# COMP90024 2023 Semester 1 Assignment 2 Report

Chenyuan Li
1405583
Taylor Tang
1323782

Qiao Chen
1391014
Yannei Xu
1275106

Shaoyu Wang
1255602

May 25, 2023

## Executive Summary

Our project introduces a comprehensive analytic tool designed to analyze an interesting yet profound topic: "Is there a correlation between alcohol, crime and crash incidents? And if so, how is the public's attitude towards alcohol? Does this different from places to places?". To realize these scenarios, we have developed an integrated cloud-based system, deployed on Melbourne Research Cloud (MRC). This system comprises various interconnected elements, including a frontend, backend, database, and data crawlers, all functioning harmoniously to provide a complete web application service.

Our frontend is built upon React.js, offering an intuitive and engaging user interface that enhances user experience. The backend, developed with Flask, utilizes ReSTful API principles to seamlessly communicate with the frontend and the database. For data storage, we implemented a CouchDB cluster to ensure enhanced data availability, fault tolerance, and overall system reliability. Our robust data foundation is derived from a mix of resources: the Spatial Urban Data Observatory (SUDO), Australian Urban Research Infrastructure Network (AURIN), a provided Twitter dataset, and a custom-built Mastodon crawler. Once collected, this data is processed and stored in the CouchDB cluster, then further analyzed using MapReduce to create insightful visualizations. To streamline operations and efficiently manage cloud resources, we leverage Ansible and OpenStack API to automate service deployment and scaling on MRC. This automation approach enables our system to smoothly operate under fluctuating demands and deliver reliable performance even during peak usage.

In summary, this project highlights the successful integration of various technologies in building a robust, cloud-based analytic tool that provides valuable insights into public attitudes towards alcohol. While challenges were encountered along the way, the successful outcomes and achievements are testament to our resilience and strategic planning, resulting in a well-executed analytic system.

***Keywords-*** SUDO; AURIN; Twitter; Mastodon; React; Flask; CouchDB; Ansible; Docker

# Contents

**1 Introduction** **3**
- 1.1 Project objective . . . 3
- 1.2 Project solution . . . 3
- 1.3 User guide . . . 4

**2 System architecture and implementation** **5**
- 2.1 System architecture . . . 5
- 2.2 Melbourne Research Cloud . . . 5
- 2.3 Ansible Deployment . . . 7
- 2.4 Docker and container . . . 9
- 2.5 Database - CouchDB . . . 10
- 2.6 Frontend - React . . . 12
- 2.7 Back-end - Flask . . . 13
- 2.8 Mastodon Crawler . . . 14

**3 Data collection and processing** **14**
- 3.1 Twitter data collection and processing . . . 14
- 3.2 Mastodon data collection and processing . . . 15
- 3.3 SUDO data collection and processing . . . 15
- 3.4 CouchDB and MapReduce . . . 15

**4 System functionalities** **16**
- 4.1 Purpose of Data Analysis . . . 16
- 4.2 Scenario 1: Do alcohol selling points have correlations with local crime rate and crash incidences? 16
- 4.3 Scenario 2: How does crash incidents correlated to liquor vary in different time? . . . 18
- 4.4 Scenario 3: How does public attitudes towards alcohol vary in a day? . . . 19

**5 Challenges and Issues** **21**
- 5.1 Development using MRC . . . 21
- 5.2 Data Collection . . . 21
- 5.3 Packages and APIs . . . 22

**6 Conclusion and Future Improvements** **22**

**7 Project Management** **23**
- 7.1 Roles and Responsibilities . . . 23
- 7.2 Project governance . . . 23
- 7.3 Individual contribution . . . 24

**8 Video Links** **25**

**9 GitHub Link** **25**

# 1 Introduction

## 1.1 Project objective

Alcohol plays a diverse role in society. Public's attitude towards alcohol range from joyous associations like "parties", "ceremonies", and "romance", to negative connotations such as "drink drive", "crimes", and "headache". Bearing this curiosity, we sought to unravel the public sentiment towards alcohol. More specifically, we have designed a few analytic scenarios:

1. How does public attitudes towards alcohol consumption vary in different geolocation?
2. How does public attitudes towards alcohol vary in different time?
3. Do alcohol selling points have correlations with local crime rate and crash incidences?

This study can make important contributions to urban planning, targeted intervention, policy-making and public awareness. Our findings could be incorporated into land-use planning by considering the correlation between liquor license locations and alcohol-related incidents This study could also guide law enforcement parties and community organizations to allocate resources and proactively conduct targeted interventions with higher alcohol-related incidents in the area and time slots. Additionally, our findings can directly contribute to policy decisions regarding issuing and placing liquor licenses. Moreover, public awareness of the risk associated with alcohol-related incidents could also be empowered by our findings.

Armed with these objectives, we embarked on an ambitious journey to collect data and construct an online analytical web application, alongside its associated services, to unveil these insights.

## 1.2 Project solution

In order to answer the posed questions, we constructed a full-scale web application service. This platform leverages cutting-edge cloud computing technologies and practices and is deployed on the Melbourne Research Cloud (MRC). Our service ecosystem consists of five main components:

1. Frontend Service: Designed using React.js, our frontend provides an intuitive and responsive user interface, enabling users to interact with our analysis tool and visualizations effortlessly.
2. Backend Service: Developed on the Flask framework and embracing RESTful API principles, our backend serves as a critical conduit between the frontend and the database, processing user requests and delivering results dynamically.
3. Database Service: We implemented a CouchDB cluster to serve as our database. This design choice ensures enhanced data availability, fault tolerance, and overall system reliability.
4. Data collection: To collect a vast and diverse range of data, we employed custom-built Mastodon Crawlers. Other supplements such as Spatial Urban Data Observatory (SUDO), Australian Urban Research Infrastructure Network (AURIN), and Twitter datasets were used.
5. Deployment: In order to ensure seamless scaling, both for our web service and data collection processes, we implemented scripted deployment using Ansible playbooks in conjunction with the OpenStack API. This approach enables script-based scaling, allowing our system to adapt efficiently to varying demands.

Each component plays a crucial role, and together they form a cohesive, highly functional analytical platform. This system is not just built to answer our initial research questions but can also be used to further study alcohol related topics.

## 1.3 User guide

The user guide is to provide an instruction of visiting frontend web page and testing the ansible deployment.

### 1.3.1 Frontend web application

User may connect to UniMelb VPN or using university network to visit the frontend web application at: `172.26.131.182:80`. A demonstration of how to use the web page service can be viewed at: `https://youtu.be/hf0FpbN_wo8` and `https://youtu.be/S6bDqL0qT6g`

### 1.3.2 Multiple Mastodon crawler deployment

The multiple crawler deployment utilizes the shell commands, ansible playbook and openstack API. User needs to input several arguments by following the shell command prompts as per table 1. User can also remove multiple crawler at once as per table 2.

Table 1: Command for Deploying Crawler

| **Field** | **Details** |
|---|---|
| Command | `./deploy_crawler.sh` |
| Arguments Required | 1. Enter the number of instances: (User enters argument)<br>2. Enter Mastodon server: (User enters argument)<br>3. Enter CouchDB IP address: (User enters argument)<br>4. Enter CouchDB database name: (User enters argument)<br>5. Confirm and enter password: (User enters argument) |
| Description | This command will gather required security groups and create a number of instances with the specified settings. For each instance, this command will update dependencies, install Docker, pull the crawler image and deploy services with arguments. |
| Video Demonstration | `https://youtu.be/Vx2fgt98ORU` |

Table 2: Command for Deleting Instances

| **Field** | **Details** |
|---|---|
| Command | `./delete_instances.sh` |
| Arguments Required | Enter the TXT file name: (User enters argument) |
| Description | This command takes the TXT file containing instance information generated from the above command, and removes the instance on MRC accordingly. |
| Video Demonstration | `https://youtu.be/Vx2fgt98ORU` (from 5:30) |

Table 3: Command for Full Deployment

| **Field** | **Details** |
|---|---|
| Command | `./full_deployment.sh` |
| Arguments Required | 1. Confirm to deploy: (User enters argument)<br>2. Enter password: (User enters argument) |
| Description | This command will use default instance settings to create two instances and share the instance information with each other. For the first instance, the command will create a CouchDB service with a network volume of 50gb storage, and a Mastodon crawler service to harvest Mastodon.au toots, saved to CouchDB. For the second instance, the command will create a frontend service and a backend service by pulling the custom images from Docker Hub. The frontend is connected to the backend, and the backend is connected to CouchDB on instance 01. |
| Video Demonstration | `https://youtu.be/jOUa72gokDI` |

### 1.3.3 Full service deployment

The full service deployment utilizes the shell commands, ansible playbook and openstack API. To start the deployment, the user needs to key in the sh command, and confirm the deployment as per table 3.

## 2 System architecture and implementation

### 2.1 System architecture

Our system is designed and deployed on the **Melbourne Research Cloud**, employing **Ansible** and the **OpenStack API** for deployment. The system's architecture comprises several key components: a **CouchDB cluster** with distributed storage for data storage and pre-processing, a **React** frontend service that presents analyzed and visualized data to users, a **Flask** backend service that processes frontend queries and retrieves data from the database, and multiple **Mastodon crawler** services scraping data from various Mastodon servers. As per figure 1.

