manual observation. Therefore, yolo is quite efficient to help judge whether a tweet image contains gluttony content in this project.

## Stories in Melbourne related to Gluttony

### Initial Assumption

Initially, we thought that the gluttony tweets might have relationship with the factors such as obesity and overweight. Therefore, the dataset “SA2 Health Risk Factors - Modelled Estimate 2011-2013” is achieved from AURIN, which includes information on prevalence of health risk factors including obesity and overweight, for further analysis.

### Analysis

After calculating the correlations between the gluttony tweets count and the Obesity and Overweight data collected from *Aurin,* it is found that there are no obvious relationship between them in Melbourne. Therefore, we tried to look for another field that might have correlation with gluttony tweets.

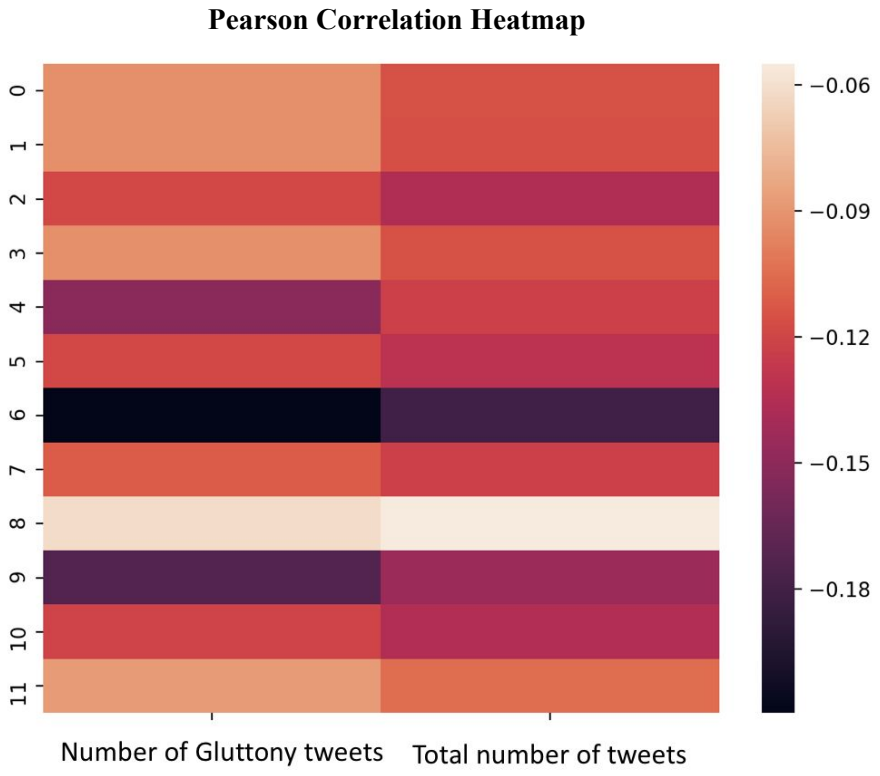



Figure 5: Heatmap for Health Risk Factors - Modelled Estimate vs Gluttony / All tweets

## Further Assumption

It is assumed that the tendency that a person posts a tweet that can be recognized to be gluttony is related to the age of the person. Therefore, several dataset related to age are achieved from AURIN. For each of them, the pearson correlation matrix is calculated and shown as a heatmap. Finally, we found some interesting relations between the gluttony tweets and the data describing the people's marital status by age in Melbourne.

## Number of total / gluttony tweets vs Marital status by age in Melbourne

After calculating the correlations between the dataset "SA2-G05 Registered Marital Status By Age By Sex-Census 2016" from AURIN and the statistical result we made based on the collected gluttony tweets, we found that several factors are highly related to the number of gluttony tweets.

- Age: 20-24
- Gendar: Female
- Marital status: Never married

Therefore, the 36 columns related to these three factors are selected and analysed.

