Cluster and Cloud Computing COMP90024 – 2025 S1

Assignment 2 – Team 55

Prepared by

Josh Feng

chenhaof@student.unimelb.edu.au

1266669

Fan Yi

yify@student.unimelb.edu.au

1193689

Siyan Pan

siyanpan@student.unimelb.edu.au

1627057

Shijia Gu

shijiag1@student.unimelb.edu.au

1266137

Weiqi Chen

weiqchen@student.unimelb.edu.au

1266370

21st of May 2025

ABSTRACT

This is a report for COMP90024 Assignment 2. This project designed and implemented a distributed system for harvesters, processing and analysing social media data from Mastodon, Reddit and Bluesky. The system is deployed on MRC/NeCTAR Research Cloud and utilised Kubernetes and Fission for function-level scheduling and scaling. We collect data through time triggers and different methods, and with Redis we complete the data cleaning, sentiment analysis and keyword extraction layer by layer, only the valid data is sent to Elasticsearch. The frontend user can send query requests through the HTTP interface to filter data, we designed three types of scenarios around AFL, Cost of Living and Australian Election. The system has clear structure, decoupled modules and good scalability.

Keywords NeCTAR Research Cloud · Kubernetes · Fission · Elasticsearch · Mastodon · Reddit · Bluesky

Table of Contents

1. Introduction

1.1. Background

In today's era of explosive data growth, big data analytics has become a powerful tool for understanding public opinion and group behaviour. With the rise of large-scale and unstructured information, building scalable and high-performance data processing systems is more important than ever.

This project is built using the Melbourne Research Cloud [3] under NeCTAR Research Cloud [4], Kubernetes [11], Fission [6], and Elasticsearch [7]. The Melbourne research cloud is a national cloud infrastructure in Australia that offers flexible and powerful computing resources for research. Kubernetes is a container orchestration platform that enables automatic deployment, scaling, and management of distributed services. To support real-time data collection and event-driven processing, we use Fission, a serverless framework on Kubernetes. Elasticsearch, a distributed search and analytics engine, is used to efficiently store and analyse social media text data, allowing fast keyword searches and sentiment trend detection.

Together, these technologies constitute a robust and adaptable analytics platform, enabling reliable ingestion and in-depth analysis of multi-source social media datasets. The system is designed to handle dynamic workloads and provide timely insights into evolving online conversations relevant to the Australian context.

A key focus of this study is the analysis of textual data from Mastodon [1], Bluesky [9], and Reddit [10], with the objective of identifying trends in public sentiment and reactions to key events. Such insights are essential for supporting decision-making in domains including governance, media analysis, and public policy. By observing shifts in social media narratives over time, the system offers a data-driven lens into public opinion and its temporal evolution.

1.2. Research

This project investigates the changing landscape of public interest and sentiment on social media from 2023 to 2025, with a focus on posts relevant to Australia. Three topics were selected based on their relevance and interest to the study:

1. Which AFL teams are most supported?
2. How do Australians feel about the rising cost of living and inflation between 2023 and 2025?
3. Who is likely to win the upcoming Australian election based on public sentiment?

Natural language processing techniques and sentiment analysis models (VADER) [8] were applied to the data collected from multiple platforms. The goal of this project is to identify influential users and extract the most discussed keywords across topics, thereby enabling a deeper understanding of platform-specific discourse and long-term public sentiment shifts.

2. System Architecture

2.1. Scenario

The initial idea of the project was to scrape recent posts and comments from various online platforms. To better understand what people in Australia are discussing, we collected the latest posts from Reddit, Mastodon, and Bluesky, which are currently three of the most active social platforms used for public discussion in Australia. In the first stage, we gathered around 2,500,000 posts to get an overview of the situation and identify common themes. For each post, we extracted a list of keywords to represent the main content and topics being discussed. These keywords serve as a simplified representation of user interests. In addition, we used a sentiment analysis library to calculate the sentiment score for each post, ranging from strongly negative to strongly positive. These two features — keywords and sentiment — allowed us to detect trending topics, observe emotional responses, and identify shifts in public opinion.

After filtering and grouping the most meaningful keywords and combining them with recent social and political events in Australia, our research team identified three key scenarios that deserve further analysis. They are the cost of living across different Australian cities, AFL --- Australia's most popular national sport, and political conversations related to elections in Australia. These three scenarios became the focus of our target data collection objectives for the next stage of the project. Each scenario offers a different perspective on Australian public opinion and allows for deeper exploration using social media data.

