COMP90024 2024S1 GROUP 78 REPORT

Shiya Fu(1475643), Jionghao Song(1428049), Zhuoyang Liu(917183)
Gaoyuan Ou (1301025), Mingxin Li (1369941)

May 2024

Abstract

The frequency of extreme weather events has surged dramatically over the past two decades, with the United Nations reporting over 400 extreme weather warnings in 2023 alone. This project aims to develop a cloud-based solution for analyzing weather data, leveraging extensive datasets from sources such as Kaggle's Victoria shared database, the Mastodon Server API, and locally captured data. Our comprehensive database includes Melbourne's weather information, VIC Liquor Venue data, Melbourne Footfall statistics, car accident records, and Mastodon user mood data. By utilizing cloud computing and virtual machines, we efficiently analyze and store this vast amount of data. This document provides a user guide for our system, explains the core functionality of our deployed scripts, and details the architecture supporting our solution. We analyze and present graphical results for various scenarios, discuss the challenges faced and solutions developed, and showcase innovative approaches for future iterations. Additionally, we provide links to our GitHub repository,SLACK,JIRA and an explanatory YouTube video.

1 Architecture and Design

1.1 UniMelb Research Cloud

The Melbourne Research Cloud is a platform that provides on-demand computing resources to researchers at the University of Melbourne. As an Infrastructure-as-a-service (IaaS), it provides researchers with the access to scalable and flexible cloud computing resources without the need for major financial investments in physical hardware including storages, CPUs and GPUs. And it is designed for the use of data analysis and hosting.

In the following sections, we will be discussing some pros and cons of MRC along with how we would benefit from MRC.

1.1.1 Advantages

Scalability: We can dynamically scale up or scale down and adjust the resources allocation according to the project specifications and this optimises the resource utilisation and minimises the cost. So whether some big data is going to be processed or we want to deploy a service on the cloud, this scalability will ensure that we will always have the suitable and necessary resources to be able to perform the task efficiently and productively.

Cost-effectiveness: Another important advantage of MRC is cost-effectiveness since we can have free access to it. Basically, by leveraging MRC's cloud services, it eliminates the concerns of financial investments unlike other commercial cloud services like Amazon and Microsoft Azure, which allows us to focus more on the project development and also allows us other researchers to better manage their budget.

Accessibility: MRC can be easily accessed with an internet connection through SSH with unimelb's VPN. This eliminates the need to be geographically sitting on campus to get the access to MRC. So we are able to get access to MRC even at home and we can develop our system and our team can collaborate with each other across locations. MRC also offers a user-friendly web-based dashboard and command line tool to facilitate deployment, monitoring, resource allocation, etc.

Availability: MRC is live 24 hours, 7 days a week so this availability allows some long-duration processes to be able to run on MRC such as data processing, deep learning model training. Also it makes it possible for us or researchers to be able to access the platform even in the evening or on weekends.

Customizability: In MRC, we are able to create an instance and configure the instance (install necessary software, libraries and dependencies). We could also customise some settings such as networking settings, security settings etc. We could also capture a snapshot of the instance and convert it to an image so that we can directly use this image to launch a new instance next time if we want the same configuration.

1.1.2 Disadvantages

Steep Learning Curve: Even though there are lots of tutorials, support and documentation available, we still need to take some time and put in effort to be familiar with using MRC especially how to deploy and manage nodes, configure security rules and how to utilise resources more efficiently. For those who are new to MRC, the learning curve might be steep.

Variable Performance: Sometimes connection with MRC could be unstable or laggy since there are too many users using the shared resources so the speed or the computational process could fluctuate, which would decrease the efficiency of our project development.

Availability: It is possible that the MRC will experience downtime occasionally. So when this happens, all the services or applications that are deployed on it will become unavailable and we will not have access to them, which could potentially be a significant issue.

Having considered all these pros and cons of MRC, we had a more clear understanding of what MRC can really do and how it will bring benefits to us. We are also aware of the potential issues that we might encounter when using MRC.

Resource Allocation:

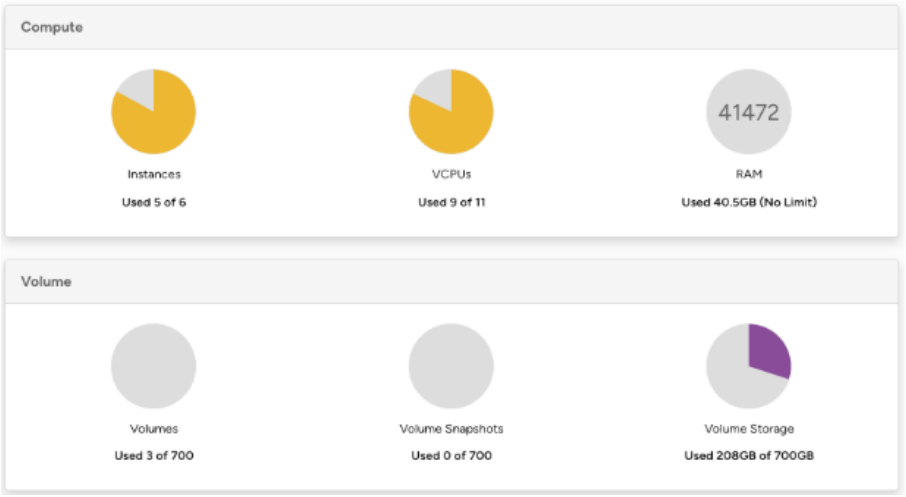


Figure 1: Resource Allocation

As illustrated in the graphs above, we have been allocated 6 instances with 11 virtual CPUs and 700 GB of volume storage. We will optimally utilise these resources to develop our system, ensuring it effectively supports the scenarios we have devised.

1.2 Front End

1.2.1 Framework

The front end of our system is based on the Jupyter Notebook. What the front end would do is some simple data processing and data visualisation. To get the data from ElasticSearch, the front end will make requests through the forwarded port (to Fission on the cluster) and will get responses in the format of JSON. So we will extract useful information in the response JSON for visualisation by data processing. Then we will visualise these data based on our scenarios

1.2.2 Data Fetching

- **Error Handling and Retries:** Our script includes robust error handling, which catches and manages timeouts, HTTP errors, and other request-related exceptions. We also implemented a retry

mechanism that allows up to five retries if a timeout occurs, enhancing the reliability of data fetching operations.

- **Debugging and Monitoring:** Our code outputs logs that inform about the progress of data fetching, including successful retrieval of batches and any issues encountered. This is useful for monitoring the process and debugging if issues arise.
- **Request Path Flexibility:** The script is designed to work with variable endpoint paths and has the base URL and specific path parameters, making it flexible to adapt to different data endpoints managed by the Fission function.

1.3 Back End

1.3.1 Overall application

Data Ingestion and Processing: Use Fission to create serverless functions that can ingest and process data from various sources. For example, you can have functions that fetch and process weather data, car crash reports, and social media posts.
Orchestration and Deployment: Use Kubernetes to manage the deployment of these serverless functions and ensure they scale based on demand. Kubernetes can handle the orchestration, ensuring your application is highly available and can handle varying workloads efficiently.
Data Indexing and Search: Use Elasticsearch to index the processed data, enabling powerful search and analytics capabilities. Elasticsearch can help you perform real-time searches and generate insights from the ingested data, such as analyzing trends in car crashes or monitoring social media activity.