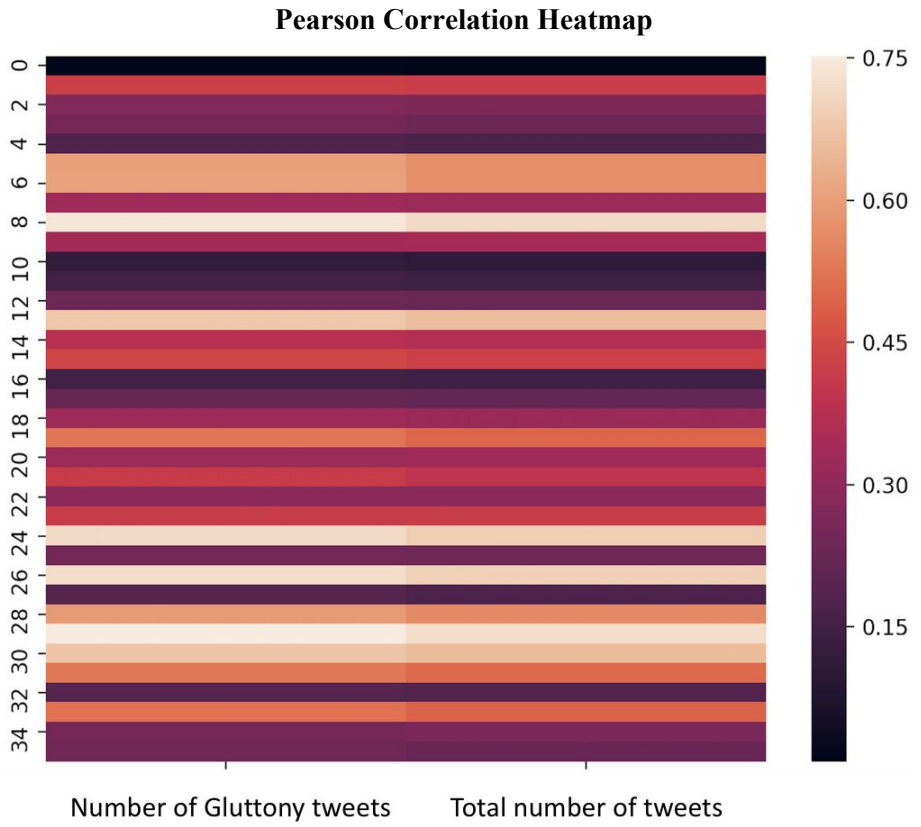



Figure 6: Heatmap for Registered Marital Status By Age vs Gluttony / All tweets

The correlation between the Gluttony tweets and the different fields in this dataset is shown above. The most significant ones of them is listed below:

| index | Description | Correlation with gluttony tweets | Correlation with all tweets |
|---|---|---|---|
| 8 | the number of female from 20 to 24 years old | 0.742 | 0.716 |
| 24 | the number of people from 20 to 24 years old | 0.715 | 0.692 |
| 26 | the number of never married people from 20 to 24 years old | 0.723 | 0.700 |
| 29 | the number of never married female from 20 to 24 years old | 0.752 | 0.725 |

Figure 7: Table for certain fields and its correlations

In order to visualize the correlations between these four fields and the number of gluttony tweets, four graph will be shown. However, as there are total 309 suburbs in Melbourne according to SAs2, while some of them have very few tweets harvested, which are regarded as outliers and remove from the graph. Aslo, the range of x axis and y axis are selected to make the graph contain more information.