2.2. Functionality

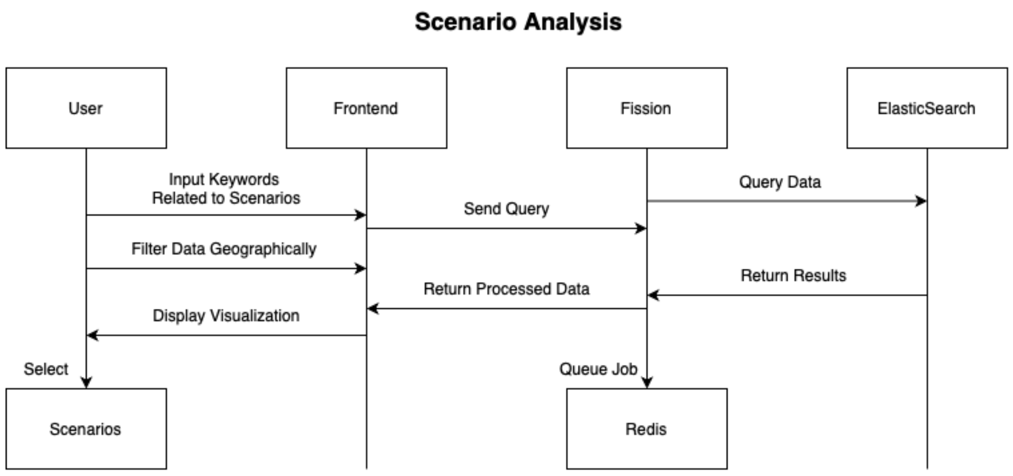


Figure 2.2.1 System architecture for scenario-driven analysis

Our system is a comprehensive platform that can conduct thematic analysis in various fields based on pre-designed requirements and scenarios. Its main purpose is to become an organized and scalable system to provide customized analysis services in hot fields. This system makes data processing easier and simplifies the generation of charts through automation. Furthermore, it reduces the complexity for end users while allowing for in-depth analysis.

Therefore, the system is separated into two layers of function:

1. An interactive frontend based on Jupyter Notebook, which displays critical information and analysis.
2. A distributed backend infrastructure that manages integrates data from multiple sources, keyword-based querying and delivers timely analytical insights.

2.3. Frontend

The frontend is designed to enable setting up a scenario, user interaction and discovering results. Its implementation takes advantage of the programmability and interactivity of Jupyter Notebook, enabling flexible workflows to support exploratory analysis. Figure 2.2.1 shows the entire data flow and describes the user's usage of the system through scenario-based queries and dynamic data retrieval and visualization of the results.

The major functions include:

- **Scenario Selection**: Users first make choices in pre-determined scenarios, such as cost of livings, AFL, or elections. Each scenario loads predefined keyword clusters and structure of analysis automatically. Choosing an election scenario loads political keywords (e.g., vote, Albanese, Liberal Party) and electoral period-appropriate temporal filters automatically.
- **Interactive User Queries**: Once a context is chosen, custom keywords are entered and optional geographic and temporal constraints added. These inputs are then bundled and passed to the backend. For instance, an election analyst might be interested in seeing week-over-week changes in mentions about "vote" prior to the election day or comparing public opinion about "Albanese" among Australian states.
- **Dynamic Visualization and Statistical Summary**: Once processed, results get displayed in visual form in the notebook interface. The visualizations, which are produced through libraries matplotlib and seaborn, consist of keyword frequency ranks, time-based graphs post counts, and SA3-based heatmaps. In the context of AFL, users can use the name of a team,

"Magpies", for a query and see an immediate sentiment ranking, indicating how polarized people's opinions on this team are.

- **Real-time Update and On-Demand Analysis**: The front end is capable of iterative exploration. The results can be refreshed by modifying keywords, adjusting the time filter, or calling a new instance of the fission function. For instance, people can filter out election-related tweets within the past 48 hours and observe a brief surge in negative sentiment.
- **In-depth Discovery**: Users can also initiate model refreshing (for example, reload the updated keyword list or sentiment dictionary) and view metadata trends. In the context of elections, this might require an in-depth study of specific SA3 areas to compare the performance of different regions in terms of the rising negative sentiment of specific political policies. This multi-functional framework supports advanced trend analysis as well as fine-grained user-driven narratives. For data scientists and researchers who need to analyze workflow transparency, customizability and controllability, the notebook interface is particularly important.