1.3.2 Kubernetes (K8s)

Advantages

- **Community and Ecosystem:** Leverage a wide range of tools and integrations from the Kubernetes ecosystem to enhance your project's capabilities, such as using Helm for easy deployment of Elasticsearch or Prometheus for monitoring system performance[1].
- **Portability:** Ensures that your data processing applications can run consistently across different environments, whether you are developing locally or deploying on the Melbourne Research Cloud (MRC).
- **Scalability:** Automatically scales to handle the fluctuating volume of Mastodon posts, weather data, and car crash reports. For example, during peak social media activity or severe weather events, Kubernetes can scale resources to ensure smooth processing.

- **Flexibility:** Supports various workloads, such as batch processing of daily weather data and real-time sentiment analysis of Mastodon posts. This flexibility allows seamless integration of different data processing pipelines.
- **Automation:** Automates deployment and updates of data processing applications, ensuring that your system is always running the latest and most efficient version of your analysis algorithms.

Disadvantages

- **Debugging and Monitoring:** Monitoring and debugging a distributed system handling multiple data sources (e.g., weather, Mastodon, car crashes) can be challenging without advanced tools and techniques.
- **Security Management:** Ensuring secure communication between microservices handling sensitive data like car crash locations and social media posts requires careful configuration and management[1].
- **Operational Overhead:** Regular maintenance and upgrades of Kubernetes clusters introduce additional operational overhead, which can be demanding for a project processing dynamic and diverse datasets.

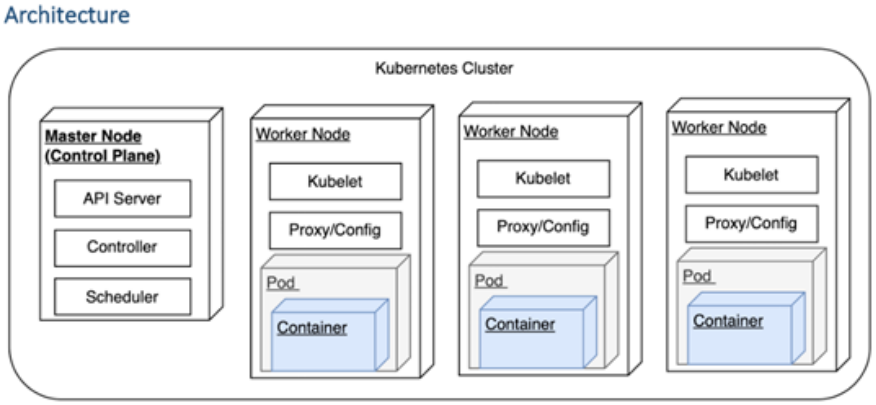


Figure 2: Kubernetes Cluster Architecture

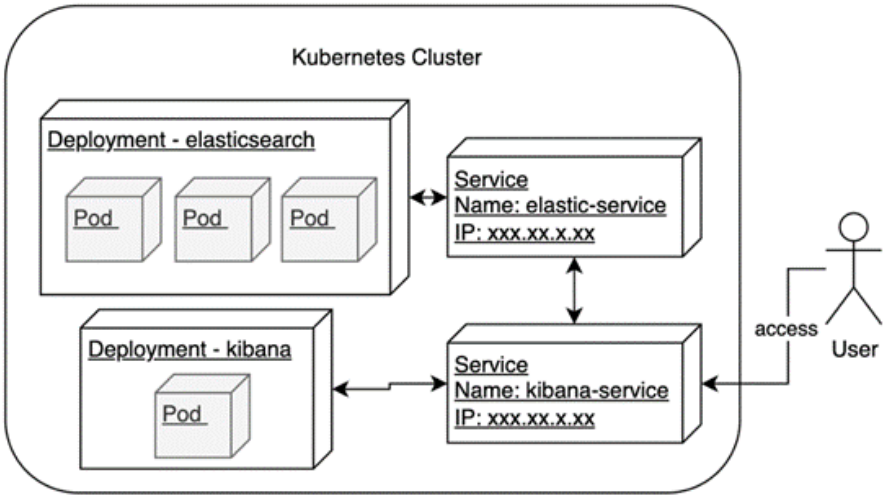


Figure 3: Kubernetes Interact with user

1.3.3 Fission

Advantages

- **Serverless Architecture:** Simplifies development by allowing you to focus on writing functions for specific tasks like weather data processing, sentiment analysis, or foot traffic estimation without managing servers.
- **Quick Deployment:** Allows rapid deployment and scaling of functions for real-time analysis of weather data and social media sentiment, ensuring timely insights.
- **Integration with Kubernetes:** Leverages Kubernetes' orchestration capabilities for managing serverless functions, ensuring they scale efficiently based on the volume of incoming data from Mastodon, weather services, and traffic reports.
- **Event-Driven:** Easily integrates with various event sources, enabling functions to trigger on new weather data, car crash reports, or social media posts.

Disadvantages

- **Limited Runtime Options:** Compared to other serverless platforms, Fission may have fewer runtime options, potentially limiting the choice of tools and libraries for processing complex datasets.
- **Complex Debugging:** Debugging serverless functions handling diverse and dynamic data sources (e.g., weather, Mastodon posts) can be more challenging due to their stateless nature.
- **Vendor Lock-In:** Depending on specific features of Fission, migrating to another serverless platform might be challenging if the need arises.

1.3.4 Elasticsearch

Advantages

- **Powerful Search Capabilities:** Enables efficient searching and filtering of large datasets, such as querying historical weather data or searching for specific sentiments in Mastodon posts related to weather conditions[2].
- **Real-Time Data Processing:** Capable of real-time indexing and querying, making it ideal for time-sensitive data like real-time weather updates and live social media sentiment analysis.
- **Scalability:** Scales horizontally to handle increasing volumes of data, such as growing datasets of car crash reports and social media posts.
- **Schema-Free:** Supports flexible, schema-free JSON data models, making it easy to index and search complex, nested data structures like detailed weather reports and social media content.

Disadvantages

- **Resource Intensive:** High resource usage for indexing and searching large datasets, such as extensive weather records and numerous social media posts, can lead to increased operational costs[2].
- **Complex Management:** Managing Elasticsearch clusters, including data sharding and ensuring data consistency, can be complex, especially when dealing with large and diverse datasets.
- **Potential for Data Loss:** Misconfigured clusters might lead to data loss, which is critical when analyzing sensitive data like car crash locations and weather conditions.

2 System Functionality

2.1 Scenario 1: How the weather affects people

Scenario Description: This scenario utilises daily weather data, pedestrian flow data, and sentiment analysis scores in Melbourne to explore their relationship. By analysing these data, we aim to investigate the impact of weather conditions on the emotions of city residents and pedestrian flow.

Why We Choose It: The purpose of this scenario is to understand how people's emotions vary under different weather conditions and how these emotional changes influence their behaviour, such as fluctuations in pedestrian flow. This helps city managers better understand the emotional fluctuations of citizens and implement corresponding policy measures according to different weather conditions to improve the quality of life for urban residents.