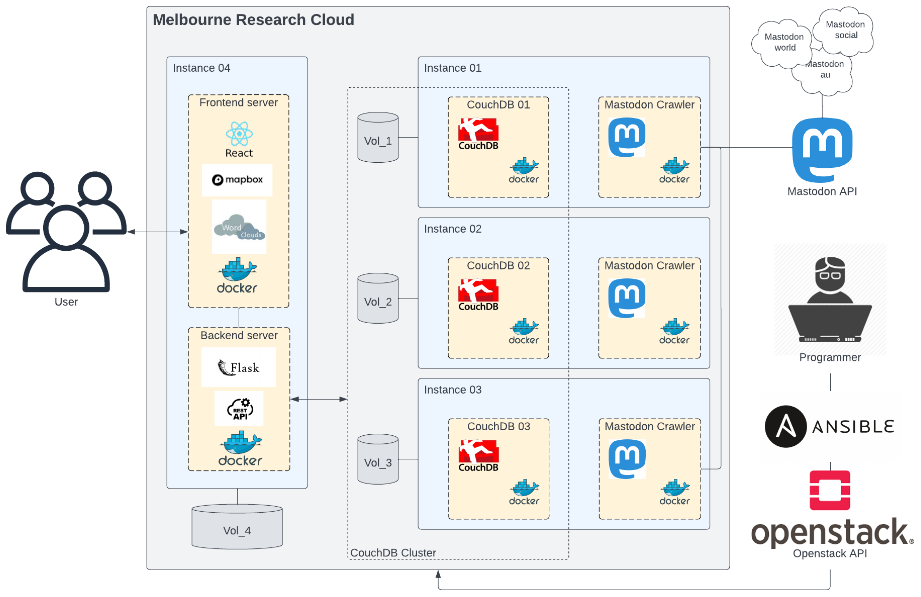


Figure 1: System architecture

In the following subsections, we will delve into the role of each component, discuss their advantages and shortcomings, compare them with alternative solutions, and highlight our implementation.

### 2.2 Melbourne Research Cloud

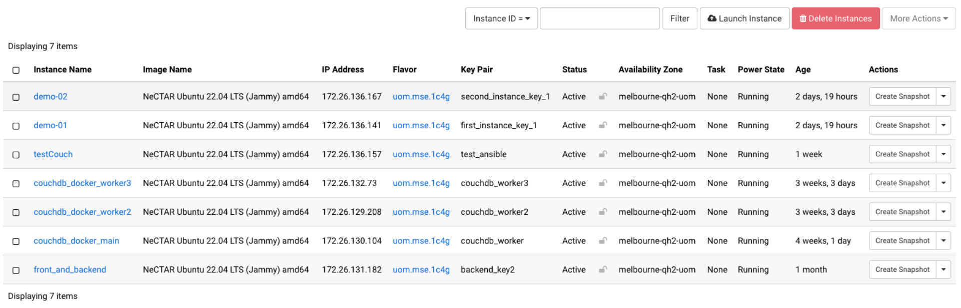
The Melbourne Research Cloud (MRC) presents researchers at the University of Melbourne and associated institutions with complimentary, readily available computing capabilities. It serves a similar purpose to that of commercial cloud services like Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform. Typical applications involve data scrutiny and website hosting, among various other possibilities.

#### 2.2.1 Role

In this project, each team participating has been granted eight virtual CPUs and 500Gb of storage space. In our final product, we allocated four CPUs to create four instances. One instance was designated to host both the frontend and backend services, while the other three were used to host CouchDB nodes, forming a CouchDB cluster, and a Mastodon crawler to collect data from mastodon.au, mastodon.social, and mastodon.world.

Out of the total storage, we employed 200Gb, distributed equally across the four volumes. One volume was tasked with storing raw Twitter and SUDO data, while the other three were used within the CouchDB cluster to store processed Twitter, SUDO, and harvested Mastodon posts, also known as ”toots”.

The remaining resources were kept in reserve for presentation needs, allowing us to deploy multiple Mastodon crawlers dynamically by executing Ansible scripts or to deploy the entire system as needed, as per Figure 2. An overview all the MRC usage is presented in this video: `https://youtu.be/pQJ5kDbtCig`



Displaying 7 items

| Instance Name | Image Name | IP Address | Flavor | Key Pair | Status | Availability Zone | Task | Power State | Age | Actions |
|---|---|---|---|---|---|---|---|---|---|---|
| demo-02 | NeCTAR Ubuntu 22.04 LTS (Jammy) amd64 | 172.26.136.167 | uom.mse.1c4g | second_instance_key_1 | Active | melbourne-qh2-uom | None | Running | 2 days, 19 hours | Create Snapshot |
| demo-01 | NeCTAR Ubuntu 22.04 LTS (Jammy) amd64 | 172.26.136.141 | uom.mse.1c4g | first_instance_key_1 | Active | melbourne-qh2-uom | None | Running | 2 days, 19 hours | Create Snapshot |
| testCouch | NeCTAR Ubuntu 22.04 LTS (Jammy) amd64 | 172.26.136.157 | uom.mse.1c4g | test_ansible | Active | melbourne-qh2-uom | None | Running | 1 week | Create Snapshot |
| couchdb_docker_worker3 | NeCTAR Ubuntu 22.04 LTS (Jammy) amd64 | 172.26.132.73 | uom.mse.1c4g | couchdb_worker3 | Active | melbourne-qh2-uom | None | Running | 3 weeks, 3 days | Create Snapshot |
| couchdb_docker_worker2 | NeCTAR Ubuntu 22.04 LTS (Jammy) amd64 | 172.26.129.208 | uom.mse.1c4g | couchdb_worker2 | Active | melbourne-qh2-uom | None | Running | 3 weeks, 3 days | Create Snapshot |
| couchdb_docker_main | NeCTAR Ubuntu 22.04 LTS (Jammy) amd64 | 172.26.130.104 | uom.mse.1c4g | couchdb_worker | Active | melbourne-qh2-uom | None | Running | 4 weeks, 1 day | Create Snapshot |
| front_and_backend | NeCTAR Ubuntu 22.04 LTS (Jammy) amd64 | 172.26.131.182 | uom.mse.1c4g | backend_key2 | Active | melbourne-qh2-uom | None | Running | 1 month | Create Snapshot |

Displaying 7 items

Figure 2: System architecture

### 2.2.2 Benefits

**Cost-Efficiency**: A key advantage of using cloud services is the potential for cost savings. Rather than investing heavily in hardware and ongoing maintenance, users pay only for the resources they actually use. In this project, students receive cloud services at no additional charge, which is embedded in their institutional fees, eliminating the need for additional expenditures on hardware.

**Scalability and Flexibility**: Cloud services offer unmatched scalability and adaptability. Users can effortlessly adjust their IT resources according to their requirements, allowing businesses to grow without the need for costly modifications to their existing IT systems. In this project, students are allocated eight CPU resources, which can be configured to either a 1-core-4-gb or 2-core-8-GB setup, depending on the scenario. Students can also customize their volume sizes anywhere between 1 and 500 GB, offering a significant advantage in managing resources effectively and preventing wastage.

**Data Recovery and Backup**: Cloud services offer reliable data backup and recovery solutions, which are quicker, more affordable, and more dependable than traditional on-premise methods. Within the context of this project, MRC offers snapshot functionality for instances and volumes, which can be utilized under various conditions including recovering from service failure, or creating new services using these snapshots.

**Collaboration and Accessibility**: Cloud platforms enable universal access to data at any time from anywhere, using any internet-enabled device. This promotes collaboration, allowing team members to view, access, and modify documents or share resources simultaneously. Within this project, all team members have the ability to utilize the actual resources, ensuring a collaborative environment.

**Security**: Despite some apprehensions, cloud services typically provide stronger security measures than an individual organization can establish. High-level encryption and multiple layers of security controls are typically offered to safeguard stored data.

**Sustainability**: Lastly, utilizing cloud services promotes environmentally friendly practices. It decreases energy consumption and reduces the carbon footprint compared to traditional data centers, particularly where unused resources still require power.

### 2.2.3 Limitations

**Dependency and Outages**: Cloud services are wholly reliant on a stable internet connection. Interruptions in internet service result in loss of access to data and applications. Furthermore, service outages can occur due to technical difficulties faced by cloud service providers. For instance, during this project, the Melbourne Research Cloud (MRC) occasionally responded slowly or even incorrectly when creating new instances, highlighting some risks associated with cloud service dependency.

**Security and Privacy**: Despite robust investments in security by cloud providers, storing sensitive data in the cloud can entail risk. Privacy becomes a concern when data is housed remotely, under the stewardship of a third party.

**Limited Control**: Customers have limited control and influence over the cloud infrastructure as it is fully owned, managed, and monitored by the service provider. For example, during this project, we as developers did not have direct authority over the data stored on the cloud server. Additionally, our CPU and storage resources were limited by the development environment, denying us the opportunity to request additional resources.

**Data Transfer**: The need to transfer all data over the internet to MRC presents a risk in terms of speed and stability of data transfer, adding another layer of complexity to this project. Additionally, students not using the campus network can only access MRC via VPN, further complicating the data transfer process.

**Performance**: The performance of cloud services can be affected by user numbers, bandwidth restrictions, and the physical distance between the user and the server, potentially leading to slower load times or latency issues. During this project, network problems impacted our frontend loading speed, illustrating this drawback.

### 2.2.4 Alternative solutions

While there are various cloud service platforms available in the market, for the purpose of this project, our choices were limited due to project constraints. The project guidelines specified the exclusive use of the Melbourne Research Cloud (MRC) for deploying our services.

## 2.3 Ansible Deployment

Ansible is an open-source software provisioning, configuration management, and application deployment tool. It utilizes a simple language (YAML, in the form of Ansible Playbooks) that allows you to describe your automation jobs in a way that approaches plain English.