2.4. Backend

To enable flexible scenario-based analysis, interactive visualizations, and real-time querying in the frontend, the backend is required to be able to handle data ingestion, transformation, and retrieval efficiently. Our application makes use of several integrated components, including distributed storage (ElasticSearch), serverless function (Fission) and task management (Redis) within a backend infrastructure.

This backend pipeline is intended to be completely automated and responsive to changing user requirements. It consists of:

- Consistent consumption of unprocessed data from social media websites.
- Scenario-based filtering and processing according to user input.
- A well-structured database.
- Incorporation of an existing model that is pre-trained for classification and sentiment scoring.

To support simultaneous access and high volumes of queries during active data analysis times, the system is hosted on a scalable cloud-based platform. This allows for responsiveness, stability, and expandability for analytical and research purposes.

Additional technical specifics and design decisions are discussed in Section 3.

3. Implementation

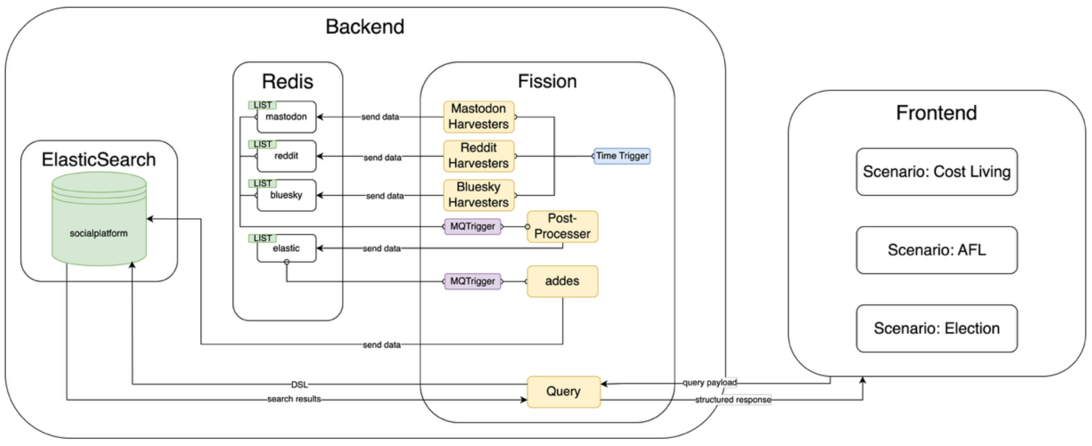


Figure 3: Overall system structure and working pipeline

3.1. Data

3.1.1. Mastodon

One of the platforms used to harvesting the data is Mastodon [1], which is an open source and decentralised social platform that allows user to post at any time and any content, which is unique advantages in terms of accessibility, freedom and data structuring, we use it as one of the important data sources for this project. Mastodon has a complete and open REST API [2] to capture structured data such as user post and user account detail without the need for crawler, which making it ideal for data collection and analysis. And it has very few API restriction, compared to other social platforms, it has low access threshold, loose flow restriction mechanism, rich fields, support paging and keyword retrieval, almost any public data can get which make it easy to build a stable harvesting module.