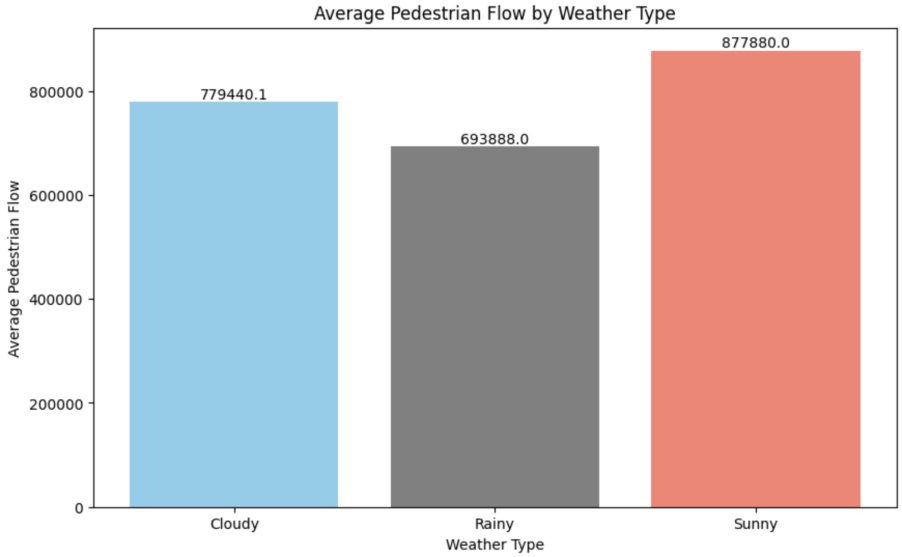


Figure 4: Pedestrian Flow by Weather Type

We categorised Melbourne's weather into three different types: sunny, rainy, and cloudy, and then observed the changes in pedestrian flow under these three weather conditions. From Figure 4, we found that pedestrian flow in Melbourne reached its peak during sunny weather, with 877,880 people, while it was lowest during rainy weather, with only 693,888 people. This indicates that sunny weather may encourage more people to go outdoors, while rainy weather may suppress people's outdoor activities. Such observations are crucial for city managers because they can adjust the allocation of urban resources based on weather forecasts, such as planning public transportation services and outdoor activities, to better meet the needs of citizens.

In addition, by analysing data from social media, we found that the temperature of the weather also had an impact on people's mood.

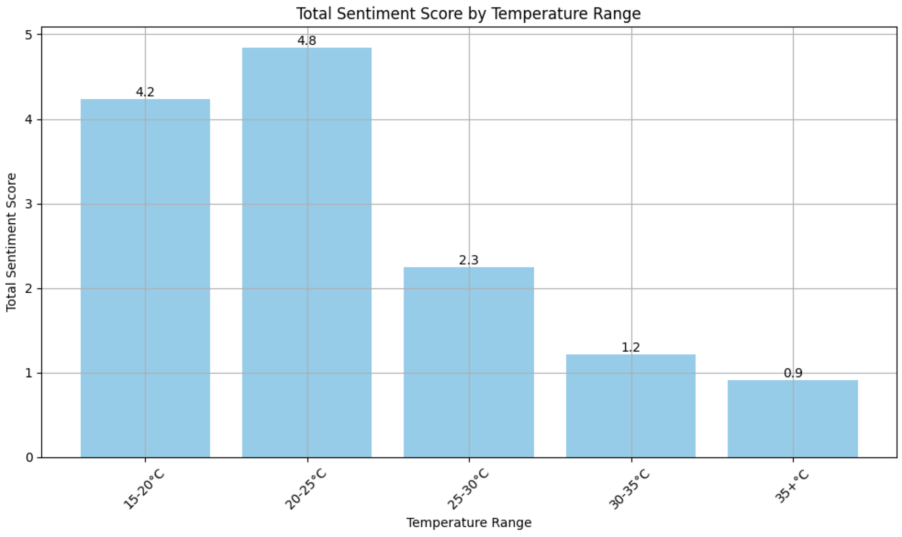


Figure 5: Sentiment Score affected by Temperature

FIgure 5 illustrates the overall sentiment scores within different temperature ranges, revealing a gradual decline in sentiment scores as temperatures rise. Within the temperature ranges of 15-20°C and 20-25°C, sentiment scores are relatively high at 4.2 and 4.8, respectively, indicating that people tend to express more positive emotions in these comfortable temperature ranges. However, as temperatures increase to 25-30°C, sentiment scores drop to 2.3, and further decline to 1.2 at 30-35°C. When temperatures exceed 35°C, sentiment scores reach their lowest point at 0.9. This suggests that higher temperatures significantly impact people's emotional states, possibly due to increased discomfort associated with high temperatures. Therefore, it can be inferred that the temperature range of 20-25°C is where people feel most comfortable and express the most positive emotions, while temperatures above 30°C have a notably negative effect on sentiment.

Based on the data presented, it's evident that people's moods fluctuate with weather conditions and temperature. Hence, we further analysed the sentiment data for the years 2023-2024.

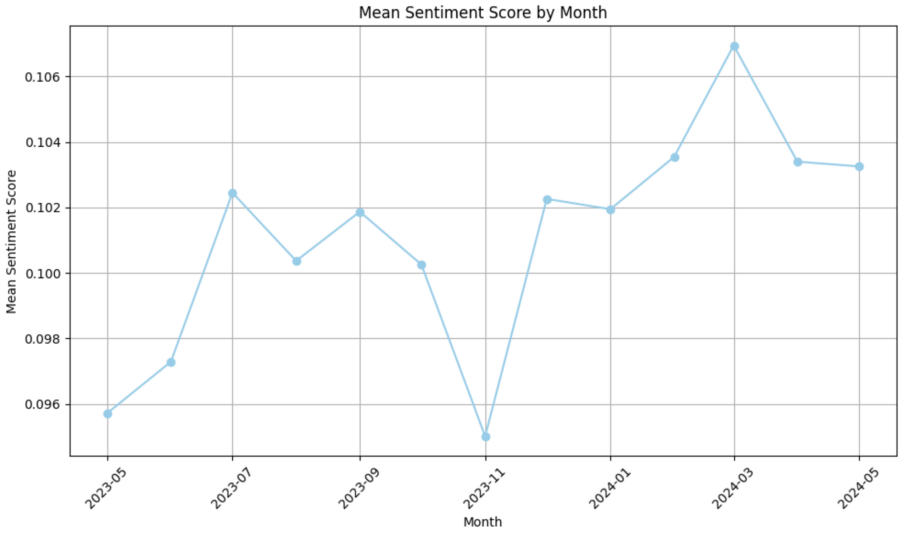


Figure 6: Sentiment Score Monthly

From Figure 6, it is evident that people's moods were the most positive in March 2024, suggesting a potential correlation with temperature. Therefore, we proceeded to examine the weather temperatures for March 2024, as shown in Figure 7.