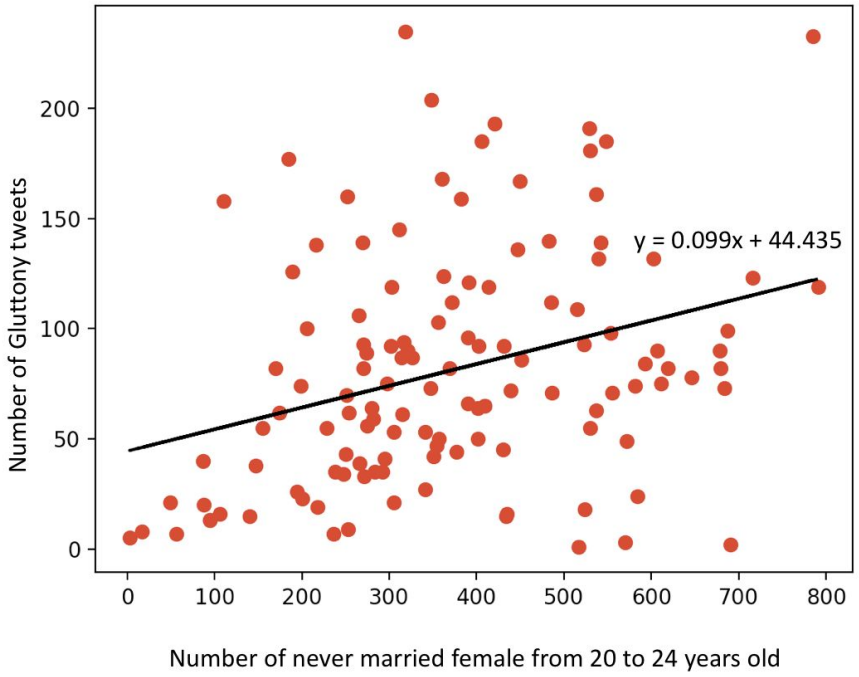



Figure 8: Gluttony tweets vs Never married female from 20 to 24 years old

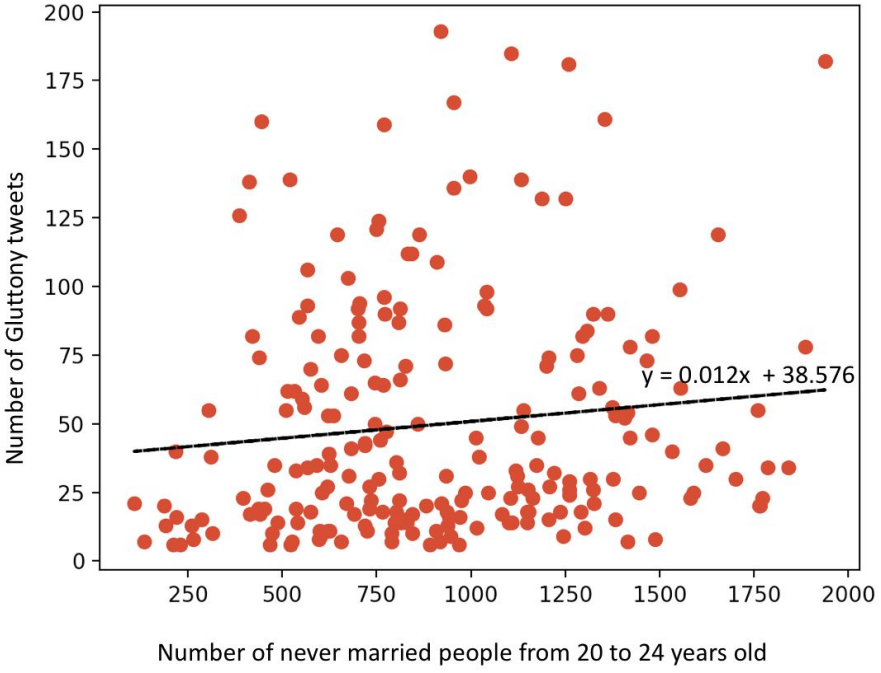



Figure 9: Gluttony tweets vs Never married people from 20 to 24 years old

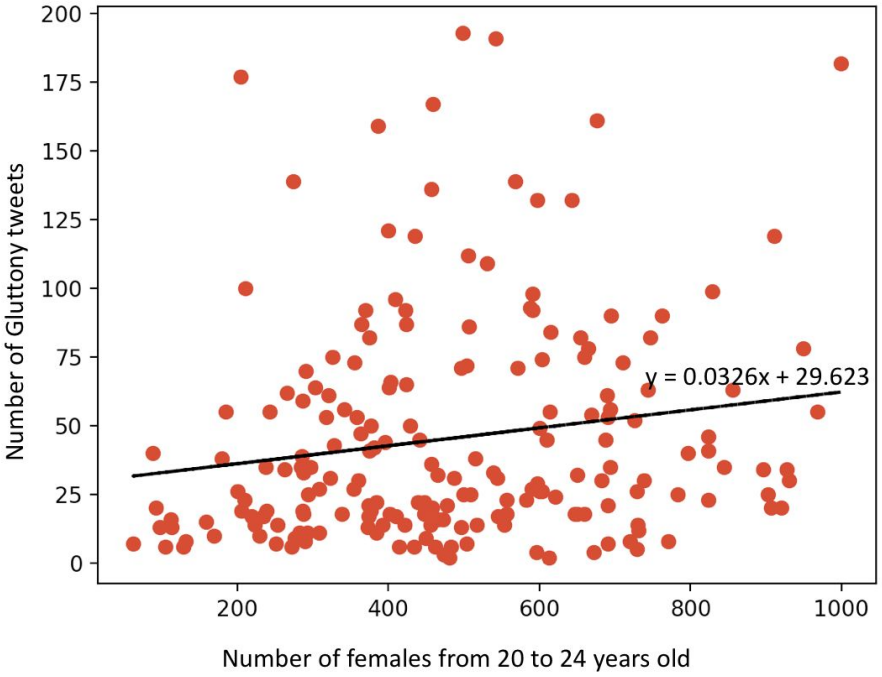



Figure 10: Gluttony tweets vs Females from 20 to 24 years old

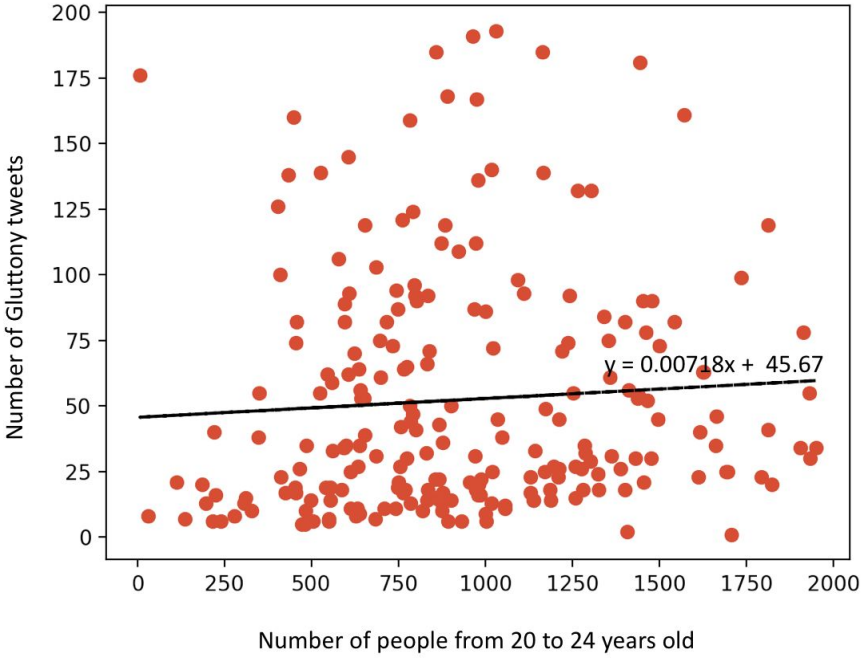



Figure 11: Gluttony tweets vs People from 20 to 24 years old

## Scenario Analysis

According to the correlations shown in the table, it can be found that the people aged from 20 to 24 years are likely to post more tweets than other groups of people. Also, they are more interested in posting gluttony tweets. Moreover, in this particular group of people, if they are either female or never married, the probability for them to post gluttony tweets is enhanced. Especially, the never married females aged from 20 to 24 years old have the highest possibility to post gluttony tweets.

Based on our assumptions, as the people from 20 to 24 years old are either university students or those who have just had a job, it can be easily imagined that this group of people are new to the society and more relying on social media, for the sake of sharing their feelings, emotions and experiences with others who might just become their friends. Also, the correlation table can conclude that more females tend to post tweets than males, which is also a common scene.

Lastly, the influence of females' marital status on their potentials to send gluttony tweets is the most interesting discovery. This indicates that a female are more likely to post gluttony tweets before they are married. One of our thoughts is that a female who's married or has once married must had her own family, which might make them focus on their families more than the social media. However, either the correlations found or the analysis does not necessarily mean that the never married female from 20 to 24 years old are likely to be gluttony, instead, which might due to that this group of people simply enjoy posting Twitter texts or images whenever they have a meal. Furthermore, this might also be the reason that it is hard for us to find any correlations between the gluttony tweets data and the obesity data collected from Aurin, as a obesity or overweight person does not always posts what he eats, and a person who posts what he eats does not necessary to be obesity or overweight. Instead, what really matters is whether a person enjoys posting tweets.