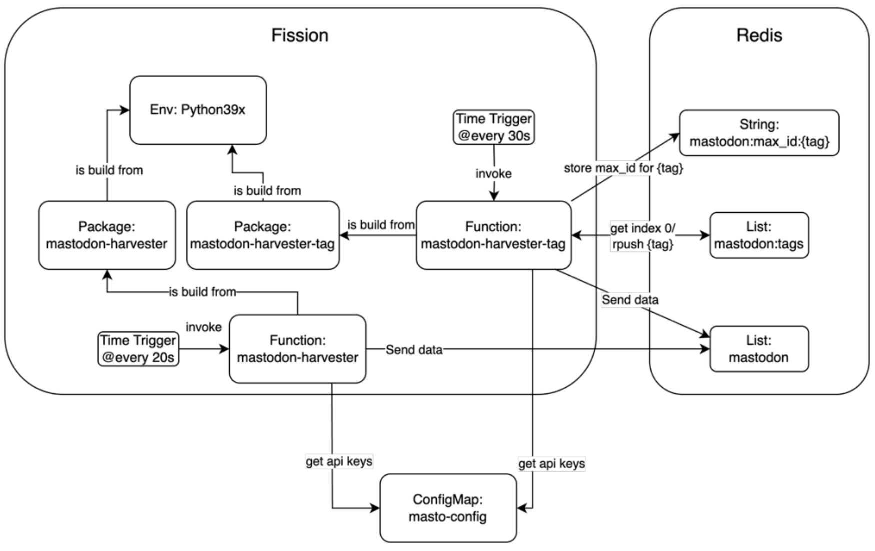


Figure 3.1.1: Mastodon harvester architecture in cluster

The diagram above shows the deployment structure and component interactions of the Mastodon data harvesters in our cluster. The Mastodon data harvester has two main functions:

- mastodon-harvester: the function is to collect posts from the public timeline of Mastodon, which is bound to a Time Trigger that is automatically called every 20 seconds. Each call will fetch up to 40 recent public posts from Mastodon and sends the data to a list in Redis called mastodon.
- mastodon-harvester-tag:
    - The function starts each time will first get the tag with index 0 from the *mastodon:tags* list in Redis as the keyword for the current round of harvesting. The function uses that tag as the keyword to fetch relevant posts from the current time until 1st of January 2023, working backward in time.
    - Each call will only fetch 40 posts and saves the maximum ID (*max_id*) of the posts that have been crawled as a Redis string, e.g. *mastodon:max_id:{tag}*, and then, the tag will put back at the end of the *mastodon:tags* list for subsequent polling. The function will continue to page to earlier data based on that *max_id* until it meets the termination condition, i.e., reaches the lower time limit or there are no more posts. After the reached termination condition, the tag will not put back.
    - The collected data will put into mastodon list in Redis. In addition, in the *mastodon:tags* list, the tags in this list are all related to Australia and cover a wide range of areas such as place names, government, science and technology and culture.

Both functions will send the harvested data to a list in Redis called mastodon, which is bind with an MQTrigger that triggers a subsequent data processing function. This process is described in detail in the section 3.1.4.

3.1.2. Reddit

Reddit, a platform possessing a vast quantity of users and communities with a high degree of activeness for social news aggregation, content rating, and discussions, is one of the significant data origins for this project to capture public opinions and trend dynamics. Its special structure of subreddits (the so-called subreddit) make it possible that users to deeply discuss some special topics, providing for us abundant and also diversified datasets. This project make utilization of PRAW (Python Reddit API Wrapper) library (version 7.8.1) for the purpose of interacting with Reddit API, to actualize an efficient and pinpointed data collection.