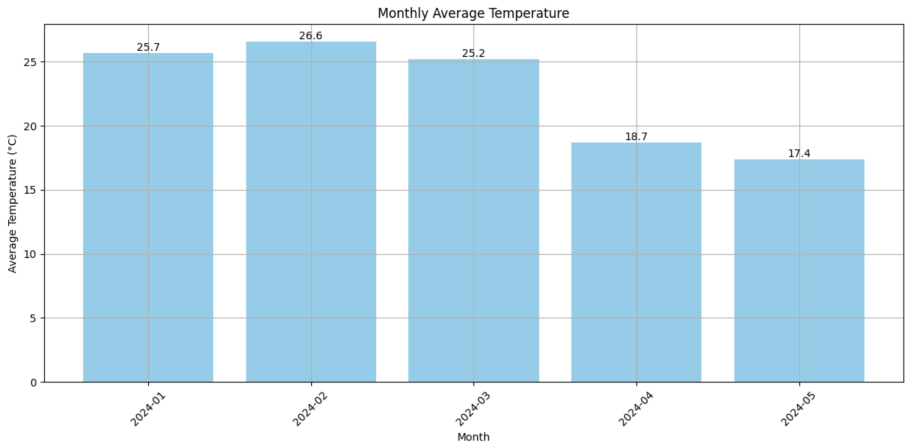


Figure 7: Average Temperature Monthly

Here, we observe that the average maximum temperature in March 2024 was 25.2 degrees Celsius, which falls within the range where people feel most comfortable and express the most positive emotions. Consequently, the mood index of individuals was higher during this period. However, as temperatures decreased to 17-18 degrees Celsius in April and May, indicating cooler weather, the mood index of people also declined. Thus, we have observed a significant impact of weather on people's emotions and behaviour. Our analysis not only helps in understanding the relationship between weather, temperature, and human emotions but also provides guidance for city managers. For instance, in urban planning and management, it is essential to consider weather factors, especially during periods of high temperatures, and take appropriate measures to mitigate the negative impact on people's emotions. This can contribute to better urban resource allocation and an improved quality of life for residents. Therefore, these research findings are crucial for optimising urban development and enhancing the well-being of citizens.

2.2 Scenario 2:Impact of alcohol premises activities on traffic safety

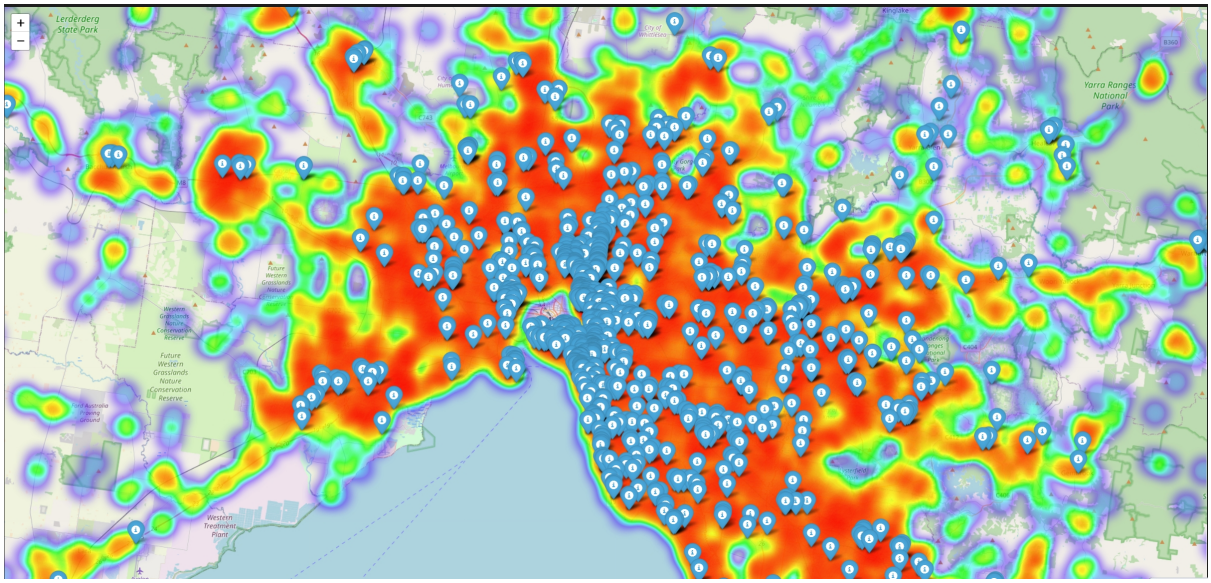


Figure 8: alcohol premises activities on traffic safety

Scenario Description: In this scenario, we aim to explore the relationship between activities at alcohol premises and traffic safety in Melbourne. We will utilise data on venues allowing on-premise alcohol consumption like pubs or nightclubs, and car crash incidents to investigate potential correlations between the two factors. By analysing this data, we seek to understand whether there is a link between alcohol-related activities and the occurrence of car crashes, and if so, to what extent.

Why We Choose It: This scenario was chosen due to its significant implications for public safety and urban planning. Understanding the impact of alcohol premises activities on traffic safety is crucial for city authorities to implement effective measures to reduce the risk of accidents and enhance road safety. By examining the relationship between alcohol-related activities and car crash incidents, we can provide valuable insights for policymakers and law enforcement agencies to develop targeted interventions and policies aimed at reducing the incidence of alcohol-related accidents on the roads.

Additionally, this scenario aligns with broader societal concerns regarding alcohol consumption and its potential negative consequences, particularly in relation to road safety. By shedding light on the correlation between alcohol premises activities and car crash incidents, we can contribute to ongoing efforts to promote responsible alcohol consumption and mitigate the risks associated with impaired driving.

Overall, this scenario was chosen for its potential to generate actionable insights that can inform evidence-based policy decisions aimed at improving public safety and reducing the prevalence of alcohol-related traffic accidents in urban areas.