### 2.3.1 Role

Our project utilizes Ansible Playbook in conjunction with the OpenStack Cloud API to streamline the deployment of multiple instances and services. This automation assures consistent configurations across diverse instances while mitigating error risks inherent to manual setups. The OpenStack Cloud API, meanwhile, grants us programmatic control over MRC resources. It facilitates the configuration of security groups, volumes, and other necessary elements, thereby simplifying the process of establishing a functional environment for hosting CouchDB, frontend, backend and Mastodon crawler services.

### 2.3.2 Benefits

**Simplicity**: Ansible uses a simple language YAML that allows tasks to be described in a way that closely approximates plain English. This makes the tool easy to use for most programmers.

**Consistency**: With Ansible, you can ensure that your deployments and environments are consistent, reducing the likelihood of errors that often accompany manual setups.

**Automation**: Ansible enables the automation of repetitive tasks, freeing up time for more complex tasks.

**Versatility**: Ansible can be used to manage a broad range of systems - from a few local devices to large-scale, multi-cloud infrastructures.

### 2.3.3 Limitations

**Complexity for Complex Tasks**: While Ansible is relatively straightforward to start a simple project, it does have a learning curve, particularly for complex project deployment or orchestration.

**Lack of GUI**: Ansible lacks a graphical user interface, making it less approachable for those accustomed to GUI-based tools.

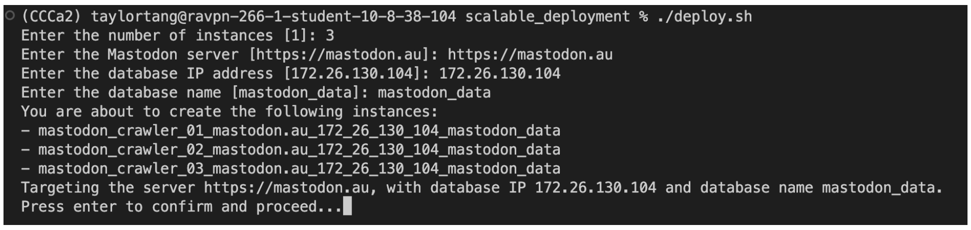
### 2.3.4 Alternative solutions

While there are several other configuration management and deployment tools available, such as Puppet, Chef, and SaltStack, the use of Ansible was a specified requirement for this project. Despite any limitations or challenges associated with learning a new tool, the successful use of Ansible in our project confirms its efficacy and versatile capabilities in a variety of deployment scenarios.

### 2.3.5 Implementation highlights

In this project, we have successfully implemented dynamic and full-system deployment capabilities through two distinctive shell scripts.

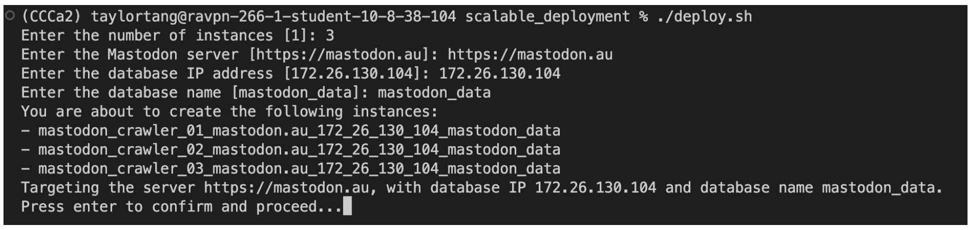
The first shell script enables a flexible deployment of our Mastodon crawling service. It prompts 3 the user to specify the number of instances to deploy, the target Mastodon server to be crawled, and the desired database for storing the scraped data. This eliminates the need for manual playbook script configuration, enabling rapid and tailored deployment of Mastodon crawlers (as depicted in the diagram). Furthermore, the IP addresses of the created instances are saved, providing an efficient means of managing these instances for tasks such as batch rebooting or deletion. This is demonstrated in this video: https://youtu.be/Vx2fgt98ORU



```
(CCCa2) taylortang@ravpn-266-1-student-10-8-38-104 scalable_deployment % ./deploy.sh
Enter the number of instances [1]: 3
Enter the Mastodon server [https://mastodon.au]: https://mastodon.au
Enter the database IP address [172.26.130.104]: 172.26.130.104
Enter the database name [mastodon_data]: mastodon_data
You are about to create the following instances:
- mastodon_crawler_01_mastodon.au_172_26_130_104_mastodon_data
- mastodon_crawler_02_mastodon.au_172_26_130_104_mastodon_data
- mastodon_crawler_03_mastodon.au_172_26_130_104_mastodon_data
Targeting the server https://mastodon.au, with database IP 172.26.130.104 and database name mastodon_data.
Press enter to confirm and proceed...
```

Figure 3: Ansible multi crawler deployment

The second script offers a comprehensive one-stop solution for deploying the full suite of services: the frontend, backend, database, and crawling service. This provides the convenience of setting up the entire system with a single command 4. This is demonstrated in this video: https://youtu.be/jOUa72gokDI



```
(CCCa2) taylortang@ravpn-266-1-student-10-8-38-104 scalable_deployment % ./deploy.sh
Enter the number of instances [1]: 3
Enter the Mastodon server [https://mastodon.au]: https://mastodon.au
Enter the database IP address [172.26.130.104]: 172.26.130.104
Enter the database name [mastodon_data]: mastodon_data
You are about to create the following instances:
- mastodon_crawler_01_mastodon.au_172_26_130_104_mastodon_data
- mastodon_crawler_02_mastodon.au_172_26_130_104_mastodon_data
- mastodon_crawler_03_mastodon.au_172_26_130_104_mastodon_data
Targeting the server https://mastodon.au, with database IP 172.26.130.104 and database name mastodon_data.
Press enter to confirm and proceed...
```

Figure 4: Ansible full service deployment

For a more detailed walkthrough of our Ansible automated deployment playbook, please refer to the demonstration video available in the link section below. This video provides an in-depth guide of our deployment process, elucidating the robust capabilities of our project.

## 2.4 Docker and container

Docker is a renowned open-source platform that facilitates automation in deploying, scaling, and managing applications by using containerization. It wraps software into standardized units called containers that have everything needed to run the code, including the runtime, system tools, libraries, and settings.

### 2.4.1 Role

In our project, Docker plays an essential role in ensuring seamless deployment and management of our services across different instances. Docker's ability to package software into containers allows for consistency across multiple development and production environments, thereby enabling smooth and predictable deployments. Moreover, we upload pre-configured images to DockerHub, and then pull images to each instance with containerized environments, to start the services such as CouchDB, backend, frontend, and the Mastodon crawler.

### 2.4.2 Benefits

**Consistency**: Docker containers ensure applications run the same way anywhere, reducing the time to configure each instance's program running environments.

**Efficiency**: Docker containers are lightweight and fast. They share the host system's OS kernel, making them much more efficient than VMs.

**Fault tolerance**: Each Docker container runs separately and in isolation from other containers, providing greater fault tolerance. Moreover, docker containers are supported with self-healing features. If one goes wrong, it can be restarted automatically.

**Scalability**: Dockerized applications can be easily scaled up and down by using programmatic deployment.

### 2.4.3 Limitations

**Container Management**: Managing many docker compose yaml, dockerfile, individual containers and their interconnections can become complex.

**Security**: While containers are isolated, they share the host OS. Any breach into the host can potentially affect all containers.

**Persistent Data Storage**: Containers are ephemeral, meaning they do not inherently store data persistently. It can be difficult to inherit previous image settings on updated container images.

### 2.4.4 Alternative solutions

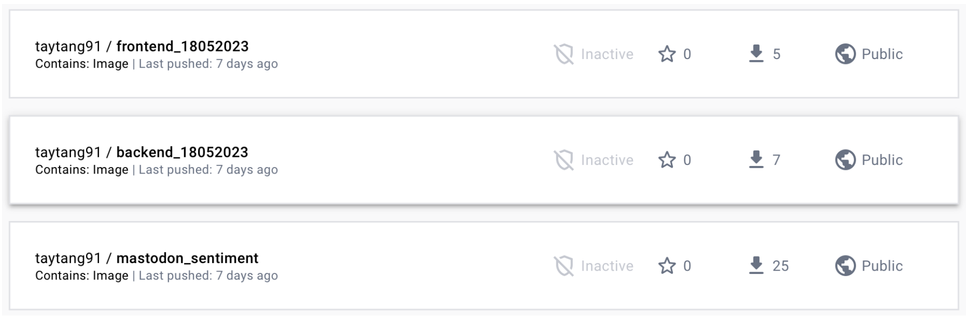
There are several alternative containerization technologies, such as LXC, rkt, and Podman. In the IT industry, Kubernetes are widely used to automate deploying, scaling and operating application containers. Kubernetes provides features such as service discovery and load balancing, storage orchestration, automated rollouts, self-healing etc. makes it a great tool for orchestrating containers in a large-scale, distributed environment.

That being said, Kubernetes can be used to replace the combination of Ansible and Docker in this project. However, the learning curve is quite high and it might be overkill for this project. Therefore, we chose Docker for this project due to its broad adoption, great community support, and extensive documentation. Docker's ease of use, robust functionality and compatibility made it a highly suitable choice for our needs.

### 2.4.5 Implementation highlights

In this project, we leverage DockerHub and pre-configured images to expedite service deployment. Initially, we configure the desired services manually, testing our Python and React.js code by initiating Docker containers with the requisite arguments. After confirming the proper functioning of the service, we upload the complete image to DockerHub 5.