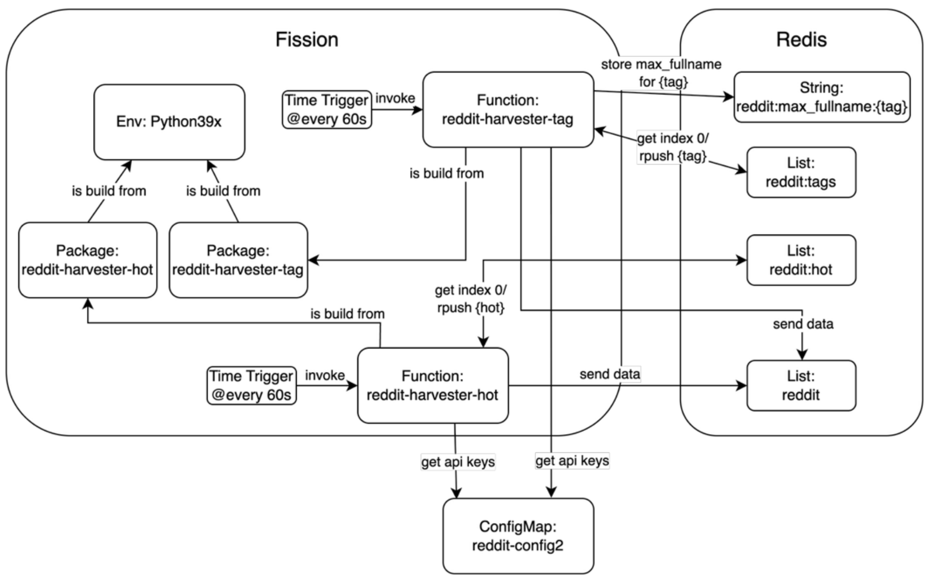


Figure 3.1.2: Reddit harvester architecture in cluster

The above figure has shown the Reddit data collector's deployment architecture and the interactions of components within the cluster. This architecture is deployed upon the Fission platform (which is based on Python 3.9x environment). Its main parts are two core Fission Functions: reddit-harvester-tag and reddit-harvester-hot. These two functions are all called by their own Time Tigger with a frequency for every 60 seconds. The API keys and other sensitive configurations are managed via the Kubernetes's ConfigMap: reddit-config2. All of the data which are collected will, in the end, be dispatched to a List in Redis, its name is reddit, and this acts as a kind of unified gateway for the data processing that comes next.

The Reddit data collector has two principal functions:

- reddit-harvester-tag:
    - Objective and Mechanism: The core objective for this function is about collecting historical posts, and also their popular comments, from a series of pre-defined, Australia-related subreddits (tags) which are stored in the reddit:tags list in Redis. This list is curated by the team manually, with an aim to complete a one-off historical data retrospective collection for specific research topics. The function will persistently fetch posts under each tag, until the post's publication time becomes earlier than a pre-set termination date, END_DATE (which is defined as January 1, 2023, UTC).
    - Data Fetching and Pagination: Each time of execution, the function will take one subreddit name from the head of reddit:tags list. By means of PRAW library, it will fetch new posts from that designated subreddit. For the purpose of actualizing historical data backtracking, a pagination

Copyright © Feng. J, Yi. F, Pan. S, Gu. S, Chen. W, 2025

mechanism is adopted by the function. It utilizes a string stored in Redis, reddit:max_fullname:{tag}, which records the fullname of the oldest post fetched in the last batch, and this is used as the before parameter for the next request, thereby to obtain earlier posts. As per the code (functions/reddit_harvester_tag/reddit_harvester_tag.py), each invocation will fetch a LIMIT of 8 posts. Furthermore, for every post, this function will also fetch up to 5 top-level comments, after sorting them by 'best'.

    - Data Transformation and Output: The fetched post data and comment data will be converted into the project-defined standard JSON format through convert_reddit_post_to_target_format and convert_comment_to_target_format functions, respectively. This format includes platform information (platform: "Reddit", version: 1.1), the WorkspaceedAt time, post/comment ID, createdAt time, the content, a boolean sensitive flag, favouritesCount, repliesCount, tags (which includes original post flair, subreddit name, and for comments, an additional parent post ID), the post url, and also detailed author account information. The formatted data is subsequently dispatched via an HTTP POST request to Fission Router's /enqueue/reddit interface, ultimately entering into the reddit list of Redis.
    - State Management and Termination: After each round of fetching, the fullname of the oldest post under that subreddit is updated to the corresponding reddit:max_fullname:{tag} Redis key. Once the timestamp of the oldest fetched post is found to be earlier than the END_DATE, that subreddit name will be removed from the reddit:tags list, which signifies the completion of historical data collection task for that specific tag.

- reddit-harvester-hot:
    - Objective and Mechanism: This function is designed for the continuous monitoring and fetching of the newest posts from a series of Australia-related, popular subreddits. This list of subreddits (for example, AusEcon, AustralianPolitics, AFL, sydney, melbourne, etc.) is manually maintained and stored in the reddit:hot list within Redis. Despite the function's name including "hot", its actual operation, according to the code (functions/reddit_harvester_hot/reddit_harvester_hot.py), is to fetch "new" (most recent) posts from these subreddits.
    - Data Fetching and Polling: Upon each execution, the function retrieves a subreddit name from the head of the reddit:hot list. After completing the fetch of the latest posts from this subreddit (the LIMIT is 40 posts,

according to the code), this subreddit name is then re-appended to the tail of the reddit:hot list. Such a design actualizes a continuous polling monitor and data collection from these key subreddits. This function only fetches post data, and does not process comments.

- Data Transformation and Output: The fetched post data are converted to the standard JSON format using the same convert_reddit_post_to_target_format function as used in reddit-harvester-tag. Similarly, these are also dispatched to the /enqueue/reddit interface, flowing into the reddit list in Redis.