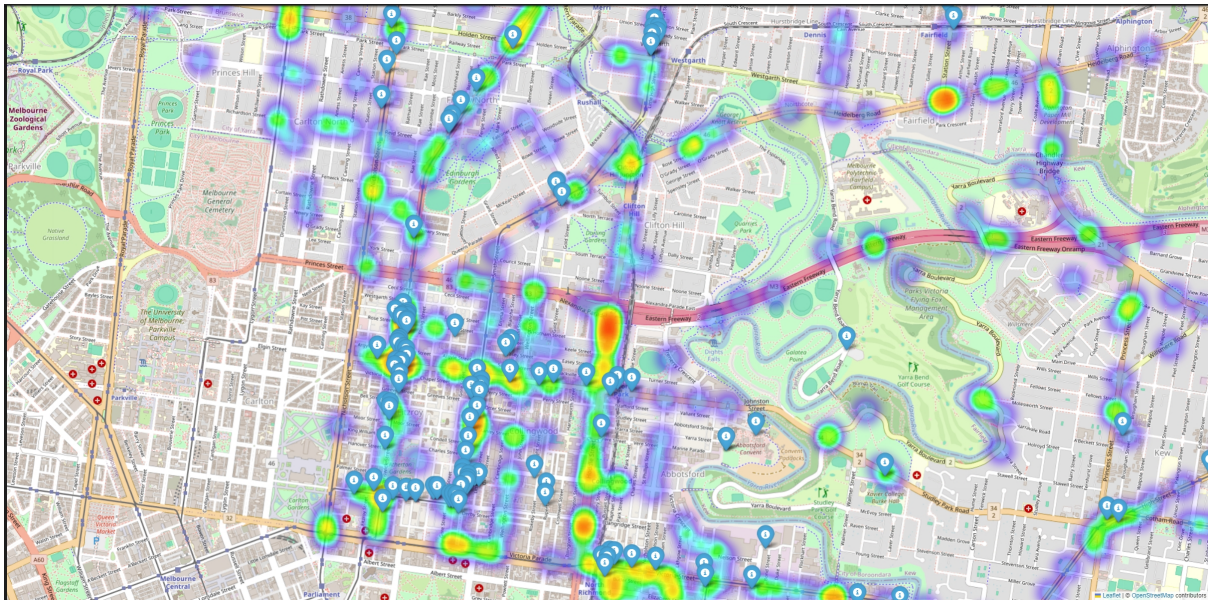


Figure 9: alcohol premises activities on traffic safety

In the map above, which shows the area of collingwood and Fitzroy, we can see some high density clusters which are in yellow and red. They are more likely to be surrounded by pubs or nightclubs. And this pattern could be a possible indication that car incidents are more likely to happen near alcohol premises like pubs and nightclubs.

Figure 10: Map in South yarra

The second map in South yarra, is also suggesting the same pattern we have found in the previous map. To explain the potential correlation between the location of alcohol premises and the occurrence of car incidents, we think there should be several reasons for this:

- Pubs and nightclubs are the venues where alcohol consumption is high. Alcohol impairs people's reaction times and driving skills. There could be a higher possibility that people may drive after alcohol consumption, which would also increase the risk of accidents.
- Pubs and nightclubs could attract more traffic and crowds than normal at night. So this increased traffic volume could be another factor for higher possibility of accidents.
- After alcohol consumption, people are less aware of the surroundings and are more likely to jaywalk without checking the traffic. So that would also increase the risk of vehicle-pedestrian collisions.

To decrease the possibility of car accidents near on-premise alcohol venues, we think there could be some actions that the government can take:

- The government could deploy more police force in these areas, especially during late hours.
- Set up sobriety checkpoints near these venues to catch drunk drivers.
- Set well-marked and well-lit pedestrian crossings near these venues and lower the speed limit in these areas during late (alcohol) hours.
- Initiate campaigns to make the public aware of the dangers of drunk driving or driving after alcohol consumption.

3 Data Processing

3.1 Data Collection

Data Integration and Processing

This research project integrates multiple data sources to investigate the interrelations between weather conditions, social media sentiments, road accidents, and pedestrian traffic in Victoria, Australia. The details of the data sources used are as follows:

Weather Data

- **Source**: This dataset encompasses daily meteorological records from Melbourne, including temperature, humidity, and wind speed.

- **Application**: This data is pivotal for analyzing the impact of weather conditions on social media sentiments, frequency of road accidents, and variations in pedestrian traffic. It allows us to establish patterns and correlations, providing insights into how weather influences human behavior and safety metrics.

Road Accident Data

- **Source**: The dataset contains detailed records of road accidents in Victoria, including specifics about locations, timings, and conditions of accidents.
- **Application**: The analysis of this data helps in understanding the correlation between adverse weather conditions and the incidence of road accidents. Furthermore, spatial analysis techniques are used to explore the proximity of liquor shops to accident sites, potentially identifying risk factors contributing to accidents.

Pedestrian Traffic Data

- **Source**: Data from the Melbourne Pedestrian Counting System provides hourly, monthly counts of pedestrian flow.
- **Application**: This data is crucial for examining how pedestrian volumes fluctuate under different weather conditions. The comparison between clear and cloudy/rainy days offers valuable insights into how weather can influence urban mobility patterns.

Social Media Data (Mastodon)

- **Source**: Data has been collected from the Mastodon platform, focusing on posts from users across Australia.
- **Application**: Sentiment analysis tools are employed to assign sentiment scores to the posts, facilitating a study on the relationship between public mood and weather conditions. This will help determine if weather variations significantly affect people's emotions as expressed through social media.

3.2 Data Processing Pipeline for Mastodon Data involves

Setting Up the Servers and Tokens

The process begins by configuring access to popular Mastodon servers and retrieving necessary authentication tokens:

- **Server Configuration**: The servers ”mastodon.social” and ”aus.social” are targeted for data collection due to their high user activity.
- **Token Retrieval**: Access tokens required for authenticating API requests to these servers are stored in a JSON file named `token.json`. The script reads this file and loads the tokens into a dictionary for easy access.

Fetching Data from Mastodon Servers

Data collection from Mastodon servers is handled by a multi-threaded approach:

- **User Agents**: A list of different user agents is defined to rotate through and avoid detection and blocking by the servers for making repetitive requests.
- **Concurrent Fetching**: Separate threads are initiated for each server. Each thread sends requests to the server's API to fetch user posts over the past year. These requests are authenticated using the tokens loaded in the previous step. The data collected includes post ID, creation date, language (only English posts included), and content (split into tokens). Each server's data is saved into an output file for further processing.

This concurrent approach ensures efficient data collection by utilizing multiple threads to fetch data simultaneously from different servers.

Handling API Requests and Data Extraction

API requests and data extraction are managed by a dedicated script that performs the following tasks:

- **Asynchronous Requests**: The script uses asynchronous requests to fetch data from the Mastodon servers. This ensures that multiple requests can be sent and processed concurrently, reducing the overall time taken for data collection.
- **Data Extraction**: Once the data is fetched, it is processed to extract relevant information. HTML tags are removed from the post content, and sentiment analysis is performed using the TextBlob library. The sentiment analysis assigns a polarity score to each post, indicating whether the sentiment is positive, negative, or neutral. Additionally, keywords (tokens) and tags are extracted from the posts.

For example, a post like ”trimmed repotting ancient tree) I've snip like needed has (an biggest moss pot.” is processed to extract keywords and tags, and it received a sentiment score of 0.0966, indicating a slightly positive sentiment.

Filtering Data

The initial filtering process involves reading the collected data and excluding posts with a neutral sentiment score (sentiment = 0). This step ensures that only posts with a clear positive or negative sentiment are retained. For instance, a post with the sentiment score of 0.0966 was included, while a post with a sentiment score of 0 would be excluded.

Extracting Relevant Information

Further filtering is applied to focus on posts related to Melbourne. The script checks each post for keywords associated with Melbourne in the extracted tokens and tags. If a post contains any Melbourne-related keyword, it is retained; otherwise, it is discarded. This process ensures that the final dataset is geographically relevant to the project's scope. For example, if a post with a sentiment score of 0.4000 contains keywords like "Melbourne" or "Victoria," it is included in the final dataset.