Subsequently, when initializing a new instance, we can directly pull the image specified in the Docker-compose YAML template file. This method circumvents the need for cloning from our Git repository and configuring Docker in the Ansible playbook scripts, significantly simplifying the use of Ansible and OpenStack API.



| | | | | |
|---|---|---|---|---|
| taytang91 / **frontend_18052023**<br>Contains: Image \| Last pushed: 7 days ago | Inactive | 0 | 5 | Public |
| taytang91 / **backend_18052023**<br>Contains: Image \| Last pushed: 7 days ago | Inactive | 0 | 7 | Public |
| taytang91 / **mastodon_sentiment**<br>Contains: Image \| Last pushed: 7 days ago | Inactive | 0 | 25 | Public |

Figure 5: Docker Hub Images

We also utilize the self-healing feature from the docker-compose function. All our services are deployed automatically by using ansible playbook, in which we have created a docker-compose template file and set the restart policy to "always". In this way, if any of the service crashes while running, docker will try to restart the service automatically. This is demonstrated in this video: `https://youtu.be/8eX2vrSrz1A`

We did not implement docker swarm for our service due to the following considerations: 1) most of the desired features that we have implemented can be provided by using docker-compose; 2) additional network latency that caused by the swarm overlay network may reduce the effectiveness of our service; 3) difficult to be implemented with couchDB cluster because of dynamic IP address and data persistence, it is more effective to manage couchDB cluster nodes locally.

## 2.5 Database - CouchDB

CouchDB is a versatile open-source NoSQL database that uses JSON to store data, JavaScript for indexing documents, and HTTP for its API. Equipped with features such as a web-based dashboard, built-in replication, CouchDB is an entry level user-friend database tool.

### 2.5.1 Role

Our project utilizes CouchDB as its primary database system, effectively managing and storing diverse data sets from SUDO, twitter dataset and harvested Mastodon dataset. Furthermore, we exploit its clustering and synchronization features to ensure data consistency across our distributed systems, which adds another layer of data security. Essential features such as Mango Query and MapReduce functions are actively used for data retrieval and processing for frontend web applications.

### 2.5.2 Benefits

CouchDB offers numerous benefits, including the ability to store data in a flexible JSON-based format that simplifies handling complex data structures. The database's built-in replication and conflict resolution facilitate data consistency in distributed systems. Its HTTP/REST-based API integrates seamlessly with our project, and the web-based dashboard allows for intuitive data management. The Mango Query feature enables efficient querying of the JSON data, while the MapReduce functionality aids in processing and analyzing large data sets. CouchDB's NoSQL capabilities excel in processing unstructured and semi-structured data, making it an excellent choice for this project.

### 2.5.3 Limitations

Despite its powerful features, CouchDB does have certain limitations. Being a NoSQL database, it lacks support for relationships between data entities, which can be a challenge for specific data modeling tasks. Its performance may lag compared to traditional SQL databases, especially with substantial data volumes.

### 2.5.4 Alternative solutions

Though CouchDB is the chosen database solution for this project per the guidelines, other NoSQL databases such as MongoDB or DynamoDB, or relational databases like MySQL or PostgreSQL could potentially be utilized. However, each has its strengths and weaknesses, and the choice would depend on the specific requirements of the project. In this case, CouchDB's ease of use, JSON data storage, and powerful replication capabilities make it a suitable choice.

### 2.5.5 Implementation highlights

In our project, we've employed a CouchDB cluster to amplify the robustness and stability of our database service. This cluster system fortifies our data availability and resilience against faults by distributing data across multiple nodes 6. In the event of a node failure, this arrangement ensures unbroken accessibility to the data. Video demonstration: `https://youtu.be/5Ivhj-KSMg4`

Additionally, our CouchDB cluster enhances the scalability of our database. It offers us the flexibility to dynamically scale up or down, according to the volume of CRUD requests. This adaptability is a key advantage in managing the demands of our database effectively.

Moreover, the cluster configuration bolsters the performance of our database. It achieves this by distributing read and write requests over the multiple nodes of the cluster. This tactic is notably beneficial when handling substantial data volumes or a high count of requests. Thus, our CouchDB cluster not only ensures reliability and scalability but also significantly improves our database's performance.

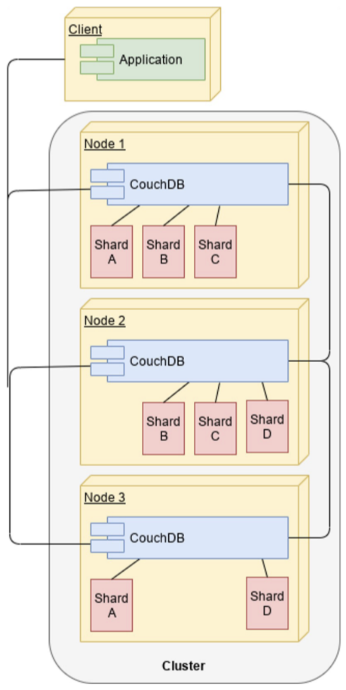


Figure 6: CouchDB data redundancy

## 2.6 Frontend - React

React.js is a powerful and popular JavaScript library for building
user interfaces and web applications. It's known for its reusable component structure, efficient rendering, and ease of integration with other technologies.

### 2.6.1 Role

In this project, React.js is leveraged to develop the frontend, providing a dynamic and interactive user interface. The modular nature of React components promotes code reusability and facilitates easier maintenance and updates. Its virtual DOM feature is also exploited to enhance our application's performance by minimizing real DOM manipulations.

### 2.6.2 Alternative solutions

While React.js was chosen as per project guidelines, there are alternative solutions for frontend development like Angular and Vue.js. Each offers different philosophies and features. Angular provides a comprehensive framework with robust features but has a steeper learning curve. Vue.js, on the other hand, is renowned for its simplicity and gradual learning curve but might lack some advanced features. Despite these alternatives, React.js's component-based architecture, efficient rendering, and strong community support make it an ideal choice for this project's frontend development.

### 2.6.3 Implementation highlights

**Decoupling the Frontend and Backend**: In our developed system, the frontend and backend are decoupled, which means they are developed, deployed and maintained separately. The frontend and backend communicate via the API that is exposed by the backend. Thus, the frontend will only focus on capturing the client's request and delivering and visualising the requested data; Our backend will only focus on handling requests and establishing CRUD operations with the database. When the client interacts with our system, the data or the resource requested by the client will be delivered to the client through the ReSTful design.

**Mapbox**: The map on the Home page is an important visualization that displays the density of liquor sales points and car accidents in Victoria. We chose to use Mapbox to implement this feature. To achieve dynamism and reusability in the frontend system, instead of simply importing data into Mapbox Studio, we opted to draw the base map using Mapbox GL JS code. We then utilized the ”add source” functionality to add data that was received from the backend. The retrieved data was then added to the map using the ”add layer” function. However, the data returned from the backend was in JSON format, while Mapbox GL JS only supports the GeoJSON file format and not JSON. To address this, we included conversion code in the frontend to transform the JSON data received from the backend into GeoJSON format before importing it into the map.

Since we used React to build the frontend, while most Mapbox GL JS tutorials are geared towards HTML development, implementing the map feature took us a significant amount of time to resolve.

Currently, the implemented Mapbox map displays liquor sales points using circles or clustered circles with corresponding numbers, while car crash cases are shown using a heatmap. Clicking on a liquor sales point reveals detailed information about that particular sales point. Additionally, the map supports selecting data for any year from 2017 to 2019 through a dropdown menu. Furthermore, by toggling the visibility of layers, users can choose to view liquor sales points and car crash cases individually or compare them side by side, allowing for clearer data presentation.

**Wordcloud**: A word cloud component is used in our frontend web application to reflect the harvesting functionality of Mastodon Crawler. This React component is implemented to dynamically visualize the words which are related to attitude or emotional expression. In order to deliver a good user experience and interaction, we have conducted sentiment processing on the toots of Mastodon in our database so that we can provide a relatively clear view of words. In particular, we have a drop-down option list to enable end-users to select either "Positive" or "Negative", and the associated vocabulary will be displayed as a word cloud, in which word sizes are varying based on the frequency mentioned in a few of recent posts. In order to match the theme of the whole presentation, we only focus on the toots which are related to liquor or alcohol. The component is designed and implemented to refresh frequently for catching database updates in an in-time manner. This might be helpful for research or interesting purposes.

Video demonstration of frontend services: Mapbox, twitter  mastodon https://youtu.be/hf0FpbN_wo8 SUDO analysis https://youtu.be/S6bDqL0qT6g

## 2.7 Back-end - Flask

Flask is a lightweight, flexible microframework for Python that's suited for developing web applications and APIs.

### 2.7.1 Role

In this project, Flask plays an essential role in creating a robust backend for our web application. We employ Flask to develop a ReSTful API that interfaces with the frontend and the CouchDB database with its flexibility and simplicity. The API receives HTTP requests from the frontend, processes the data, interacts with the database, and returns appropriate HTTP responses.

### 2.7.2 Alternative solutions

Other web frameworks, like Django, offer more features out-of-the-box, which could potentially speed up development time for complex projects. Node.js with Express.js is also a popular choice for building ReSTful APIs, especially in projects where JavaScript is heavily used. However, Flask was chosen due to its simplicity, ease of use, and compatibility with the Python-based components of this project.