Through these two complementary harvester functions, the system is enabled to both comprehensively look back at historical discussions on specific topics and to persistently track the latest dynamics from active communities. All data collected from Reddit enters the subsequent processing flow in a uniform format, which lays a solid foundation for further data analysis, which are detailed in section 3.1.4.

3.1.3. Bluesky

Bluesky is a 2023-launched decentralized social networking site that seeks to give users more agency over their data and how content is disseminated. The platform is distinct in that it isolates identity, storage, and feed algorithms and is more modular and open in nature than typical platforms. However, because it is so recently launched, the quantity of data that is readily available is yet less than more well-established platforms, including Reddit and Mastodon.

Also using a username and application key to access data, Bluesky API is restricted to accessing only material posted by individuals followed by the user account. This limit gathering broad, publicly available data streams. Thus, we use a keyword-based search strategy to target pertinent publicly posted material throughout the network. This is a technique that necessitates predefined determination of the analysis scenario so that an adequate keyword list can be produced.

As soon as the context is established, the Bluesky harvester retrieves posts with those keywords at a predetermined frequency and adds real-time data into an aggregated pipeline shared among platforms. This makes Bluesky an auxiliary data source.

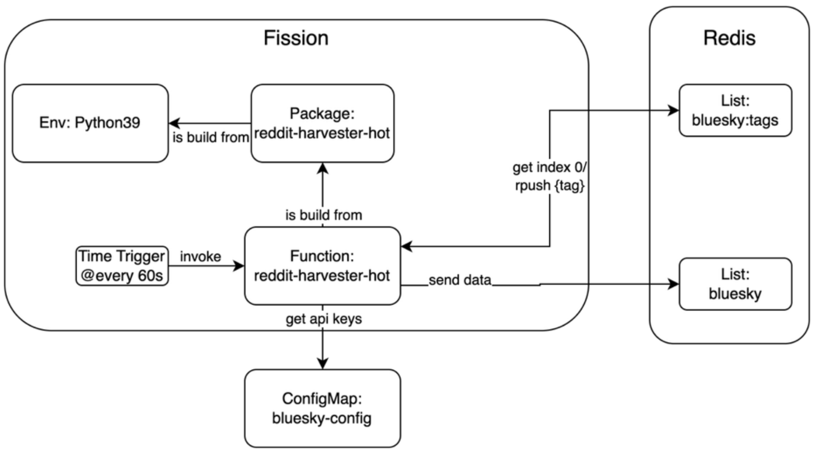


Figure 3.1.3: Bluesky harvester architecture in cluster

The picture above shows how the Bluesky data collector is set up and how its parts work together. This function runs again and again at fixed times and works with Redis to get data using keyword tags. The Bluesky harvester includes the main function:

- bluesky-harvester-tag:
  - This function is set up with Fission and runs every 60 seconds using a timer (CronTrigger). It runs in a Python 3.9 environment.
  - Each time it runs, the function gets the first tag (at index 0) from a list in Redis called bluesky:tags. This tag contains the regions in Australia to get the Australian related data.
  - The function uses the tag to ask the Bluesky API for posts. It gets up to 40 posts each time. These posts must include the tag in their content or extra info.
  - After it gets the posts, the function saves any needed posts. It then puts the used tag back at the end of the tag list. This lets the system go through all tags in a cycle.
  - The posts it gets are saved in a Redis list called bluesky. This list is connected to a message queue that starts the next steps, like parsing the post, checking sentiment, or other actions. This works the same way as it does for Reddit and Mastodon.

3.1.4. Post data processing and send to Elasticsearch

Mastodon, Reddit and Bluesky each have their Redis list after harvesters send to corresponding list, which is bound to the same post-processor function that is triggered by Fission MQTrigger. The post-processer function will first clean up the raw data and filter the language, clean up the HTML tags in the content and if the content is not English then will drop it. Then, using the VADER model to score the sentiment of the content, generating sentiment value and sentiment Label, positive, neutral or negative, fields for subsequent query and visual analysis. After that, the function will be using

YAKE keyword extractor to extract the top 5 keywords from the content, and all the keywords are converted to lowercase. Finally, the processed data will send to the Redis list called *elastic*, waiting for the downstream function, *addes* to perform the final checksum and send to Elasticsearch.