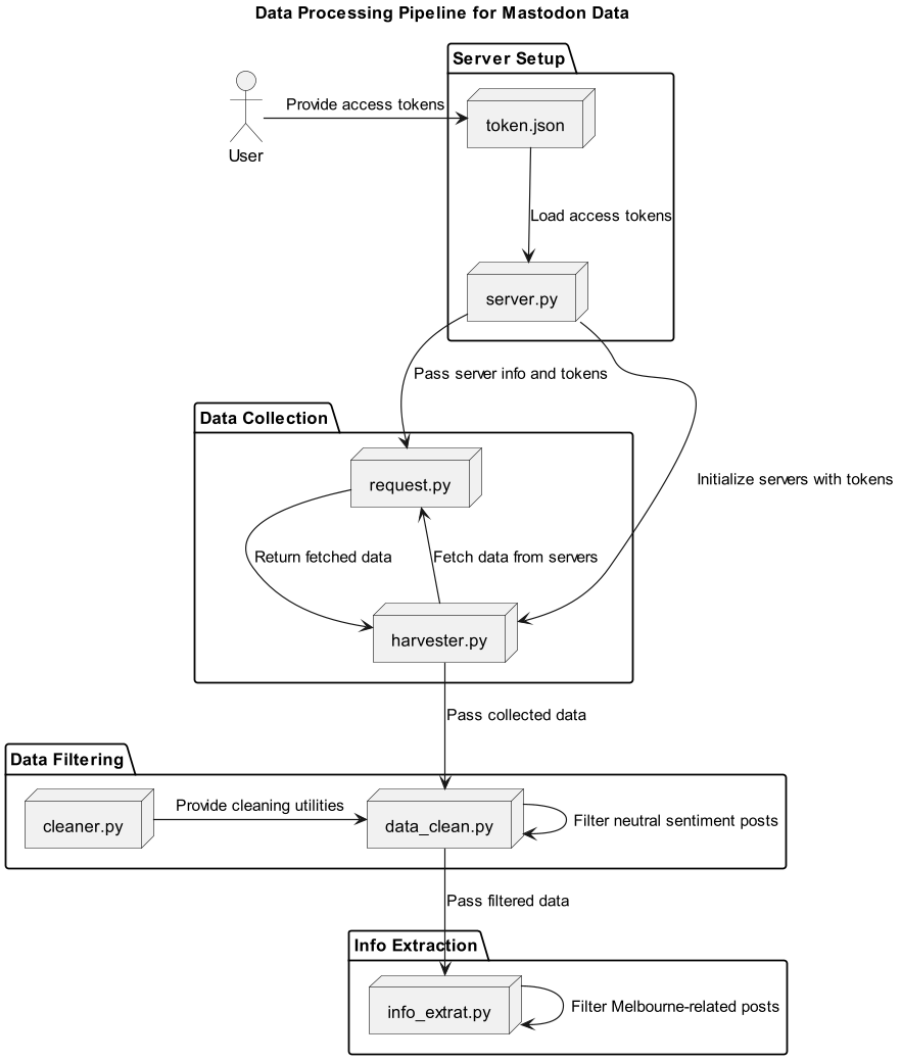


Figure 11: The structure of Mastodon

3.3 Data Retrivial

3.3.1 API Integration Using Fission

To facilitate data accessibility and interaction between the backend and frontend components of our project, we have implemented RESTful APIs using Fission, a serverless framework. These APIs provide a standardized way for the frontend to retrieve and display data dynamically from various sources. The APIs include endpoints for accessing liquor store locations (‘GET /liquor/data‘), road accident details (‘GET /accidents/data‘), sentiment-analyzed Mastodon posts (‘GET /mastodon/data‘), pedestrian traffic data (‘GET /sensors/data‘), and daily weather metrics (‘GET /weather/data‘). An example command to access sensor data is ‘curl ”http://127.0.0.1:9090/sensors/data” — jq ’.’‘, illustrating how frontends can fetch real-time data efficiently, thus enhancing the system’s responsiveness and interactivity[3].

4 Challenges and Handling

4.1 Environment Configuration

4.1.1 Local Environment Configuration

Firstly, as we chose to use Kubernetes (k8s) as the framework for our deployment, which is based on the Linux operating system, the team members needed to build their own local development environments according to their needs. However, as WSL and Ubuntu are currently only supported in environments that have a Virtual Machine (VM) upgraded to the latest Windows 11-22h2, some team members had to travel to the University of Melbourne campus to use contributing computers on campus. However, as the Melbourne Research Cloud requires VPN access to the campus intranet, those team members who have successfully deployed their development environments still need to establish complex VPN connections each time they debug.

4.1.2 Fission

While using Fission as our backend service, we encountered some challenges but also found corresponding solutions. Firstly, we faced difficulties in accessing the correct URL for Elasticsearch-master. After several attempts, we finally determined the correct method to access it. This experience taught us the importance of detailed configuration documentation and accurate environment settings during deployment and setup. Secondly, another issue was the insufficient number of results returned by Fission’s API when handling large volumes of data. To effectively manage this data, we implemented the scroll-id mechanism, which allowed us to record the position of the last API call and continue outputting data from that point. This method not only optimized our data processing workflow but also ensured data

integrity and access efficiency.

4.1.3 Testing

We edited the backend code with a unit test using Python's own unittest package, which will test the smallest testable unit of the backend server by using `mock_client` to build a mock backend server and sending API requests to it to see if these operations on the backend server are successful. However, when integrating these tests with GitHub Actions, the containers that come with GitHub Actions don't have a runtime environment and will throw errors when installing the necessary dependencies. This defeats the purpose of using the GitHub Actions pipeline for CI/CD.

4.2 Single Point of Failure

In the real world, servers tend to crash for various reasons and such crashes can happen at any time. To avoid such crashes that cause the backend server to not work properly, we try to ensure data integrity during data deposit by cleansing the data and using shards to slice all the documents in a particular index and store them separately on different nodes. This not only allows simultaneous queries on different nodes, thereby improving query performance, but also enhances the scalability of the database. When a request is passed in, the shard will query the data in its own node based on this request. The stability of the database is improved with the addition of new nodes. When one of the nodes is not working properly, the remaining nodes will take over to support the incoming request. If a node is shown in green, it means that it does not have the same shard, which maximizes the security and scalability of the architecture. Storing two identical shards in the same node degrades search performance. Subsequently, replicas, which are copies of the shards, are also used, improving redundancy and disaster recovery.

4.2.1 Index Life-cycle

We are also trying to set the life-cycle for the data in Elasticsearch. The life-cycle policy will be set when the index is created. When an index is in an active state, i.e., frequently updated and looked up, it will be marked as Hot and stored in a high-performance node for possible future operations. Over time, some indexes may no longer be updated frequently but still have occasional query requests; the index will be marked as Warm and these indexes will be removed from the high-performance node and added to the lower-performance node. When the data is no longer frequently queried or updated, it will enter the Cold phase, and the index will be moved to a lower-cost node to be stored in case it is needed. Finally, the index is moved to the Frozen phase, where the data is considered fully archived and is stored on static media. If there are no lookups or rewrites for a certain period of time, then it will be moved

to the Delete phase, where the data in the index will be completely deleted. This state transfer not only optimizes our resource management but also makes sense for valuable server performance.