### 2.7.3 Implementation highlights

**Fault Tolerance**: We have designed our backend service to incorporate the CouchDB cluster feature to enhance the resilience of our backend service. Our system stores addresses for all CouchDB nodes, rotating data fetching between them. Should data retrieval fail from one, it attempts the next node, ensuring system continuity even if one or two nodes fail. A more detailed video demonstration: https://www.youtube.com/watch?v=SB4674Mjc7U

**Load balancing**: Since our backend service communicates with all three CouchDB cluster nodes, we have designed our backend to evenly distribute the storing and retrieval loads evenly to all three nodes. Every time the backend service needs to communicate with CouchDB, it iteratively selects the next available node as the communication target. Hence, the load is balanced across the CouchDB cluster.

**ReSTful Architecture**: The backend employs a ReSTful design, with a ReSTful API created within Flask acting as the liaison between the frontend and the CouchDB database. Being stateless and cache-able, this API manages HTTP requests from the frontend, processes these requests, interacts with the database as needed, and returns HTTP responses, ensuring a seamless data flow throughout the application.

**Self-healing**: The backend operates within a Docker container, set to restart automatically upon service failure. This feature enables the system to recover from unexpected errors, reinstating the service without manual intervention.

**Dynamic Deployment**: Our backend is designed for flexibility, allowing configurable database addresses upon initiating the Docker container. This feature facilitates dynamic deployment, demonstrated by our ability to create a CouchDB service on a new instance and automatically link the backend service to this new instance. We showcased this ability in the demonstration video.

## 2.8 Mastodon Crawler

Mastodon provide a public API that let developers to interact with Mastodon data. We utilize this API to harvest toots from Mastodon.

### 2.8.1 Role

In our project, we deploy Mastodon Crawlers to gather 'toots' or posts from multiple Mastodon servers. We use sentiment analysis and sentence tokenization to filter toots related to our analytical theme: public attitude towards alcohol. The harvested information is stored in our CouchDB database and pre-processed via mapReduce for the frontend application, enabling our web app to analyze real-time global data.

### 2.8.2 Alternative solutions

As per the project guidelines, the teams are required to construct Mastodon crawlers, despite the availability of numerous other social platforms, like Twitter, Facebook, Pinterest, Instagram, Tiktok, etc., whose APIs can also be used to harvest live data.

### 2.8.3 Implementation highlights

Our Mastodon crawler is flexible and adaptable. It is currently configured (by specify the server as an argument to start the service) to harvest toots from three top English Mastodon servers (Mastodon.au, Mastodon.social, and Mastodon.world) and can easily be reconfigured to store data in different locations by simply changing the target database address and name.

The crawler is also designed for efficiency. It uses a dual-channel listener to scrape data from both public and private user channels (with provided access tokens), and utilizes a queue for sequential processing of harvested toots. To distribute the saving load evenly, the crawler takes full advantage of our CouchDB cluster to evenly store data across the three CouchDB nodes.

# 3 Data collection and processing

## 3.1 Twitter data collection and processing

After processing, we have around 30,000 tweets posted in Victoria state with valid geography location data, which is judged by the "includes" key of a tweet. Through analysis of the sentiment of the content of these tweets, we can find an interesting story about the impact of liquor on the life of residents in Victoria state. Before uploading all valid tweets in our remote CouchDB database, we still need to add a key called label to the data to make the analysis more convenient for follow-up work. The label is divided into totally three types, liquor, crash, and crime respectively. The first job is to design a method to distinguish what content belongs to which label. Because we already have tokens in tweet data, it is easy for us to traverse every token and check if each token is related to the liquor, crime, or crash. We pick 30 to 40 words for each label to check if it is related or not such as alcohol for liquor, accident for crash, and illegal for crime. In addition, we use a stemmer to transform each word into a stemmed word so that we can avoid missing any related words in different tenses. Meanwhile, we do stemming for each token in tweets as well. Finally, we upload these tweets with a label into the database.

### 3.2 Mastodon data collection and processing

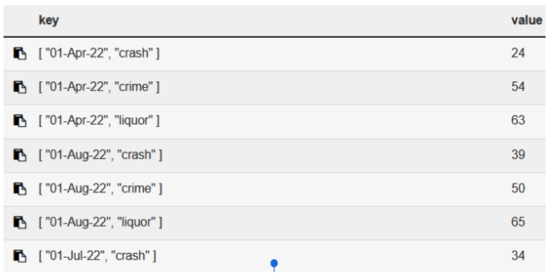
After the mastodon crawler gets toots from mastodon servers, before uploading it, we need to add keys to the original data. Compared with tweets, it needs some additional steps to process because its original data has not any tokens and even its content includes special characters. In the first step, we find that the content of each toot includes angle brackets but they are corresponding fortunately. We use regex in Python to pick the real content out. In the second step, we need to add three labels to the data, tokens, sentiment, and sentiment words. Token's label includes every word in the toot content related to the liquor. The sentiment stands for the sentiment score of the whole toot and the sentiment words label includes the positive or negative words, which is achieved by the NLTK module in Python. After ensuring that the toot is related to the liquor, we upload it.

### 3.3 SUDO data collection and processing

Besides data from the SUDO website, we also collect some data from other official websites including Victory Government Website. This part will be easier for other parts of data processing because most data is stored in the CSV file or the XLS file. The only task we need to do in Python is to extract data with the keys we want from those files and upload it to the database. More processing tasks are completed by the map and the reduce function in CouchDB.

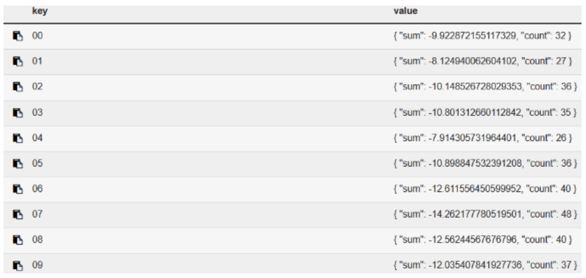
### 3.4 CouchDB and MapReduce

For the convenience to visualize the data on the frontend website, most data in databases of CouchDB is processed by views before being fetched to the frontend. The functions in the view are the map and reduce functions. For the Twitter data, we mainly care about the count of tweets related to liquor, crime, and crashes based on the date and 24 hours of that date, and the positive or negative extent of the content based on 24 hours. The two figures below 7 show what we want. Here is an video demonstration: `https://youtu.be/X0AkBCit8eY`



| key | value |
|---|---|
| [ "01-Apr-22", "crash" ] | 24 |
| [ "01-Apr-22", "crime" ] | 54 |
| [ "01-Apr-22", "liquor" ] | 63 |
| [ "01-Aug-22", "crash" ] | 39 |
| [ "01-Aug-22", "crime" ] | 50 |
| [ "01-Aug-22", "liquor" ] | 65 |
| [ "01-Jul-22", "crash" ] | 34 |

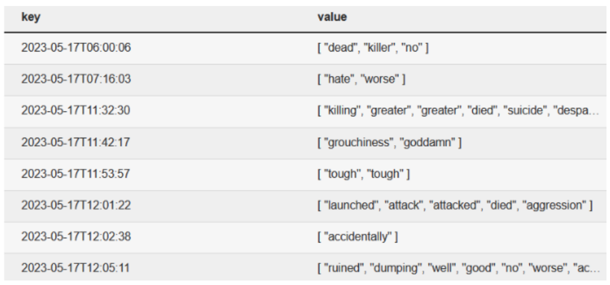
Figure 7: MapReduce



| key | value |
|---|---|
| 00 | { "sum": -9.922872155117329, "count": 32 } |
| 01 | { "sum": -8.124940062604102, "count": 27 } |
| 02 | { "sum": -10.148526728029353, "count": 36 } |
| 03 | { "sum": -10.801312660112842, "count": 35 } |
| 04 | { "sum": -7.914305731964401, "count": 26 } |
| 05 | { "sum": -10.898847532391208, "count": 36 } |
| 06 | { "sum": -12.611556450599952, "count": 40 } |
| 07 | { "sum": -14.262177780519501, "count": 48 } |
| 08 | { "sum": -12.56244567676796, "count": 40 } |
| 09 | { "sum": -12.035407841927736, "count": 37 } |

Figure 8: MapReduce result

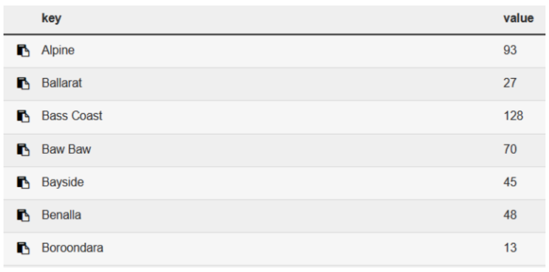
In the first figure, it shows the total number of tweets at a specific time for a specific label. In the second figure, it shows the sum of all sentiment scores of tweets at a specific hour. In addition, we can see that we only write the map function for the mastodon data in some views.



| key | value |
|---|---|
| 2023-05-17T06:00:06 | [ "dead", "killer", "no" ] |
| 2023-05-17T07:16:03 | [ "hate", "worse" ] |
| 2023-05-17T11:32:30 | [ "killing", "greater", "greater", "died", "suicide", "despa... |
| 2023-05-17T11:42:17 | [ "grouchiness", "goddamn" ] |
| 2023-05-17T11:53:57 | [ "tough", "tough" ] |
| 2023-05-17T12:01:22 | [ "launched", "attack", "attacked", "died", "aggression" ] |
| 2023-05-17T12:02:38 | [ "accidentally" ] |
| 2023-05-17T12:05:11 | [ "ruined", "dumping", "well", "good", "no", "worse", "ac... |