The *addes* function is the final step we use to send the data to Elasticsearch, it will first check each object of the data for completeness, only the valid data will send to Elasticsearch, and using *{platform}_{post_id}* as the document ID to avoid duplicate writes.

3.2. MRC and NeCTAR Research Cloud

The project is deployed in the Melbourne Research Cloud (MRC), which utilises a private cloud deployment model. It provides University of Melbourne researchers with an Infrastructure as a service cloud (IaaS) computing service that gives them on-demand access to powerful set of virtualised computing resources for storing, managing and computing research data. Also, MRC offers features comparable to AWS and Microsoft Azure commercial cloud providers and crucially is free to researchers [3]. In addition, the MRC is part of the NeCTAR Research Cloud [4] and is one of its multiple availability zones.

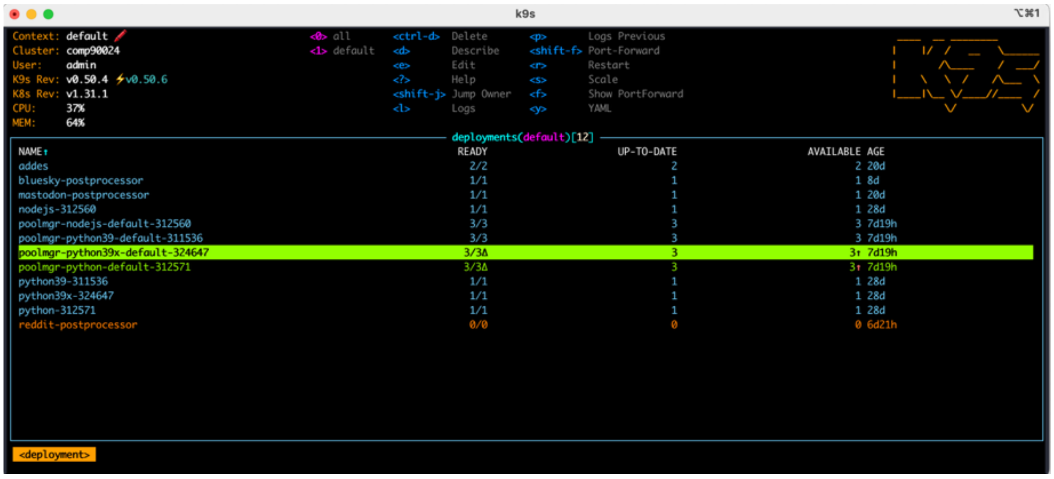
MRC is underlying architecture is based on the open-source OpenStack cloud platform [5], which provides complete IaaS capabilities including centralised management of compute, storage and network resources which can be accessed and managed through either a graphical interface or command line tools. We created a Kubernetes cluster in the MRC environment using the OpenStack CLI, the cluster consists of 1 master node and 3 worker nodes using the template “Kubernetes-v1.31.1-melbourne-qh2-unom-v4”, Appendix 8.3 shows the specific creation commands and parameter configuration.

Use of MRC will bring lots of advantage, it provides flexible resource scheduling capabilities, although currently have total of 4 instances, but retains the remaining quota resources, which provides space for future system expansion. Also, we can assign public Ips, configure network topology and firewall policies on demand. We can also utilise the cinder service provided by OpenStack to achieve volume snapshot and backup functions, improves the fault tolerance and recoverability of the system.

To improve the efficiency of the team and the flexibility of accessing resources, we share the Kubernetes configuration file so that every team member can access the cluster resources. We notice that the config file contains the credentials needed to access the cluster, we took special care to avoid leaking it. We did not upload it to GitHub, shared drive or any public network, but transferred it via a physical flash drive to ensure that access was limited to internal team members and to prevent the risk of potential unauthorised access.