4.3 Backend

4.3.1 Initial Non-compliance with RESTful API

Firstly, we use Flask with RESTful API to provide assistance for our front and back end communication. Therefore, the parameters of the GET request will change in real time depending on the scenario. However, because of this, the parameters of the GET request will be different in different scenarios. Initially, we didn't take this into account, which directly led to the system crashing due to incomplete request parameters when developing new scenarios. To solve this problem, we tried to set the parameters in the request URL, and those undefined GET requests will not be processed by the backend server. Finally, due to the use of Python exception handling (try-catch), the backend server can still work when faced with a request packet with unknown parameters. We also experimented with the technique of integrating different URLs into a single function, which is similar to the design principle of regular expressions. This improves the efficiency of the server by processing multiple requests at the same time. By looking at the technology stack, we found that a route can match one or more URLs, and multiple routes can trigger the same function due to different incoming parameters, which undoubtedly brings a new perspective to our backend development.

4.3.2 Index Deleting

It is worth noting that when using the Unimelb Research Cloud, we have experienced situations where we try to fetch an index, and the target index is deleted. We have tried to investigate this issue by saving and analyzing the corresponding logs, but so far we have found nothing, making this an open question.

Figure 12: Error When Idex Missing

Figure 13: Front End Connection Error

4.3.3 Time Out

In the process of fetching remote data, we experienced connection timeouts. The strange thing is that when two team members do the fetch operation at the same time, one of them will get the error message of connection timeout, while the other team member will succeed in fetching the corresponding data. By examining the backend and looking at the Elasticsearch documentation, we solved the problem by adjusting the timeout limit.

```
import fetch

data = fetch.fetch_all_data("http://127.0.0.1:9090", '/weather5k-fetch')
✓ 46.1s

Error fetching data: {"error":"Connection timed out"}
```

Figure 14: Front End Connection Timeout

```
cachsia@DESKTOP-QKE9EDM:~/pythonProject8/fission$ python3 fetch.py
5000 records get.
5000 records get.
5000 records get.
5000 records get.
5000 records get.
5000 records get.
5000 records get.
5000 records get.
5000 records get.
5000 records get.
5000 records get.
5000 records get.
5000 records get.
5000 records get.
5000 records get.
5000 records get.
5000 records get.
5000 records get.
5000 records get.
5000 records get.
5000 records get.
5000 records get.
5000 records get.
5000 records get.
No more results found
Retrieved 112987 records
```

Figure 15: Data Fetching Log

```
response = requests.get(url, params=params, timeout=(5, 30))
```

Figure 16: Set Timeout

4.3.4 GitHub Issue

It's worth noting that our development team has a good code development habit, whenever they encounter a tricky issue or bug, they will share the issue in GitHub Issues to other members, and get new

ideas through discussion with other members.

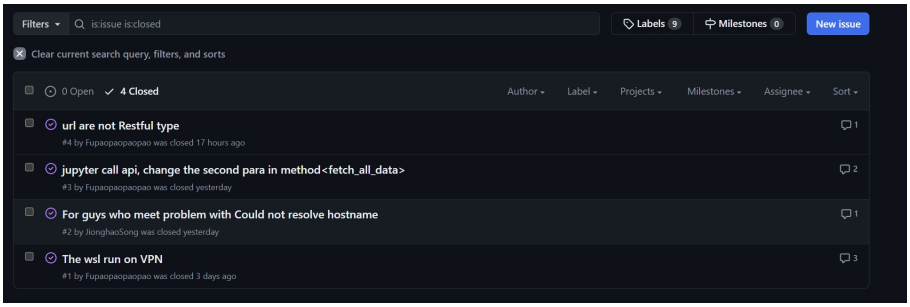


Figure 17: Github Issues

5 Evaluation and Further Development

5.1 Sentiment Analysis

Extracting a user's current emotional state from text is undoubtedly a challenging task, and these datasets harvested through various mining means often contain noisy information. Therefore, we try to use what we have learned in another course, NLP, by using the NLTK package to perform data cleaning, sentence segmentation, and word tokenization operations. These operations will remove as much noise as possible from the raw data to ensure that we have a relatively accurate dataset to support a relatively accurate result from the sentiment analysis.

5.2 Crawler

In the design phase of the project, we meticulously considered various potential error scenarios, such as exceeding the maximum number of retry attempts and failure to retrieve the `max_id` for each dataset, ensuring robustness in our data collection approach. However, after running the program continuously for two days, an unexpected issue arose when a server response deviated from the standard JSON format, leading to data misalignment in our scraping process. To address this, we implemented an additional try-catch block to handle anomalies in server responses more gracefully.

Furthermore, the data retrieved from Mastodon was in HTML rich text format, which posed a challenge for text analysis. To convert this data into plain text, which is more suitable for processing, we utilized BeautifulSoup. This tool enabled us to effectively strip HTML tags and extract clean text, thereby ensuring the integrity and usability of the data for sentiment analysis.

5.3 Creativity

5.3.1 CI/CD

We tried to use the Action pipeline that comes with GitHub for continuous integration during development. Here are the reasons why we decided to make this move. Firstly, Continuous Integration automates the running of system tests, which allows our project to enjoy the benefits of dynamic deployment from automation, and secondly since action's pipeline will be initiated every time we push to a remote repository using git, it will reduce the time spent reviewing our code during each project iteration, which in turn will increase the efficiency of our team's operations. It was also interesting to see that we tried to plug our Slack bots into the pipeline to post messages to our Slack channel after each continuous integration operation via API calls, but unfortunately this wasn't fully implemented due to the limitations of Slack. Eventually, team members can see the results of the test in the GitHub interface.