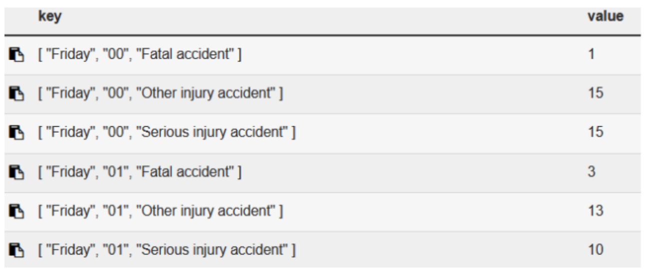
Figure 9: MapReduce result

This figure shows the negative words in the toot related to the liquor by date. For the data needed in the map of the frontend. We mainly analyze the sum of liquor licenses and crime based on each LGA in Victoria below.



| key | value |
|---|---|
| Alpine | 93 |
| Ballarat | 27 |
| Bass Coast | 128 |
| Baw Baw | 70 |
| Bayside | 45 |
| Benalla | 48 |
| Boroondara | 13 |

Figure 10: MapReduce Result



| key | value |
|---|---|
| [ "Friday", "00", "Fatal accident" ] | 1 |
| [ "Friday", "00", "Other injury accident" ] | 15 |
| [ "Friday", "00", "Serious injury accident" ] | 15 |
| [ "Friday", "01", "Fatal accident" ] | 3 |
| [ "Friday", "01", "Other injury accident" ] | 13 |
| [ "Friday", "01", "Serious injury accident" ] | 10 |

Figure 11: MapReduce result

For crash data, we not only count the sum according to LGA but also according to the severity of the car crash.

## 4 System functionalities

### 4.1 Purpose of Data Analysis

The purpose of our data analysis lies in the extraction of meaningful patterns and relevant data columns that will allow us to achieve our project objectives. This analysis can facilitate the identification of recurring trends and reveal any correlations between locations holding liquor licenses and the instances of alcohol-related car crashes or crimes. Through such an approach, we are able to validate our initial assumptions and hypotheses. For instance, one may assume that Melbourne has the highest density of liquor license distribution prior to data analysis, a supposition that is supported by our subsequent results. Moreover, the analysis of data from varied sources serves to enhance the reliability and robustness of our findings by offering a validation process against an additional data source. Lastly, it is worth noting that this process of data analysis can help us filter out most redundant information, allowing for a clearer, noise-free observation of the data.

### 4.2 Scenario 1: Do alcohol selling points have correlations with local crime rate and crash incidences?

This scenario primarily attempts to examine whether there is a geographic relationship between the density of car crashes and the number of liquor selling points from 2017 to 2019. We utilized the 2017-2019 Victoria liquor licenses dataset from SUDO and the 2017-2019 Victoria car crash dataset from the Victorian State Government. This perspective was chosen because both dataset provide precise location information, which means location is a common and crucial factor for analysis.

We conducted the analysis by mapping the locations of liquor selling points and car crashes. As shown in 12, liquor selling points are represented as individual points or aggregated circles with numbers. Crash density is visualized using a heat map. It can be observed that there is generally a positive correlation between the number of liquor selling points and the density of car crashes in terms of their geographical distribution. It can be inferred that areas with a high concentration of liquor establishments also have a higher risk of car accidents. This could be attributed to a higher probability of drunk driving in areas with a high density of liquor establishments, thereby increasing the number of car crashes. Additionally, areas with a high density of liquor establishments often indicate a larger population, which may contribute to a higher occurrence of car accidents. For example, in the densely populated central areas of Victoria on the map, there are visibly more liquor establishments compared to sparsely populated peripheral areas, and the density of car crashes follows a similar pattern.

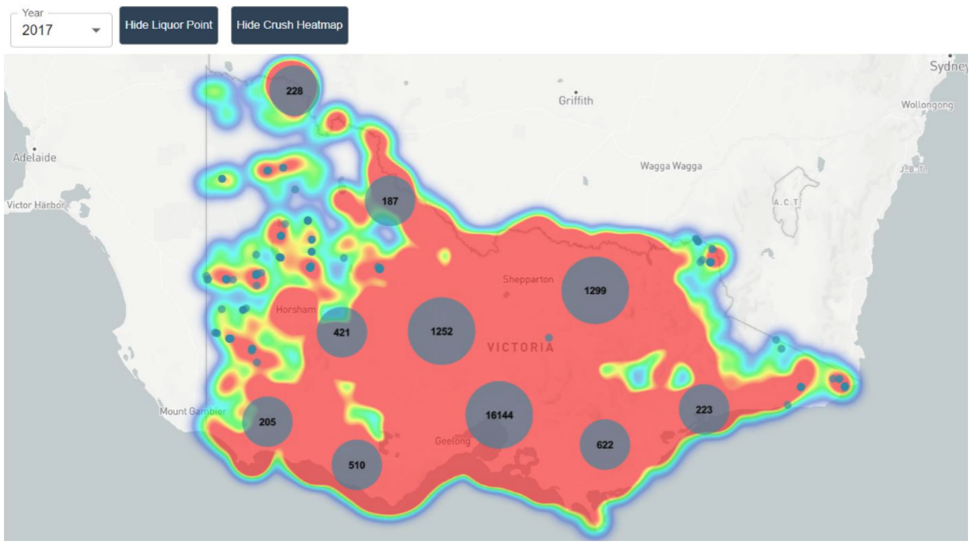


Figure 12: Liquor Selling and Crash Heat Map

The second chart in the SUDO page 13, is used to support this point. It illustrates the number of alcohol sales points, car accidents, and crimes in different regions from 2017 to 2019. It can be observed that Melbourne and Yarra are densely populated areas with a higher number of alcohol sales points compared to regions with smaller populations such as Baw Baw, Bayside, and Ballarat. These densely populated areas also experience a higher number of car accidents. However, the relationship between crime and liquor is not as pronounced. For example, although Yarra has a significant number of alcohol sales points, it experiences fewer crimes compared to Casey and Hume.

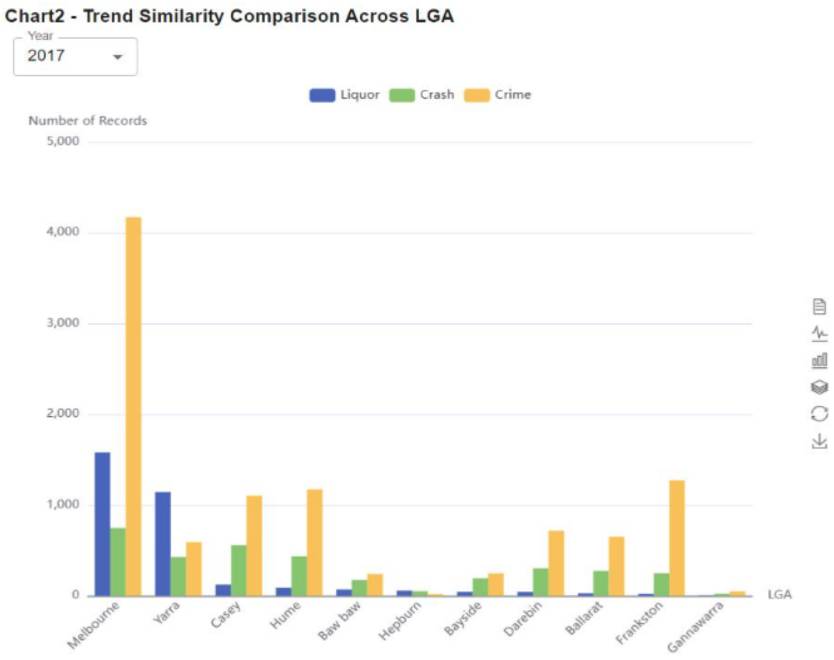


Figure 13: Liquor Selling, Crime and Crash Accidents

## 4.3 Scenario 2: How does crash incidents correlated to liquor vary in different time?

This scenario attempts to analyze the relationship between alcohol and car accidents from a temporal perspective. The analysis utilizes the Victoria car crash dataset from 2017-2019 obtained from the Victorian Bureau of Statistics. The dataset was filtered to include only alcohol-related car crashes, involving both drunk drivers and pedestrians. As Figure 14 to 17, a bar chart was employed to visualize the data, allowing for the selection of different years and days of the week using drop-down menus, and comparing the number of crashes within each 24-hour period across the seven days of the week. The severity of the crashes was differentiated on the bars using blue (Serious/Fatal) and green (other injuries) colors.