# User Guide

Source Code: https://github.com/CaviarChen/CCC_Project_2
Video Link: https://youtu.be/HiE8ropg83Q
Website: www.ccc.8bits.io
Website outside unimelb: outside.ccc.8bits.io

## Ansible Setup

Firstly, we need a deployment server with an image of 'Ubuntu 18.04'. If the server does not have the public Internet access, proxies should be added into '/etc/environment'.

```
http_proxy="http://wwwproxy.unimelb.edu.au:8000"
https_proxy="http://wwwproxy.unimelb.edu.au:8000"
ftp_proxy="http://wwwproxy.unimelb.edu.au:8000"
```

Then, we need to install Ansible on the deployment machine by

```
sudo apt-get update && sudo apt-get install
software-properties-common
sudo apt-add-repository --yes --update ppa:ansible/ansible
sudo apt-get install ansible
```

Next, you will need to clone our repository. The repository used to be private so the server's SSH public key need to be added as deployment key on Github at that time. By now, a single command may be sufficient

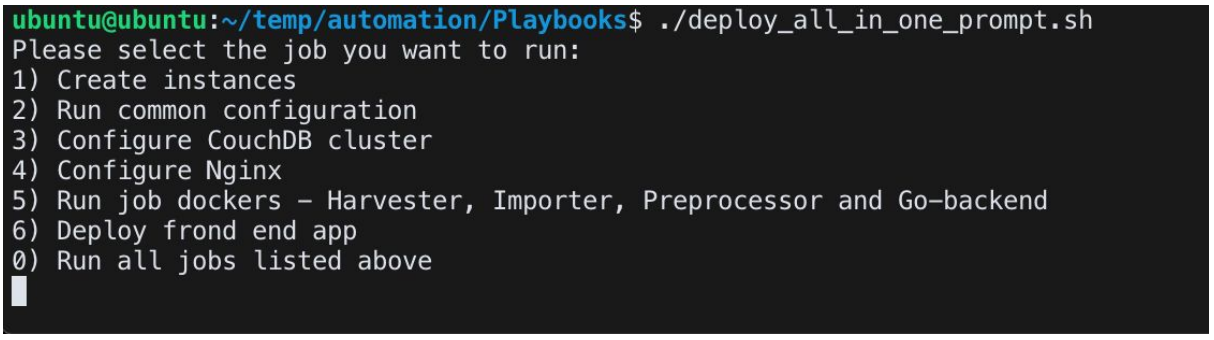
```
git clone git@github.com:CaviarChen/CCC_Project_2.git
```

If you have problem connecting to Github using SSH, add following lines to your ~/.ssh/config

```
Host github.com
  Hostname ssh.github.com
  Port 443
  User git
  ProxyCommand nc -X connect -x wwwproxy.unimelb.edu.au:8000 %h %p
```

After the repository has been successfully cloned, navigate to 'automation/Playbooks' folder. Under 'host_vars' folder, in the file named 'comm_config_vars.yaml', change the 'ansible_ssh_private_key_file' to your private key file location. You can also config the instance attributes, including name, availability zone and flavor in the file named 'creation.yaml'.

After that, run 'deploy_all_in_one_prompt.sh' will give you the following prompt:




In some cases, Openstack API password, SUDO password and/or Ansible vault password might be needed before proceeding. Please contact us if you need any of these passwords.

Before you deploy front-end application, an up-to-date Node.js package is compulsory. Since the default Ubuntu repository only has Node.js 8.10 (14/05/2019), we need to run the following command:

```
curl -sL https://deb.nodesource.com/setup_12.x | sudo -E bash -
sudo apt-get install -y nodejs
```

Now, all the set up with respect to Ansible should be completed and you can now deploy instances using the ‘deploy_all_in_one_prompt.sh’ script.

# Reference


1. Miller, G. A. (1995). WordNet: a lexical database for English. *Communications of the ACM*, 38(11), 39-41.
2. Redmon, J., & Farhadi, A. (2018). Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*.
3. Government of the Commonwealth of Australia - Australian Bureau of Statistics, (2017): SA2-G05 Registered Marital Status by Age by Sex-Census 2016; *accessed from AURIN Portal on 2019-05-14*.

   Torrens University Australia - Public Health Information Development Unit, (2014): SA2 Health Risk Factors - Modelled Estimate 2011-2013; accessed from *AURIN Portal on 2019-05-14*.
4. Cluster Set Up (n.d.). Retrieved from https://docs.couchdb.org/en/stable/setup/cluster.html
5. OpenStack Dynamic Inventory Python Script (n.d.). Retrieved from https://raw.githubusercontent.com/ansible/ansible/devel/contrib/inventory/openstack_inventory.py