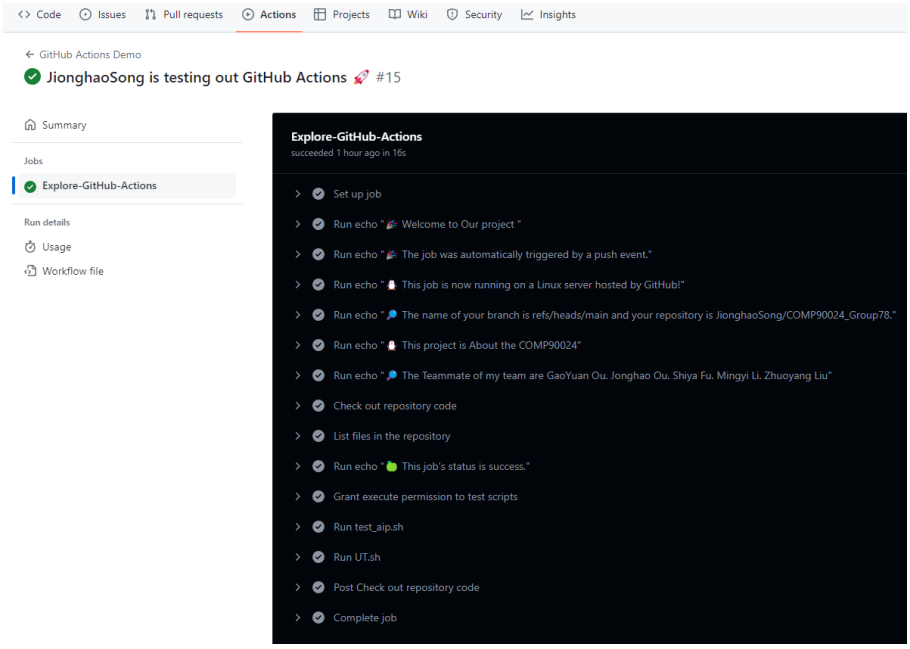


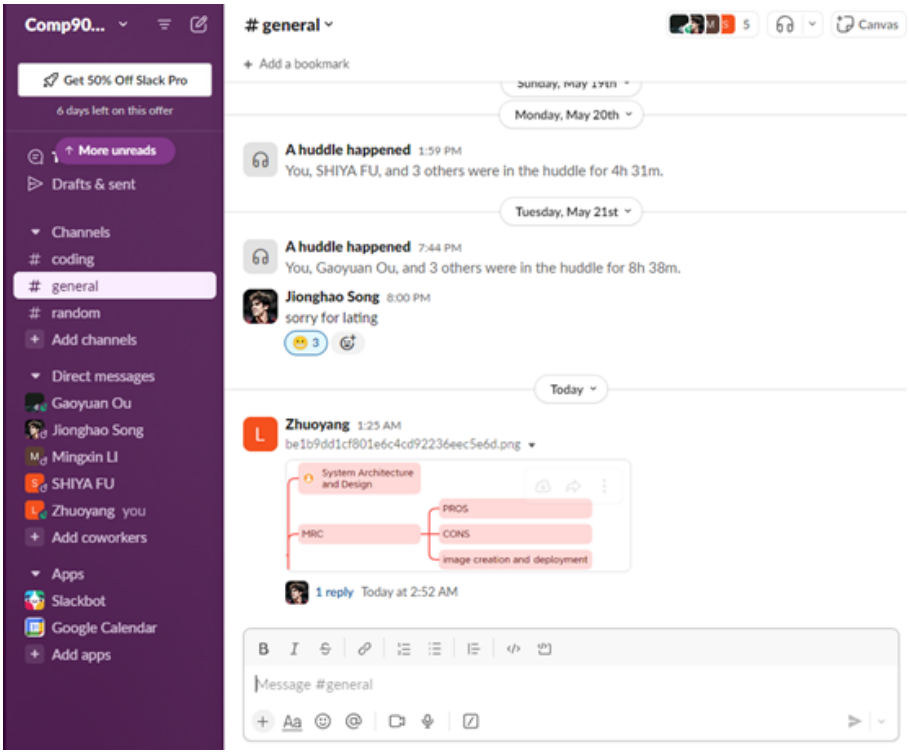
Figure 18: GitHub Action pipeline as CI/CD container

5.3.2 Diagram Drawing

For generating the corresponding system architecture and sequence diagrams (SSDs), we employ Lucidchart, an online visual drawing tool. This platform allows each team member to have editing permissions, enabling them to modify diagrams as needed. Furthermore, team members can monitor the modifications and progress of the diagrams in real-time, facilitating smoother and more effective collaboration across the team.

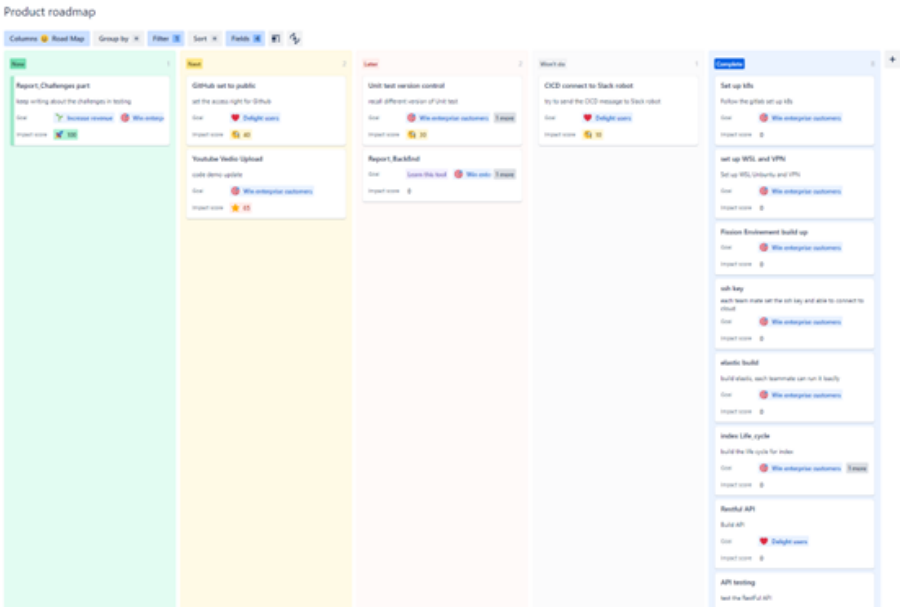
5.3.3 Communication

We utilized Slack to facilitate communication among team members, maintaining a high level of activity throughout the project's development. Each member actively participated in regular weekly discussions, and whenever a problem arose, we convened to collaboratively brainstorm and devise solutions.



5.3.4 Jira

Our team employed Jira to track the project's progress in real time. We broke down the tasks into categories such as Now, Next, Late, Won't Do, and Complete, allowing us to monitor our current status, upcoming tasks, overdue features, discarded ideas, and completed features. This structured approach provided clear visibility into each aspect of the project's development.



5.3.5 Docker

Because our backend is based on Fission and ElasticSearch that are remotely deployed on the, all connections require a successful VPN setup so that the connections can be established. This means during our development and testing phases, we need to keep the VPN connection open and also our process will be easily affected by some fluctuations on the cloud.

6 Links

- YouTube Playlist
- GitHub Repository
- Jira
- Slack
- Mastodon Data stored in Drive

7 Teamwork and Contribution

Name	Contribution
Jonghao Song	Hosted all meetings Report Latex repository administrator GitHub repository administrator Applied for a Mastodon developer account & developed a data crawler Assisted with middleware and backend integration Report writing Data collecting & cleaning
Shiya Fu	Implementation of middleware and connection to the backend Implementation of all fission function Insert data to ES Made demo video for back-end Report writing Collected Data Built and debugged with the RESTful APIs
Gaoyuan Ou	Set up Kubernetes Cluster environment and deployed ElasticSearch and Kibana Collected data Designed the scenarios Visualised the data collected Made demo video for front-end Report Writing
Mingxin Li	Data preprocessing Collected data Design the scenarios Writing the report Data visualisation in JupyterNotebook
Zhuoyang Liu	Unit test section GitHub CI/CD Building Jira Organising meeting Requirement editing Writing report Collected Data Data pre-processing Youtube video update

References

[1] M. Fogli, T. Kudla, B. Musters, G. Pingen, C. Van den Broek, H. Bastiaansen, N. Suri, and S. Webb, "Performance evaluation of kubernetes distributions (k8s, k3s, kubeedge) in an adaptive and federated cloud infrastructure for disadvantaged tactical networks," in *2021 International Conference on Military Communication and Information Systems (ICMCIS)*, 2021, pp. 1–7.

[2] J. Bai, "Feasibility analysis of big log data real time search based on hbase and elasticsearch," in *2013 Ninth International Conference on Natural Computation (ICNC)*, 2013, pp. 1166–1170.

[3] J. Bloch, "How to design a good api and why it matters," in *Companion to the 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications*, ser. OOPSLA '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 506–507. [Online]. Available: https://doi.org/10.1145/1176617.1176622