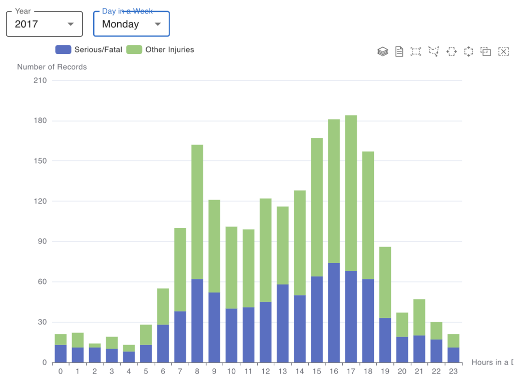


Figure 14: Weekday incidents

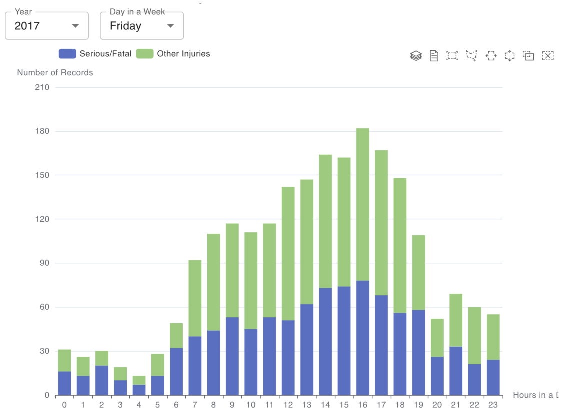


Figure 15: Weekday incidents

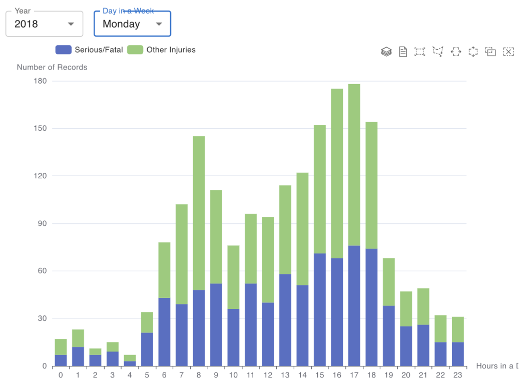


Figure 16: Weekday incidents

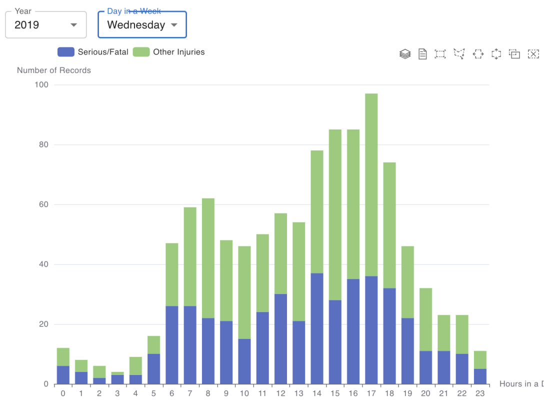


Figure 17: Weekday incidents

It is noteworthy that the observed pattern differs from our expectations, as the majority of alcohol-related car accidents occur after dinner hours in the evening. From Monday to Friday, the peak in alcohol-related car accidents is at around 7PM. However, on Saturdays and Sundays, the peak of alcohol-related accidents shifts to the time period between 11 AM and 2 PM as Figure 18 to 19. This trend remains consistent across the years 2017 to 2019. This might be attributed to the fact that the time periods of 8 AM and 3 PM to 5 PM on weekdays coincide with peak commuting hours, resulting in higher pedestrian traffic. Intoxicated pedestrians, whose unpredictable movements make it difficult for drivers to anticipate their actions, can contribute to the occurrence of accidents.

This observation is supported by our twitter data analysis as Figure 20b and 20c, from the Twitter user behavior for posting across 24 hours chart, it can be seen that people's attitude towards crime and crash peaked at 4PM as well. However, people more like to mention alcohol related topic at 12Noon and 4PM as Figure 20a. Our observation suggests that crashing incidents do happen more often at around 4PM, however it is not strongly correlated to alcohol.

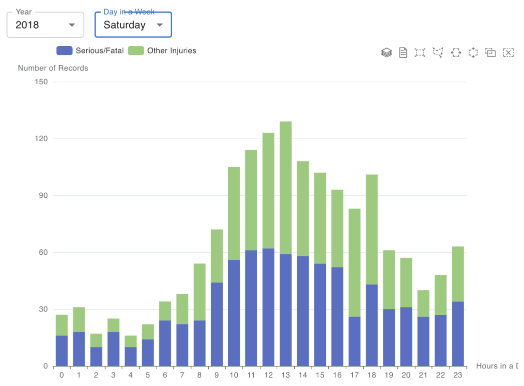


Figure 18: Weekend incidents

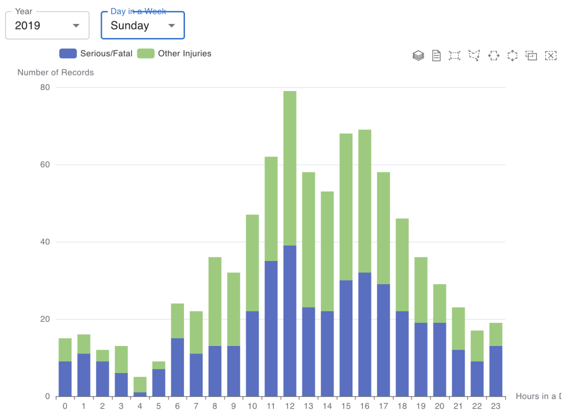
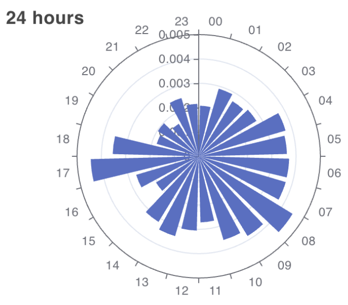
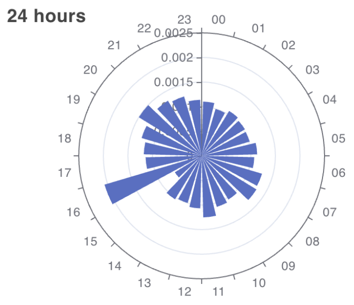


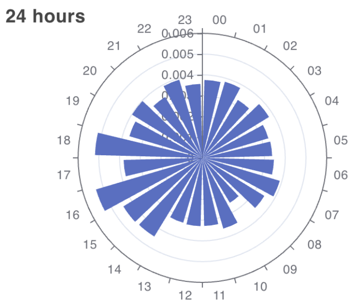
Figure 19: Weekend incidents



(a) Liquor



(b) Crime



(c) Crash

Figure 20: Twitter analysis over topics.

## 4.4 Scenario 3: How does public attitudes towards alcohol vary in a day?

This scenario aims to showcase the public's attitudes changes varies in a day by comparing both twitter and mastodon data. For twitter data, we sort sentiment analysis into three topics, and map the positive and negative tweets amount across 24 hours. From Figure 20 It shows that tweets related to alcohol have a higher proportion at 8 o'clock and 17 o'clock. We analyze that this may be related to the timing of alcohol-related social activities. Generally, social activities occur after work, and 17 o'clock is the time just after work when people may preview the upcoming activities. The high proportion of alcohol-related discussions at 8 o'clock in the morning is an unexpected finding. We speculate that this may be due to people waking up and posting discussions about alcohol-related social activities from the previous day.

For Figure 21, We visualized the results of 3 different statistical methods: total sentiment score, number of sentiment tweets, and average sentiment score for each hour of the 24-hour period. Ultimately, we found that the data displayed by the average sentiment score is more suitable for this scenario. The reason is that the total sentiment score can be influenced by the number of tweets posted, which, as shown in the previous Figure 20, is influenced by the time itself. This makes it difficult to distinguish whether the sentiment score for this particular time is due to the intensity of people's emotions or simply because more tweets were posted during that time. The number of sentiment tweets provides similar information as the previous chart.On the other hand, the average sentiment score can better analyze the intensity of people's emotions towards alcohol-related topics during this time period by eliminating the influence of tweet quantity. It allows us to focus on the emotional intensity rather than the volume of tweets. It can be observed that at 19:00, Twitter discussions related to alcohol showed a strong positive sentiment. We believe that this may be due to the social activity time during the evening when people gather with their family and friends for dining or gatherings. This positive social environment encourages people to express a more positive sentiment towards alcohol-related topics.The peak of negative sentiment related to alcohol occurs at

9 AM. We analyze that generally, 9 AM is the time when people wake up, and the hangover effect may contribute to the presence of negative emotions. Specifically, people may experience physical discomfort due to the hangover from the previous day. Alternatively, they may feel regret upon waking up for the negative consequences of their alcohol consumption.These information provides a differentiated statistical analysis of Twitter data based on positive and negative sentiment.

For Mastodon data, we collect and process the toots in real-time. Specifically,to achieve real-time analysis, every 5 seconds, we scrape the latest 100 toots related to liquor from the social, AU, and world servers. These toots are then categorized based on positive and negative sentiment. The sentiment words found in these toots are visualized in the form of a word cloud, where the size of each sentiment word corresponds to its frequency of occurrence as per Figure 22 and 23.

By doing this, we can cross compare public's attitude between Twitter and Mastodon in real-time, rather than looking at historical data.

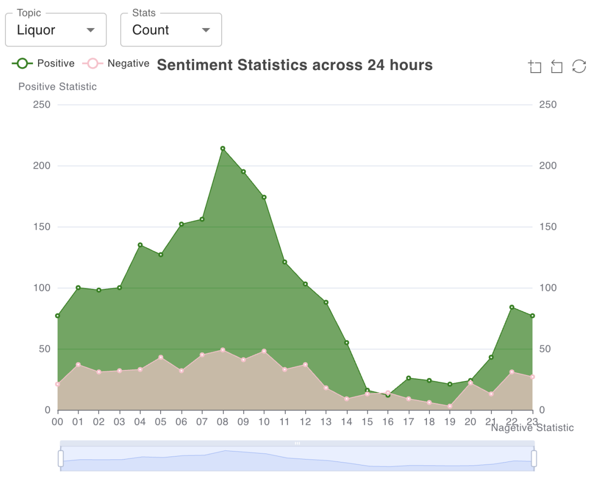


Figure 21: Twitter Sentiment

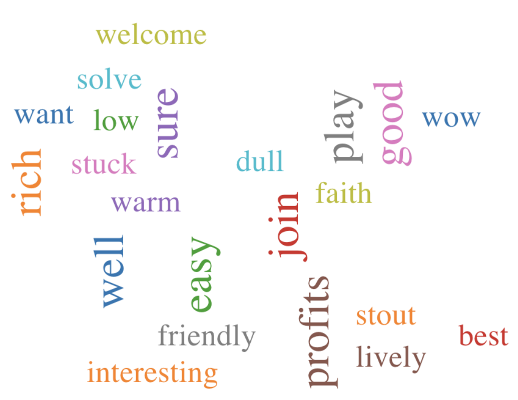


Figure 22: Mastodon Sentiment - positive

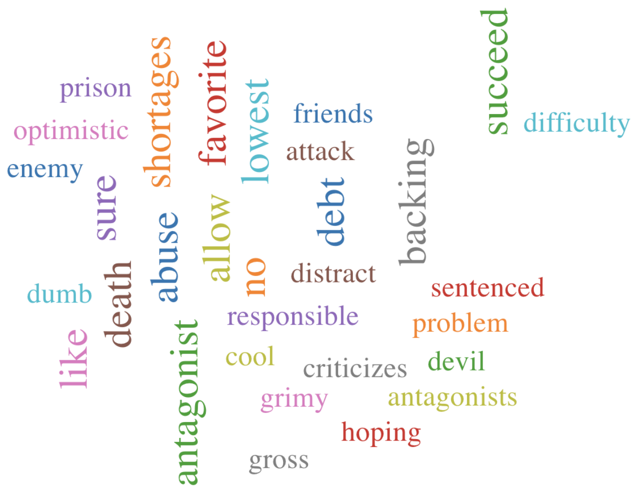


Figure 23: Mastodon Sentiment - negative

# 5 Challenges and Issues

## 5.1 Development using MRC

Our project primarily employed MRC for development, as most services were deployed on cloud instances. Although MRC offered us extensive accessibility and resources, aligning with the project's requirements, we encountered several challenges specific to MRC during development.

### 5.1.1 Unpredictability

Cloud services may sometimes exhibit unpredictable performance. Throughout development, we faced issues like unresponsive SSH to instances and slow network speeds. There were instances where we couldn't deploy our services using Ansible. These unforeseen uncertainties prompted us to debug our code, only to find that the server issues were to blame.

### 5.1.2 Complexity

Managing cloud resources is inherently complex, involving intricate aspects such as scaling, load balancing, data storage, and security. Our cloud-based applications heavily relied on internet connectivity. Any disruptions in network or internet services posed significant threats to our service stability. Therefore, we had to adopt a more thoughtful approach while developing our project to better accommodate the MRC development environment.

### 5.1.3 Resource Constraints

Every team received 8 cores and 500Gb volumes for development purposes. The allocation of 8 cores seemed restrictive considering the project's scope, which included services like a CouchDB cluster, multiple Mastodon crawlers, and front and back-end services. Furthermore, we needed to test automated deployment using Ansible playbooks. Considering a team size of five and the myriad tasks to accomplish, we believe a minimum of 10 cores per team would be more appropriate.

### 5.1.4 Scalability

While the MRC provides a significant amount of computing resources, it can still be a challenge to ensure that the system is designed to scale effectively. This involves planning for potential increases in data volume and user load, which could affect system performance and response times. A fully automated load balancing system (including resource provisioning) is difficult to be implemented in this project.

## 5.2 Data Collection

The project demanded data directly relevant to alcohol, crime, and accidents within the Melbourne region, which limited our data collection possibilities.

### 5.2.1 Twitter

From the provided 60GB Twitter dataset, we could only extract 600MB of data containing geolocation information. Among these, a mere 22MB of tweets related to alcohol, crime, and accidents. Thus, the usable Twitter data for our project was rather limited, requiring us to gather additional data from alternative sources.

#### 5.2.2 Mastodon

Mastodon toots provide less information compared to tweets, such as embedded sentiment analysis and geolocation. Consequently, cross-comparing toots with tweets regarding location was challenging. Moreover, each harvested toot required sentiment analysis.

#### 5.2.3 SUDO and other resources

The SUDO-provided data—liquor licenses, crime reports, and accident reports—were confined to the 2019 calendar year. More recent data is current not available.

### 5.3 Packages and APIs

#### 5.3.1 CouchDB Package

In our backend development, we employed the Python CouchDB package, enabling us to establish a connection between our backend service and database. To improve fault tolerance, we connected to all three CouchDB nodes, allowing us to fall back on the other two nodes if one connection failed. However, due to the default CouchDB connection timeout setting, we couldn't configure the timeout to a shorter duration, which would have expedited the fault tolerance behavior. As a result, the waiting period for connection was excessively long—about 2 minutes. As this involves thread-safety issues, we were compelled to retain the default setting.

#### 5.3.2 React

In React, props are a method of passing data from parent to child components and are read-only. This limitation implied that a component could only read the props it received and could not modify them. This aspect of React introduced additional challenges during frontend service development.

#### 5.3.3 Integration Issues

Integrating different services, such as Flask, React.js, CouchDB, Docker, and Mastodon, can bring its own set of challenges. These technologies might have their own specific configurations or requirements, which could lead to compatibility issues or complexities in getting them to work together seamlessly.

## 6 Conclusion and Future Improvements

In conclusion, this project has successfully implemented a web-based analytic tool to examine a few profound scenarios including public attitudes towards alcohol consumption, the correlation between alcohol selling points, and the rate of crime or crashes. The integration of various technologies and data sources has enabled us to deliver meaningful insights that could have significant impacts on urban planning, policy-making, and public awareness. Despite the challenges encountered during the project, the outcome has proven to be a functional and robust solution that fulfills our objectives. The use of cloud resources, distributed database, automated deployment, and reactive frontend has greatly improved the final outcomes.

However, the journey to optimize and improve our project does not end here. Looking forward, we plan to enhance the scalability and reliability of our services by incorporating Kubernetes for container orchestration. Kubernetes would provide more control over our deployed services, enabling more sophisticated scaling and automatic restarts for failed services, thereby increasing the resilience of our system. Alternatively, we could leverage Docker Swarm for better container management. Docker Swarm can help us manage a cluster of Docker nodes as a single virtual system, enabling easier scaling and providing more flexibility. It would allow us to maintain high availability of our services and make our deployment and scaling process even more efficient.

# 7 Project Management

## 7.1 Roles and Responsibilities

| Group Member | Roles | Responsibilities |
|---|---|---|
| Shaoyu Wang | Project Manager | Take responsibilities to ensure the interaction between Front-end and Back-end, including API design and implementation, data format transformation and Front-end visualization. |
| Taylor Tang | Backend Developer | Take responsibilities of backend development, including Back-end, CouchDB and Mastodon Crawler. Ansible deployment implementation. Overall system architecture designing. |
| Chenyuan Li | Data Analyst | Take responsibilities of data collection, data processing, database management and Map Reduce design and implementation. |
| Yannei Xu | Frontend Developer | Take responsibilities of building and implementing Frontend visualization idea and design components and layout for web pages. |
| Qiao Chen | Frontend Developer | Take responsibilities of building and implementing Frontend visualization idea and design components and layout for web pages. |

Table 1: Group Members, Roles and Responsibilities

## 7.2 Project governance

### 7.2.1 Communication

In this project, our team organized and worked in an Agile software development style, and we arranged our regular group meetings on a frequency of twice a week, which were typically held on Monday and Thursday evenings if there was not any time conflict with other scheduling. In terms of communication format, all meetings were held online via Zoom because there is one group member who is not based in Melbourne. Besides, we chose WeChat group as a communication channel for other group-based interactions such as daily standup, progress update or conflict resolving. Our communication plan was generally implemented well as scheduled throughout this whole project.

### 7.2.2 Task allocation

We made use of the Kanban board on the Trello platform to distribute individual tasks and track project progress, and we broke down the overall objectives into multiple subtasks and stages which spreaded across the project timeline. In terms of team role arrangement, we typically assign at least two team members for each technical stack such Front-end or Back-end, in which each team member can discuss with or seek help from the others to solve any problems encountered. In order to achieve code version management, we created a repository within a Github organization, and each developer worked on their own branch and made contributions on the whole project.

## 7.3 Individual contribution

Shaoyu Wang

- Data processing for Twitter data
- Backend functionalities testing
- Front-end data visualization for Twitter and Mastodon data
- CouchDB views implementation

Taylor Tang

- System architecture designing
- Backend development
- CouchDB implementation
- Mastodon crawler development
- Ansible deployment of the project

Chenyuan Li

- Data collection and processing
- Database management and CouchDB Map Reduce implementation

Yannei Xu

- Data collection from online resource
- Frontend visualization implementation

Qiao Chen

- Frontend UI design and framework establishment
- Frontend visualization implementation

## 8 Video Links

General overview of the MRC utilization
https://youtu.be/pQJ5kDbtCig

Frontend service scenarios and analytics Map, Twitter  Mastodon: https://youtu.be/hf0FpbN_wo8
SUDO: https://youtu.be/S6bDqL0qT6g

Backend service fault tolerance feature
https://www.youtube.com/watch?v=SB4674Mjc7U

CouchDB cluster demonstration
https://youtu.be/5Ivhj-KSMg4

CouchDB Cluster and MapReduce Demonstration
https://youtu.be/X0AkBCit8eY

Docker containerization and self-healing features
https://youtu.be/8eX2vrSrz1A

Ansible playbook dynamic multiple Mastodon crawler deployment
https://youtu.be/Vx2fgt98ORU

Ansible playbook full service (Frontend, backend, database and crawler) deployment
https://youtu.be/jOUa72gokDI

## 9 GitHub Link

https://github.com/COMP90024-2023-Group-55/cloud-computing-project