3.3. Kubernetes

We use Kubernetes as container orchestration engine to manage all the service components deployed on MRC. Kubernetes is responsible for the unified scheduling of pods, configuring network services, managing storage volumes and maintaining the desired state of the system, it is the core of operation of the entire cloud application. The Kubernetes is created on the MRC through the Magnum service provided by OpenStack, which automatically schedules the requirement VM resources and worker nodes. We created with 1 master node and 3 worker nodes with each node corresponding to separate VM instance with 2 vCPUs and 9GB of RAM. This deployment structure provides good foundation for the system, and it has good horizontal scalability, we can later add more worker nodes according to the use of resources and not need to interrupt the existing service.



```
Context:  default          <0> all       <ctrl-d>  Delete        <p>        Logs Previous
Cluster:  comp90024        <1> default   <d>       Describe      <shift-f>  Port-Forward
User:     admin                          <e>       Edit          <r>        Restart
K9s Rev:  v0.50.4 ⚡v0.50.6               <?>       Help          <s>        Scale
K8s Rev:  v1.31.1                        <shift-j> Jump Owner    <f>        Show PortForward
CPU:      37%                            <l>       Logs          <y>        YAML
MEM:      64%

                                  deployments(default)[12]
NAME↑                                     READY          UP-TO-DATE          AVAILABLE AGE
addes                                       2/2                   2                  2 20d
bluesky-postprocessor                       1/1                   1                  1 8d
mastodon-postprocessor                      1/1                   1                  1 20d
nodejs-312560                               1/1                   1                  1 28d
poolmgr-nodejs-default-312560               3/3                   3                  3 7d19h
poolmgr-python39-default-311536             3/3                   3                  3 7d19h
poolmgr-python39x-default-324647            3/3Δ                  3                 3↑ 7d19h
poolmgr-python-default-312571               3/3Δ                  3                 3↑ 7d19h
python39-311536                             1/1                   1                  1 28d
python39x-324647                            1/1                   1                  1 28d
python-312571                               1/1                   1                  1 28d
reddit-postprocessor                        0/0                   0                  0 6d21h

<deployment>
```

Figure 3.3: deployment status by k9s

Kubernetes contains many advantages, one of which is pod abstraction and service isolation. Kubernetes uses pods as the smallest scheduling and execution unit and each pod runs one or more coupled containers. In our cluster, all the key components such as Fission, Elasticsearch and Redis runs as pods, which fully reflecting the advantages of Kubernetes architecture. In particular, each Fission function runs in separate pod, which automatically scheduled, and pod creation, updates, restart and other operation are done by the Kubernetes control plane, so we don't need to worry about which node underlying run is on. Figure 3.3 shows the status of the system, and we can see that each Fission function runs as separate pod and automatically creates or destroys pods based on function triggers.

Kubernetes also provides complete RESTful API server that supports for any operations on all resource which improves development efficiency in cluster management, service deployment and testing. We call the RESTful API indirectly through the *kubectl* command to create, update and monitor resource. Kubernetes provides unified internal DNS resolution mechanism for services within the cluster, we can access the internal address of component by its service name, these internal addresses do not require fixed Ips, they are automatically resolved and scheduled by Kubernetes internal DNS and web service, this improves our development efficiency, also enhances system security.

3.4. Fission

We use Fission [6] which is Function as a Service deployed on Kubernetes, all the core functional modules in the project are composed of small independent function, each function is responsible for handling specific type of task, we have variety of independent functions, such as collecting data, processing to calculate the sentiment value for the data and extracting the keywords in the data content, sending to Elasticsearch and for interacting with frontend to handle the data filtering, these function are run in event-driven manner and deployed as pods in the cluster through Fission function scheduling and management mechanism. This brings many benefits:

- Highly module: each function is encapsulated as independent function unit with single responsibility, making easy to develop, test and deploy separately. In the actual development process, we assign different function to team members to be responsible, each function can be developed, debug and deploy independently of each other, this structure significantly improves efficiency and collaboration flexibility.
- Event-driven: Fission supports many different event triggering mechanisms, we mainly use three types of triggers
    - HTTP Tigger: create route via fission route so that the function can be accessed via URL, we mainly used for called by front-end
    - Timer Tigger: through the timed task regularly triggers the function to execute, we mainly used for data harvest, so that in the cluster, the data will be collected 7 * 24 hours.
    - Message Queue Tigger: we use Redis as messaging middleware to push data into specified queue, and when new data comes in, it triggers subsequent processing function to complete analysing, data cleaning and send to Elasticsearch.

We use Python language for function development. For function is develop locally, we create corresponding folder, which contain:

```
function/
├── __init__.py
├── function.py
├── build.sh
```

├── requirements.txt

Where __init__.py is the standard Python package initialisation file that declares the directory as a module. function.py is the main logic entry for the function and must define main entry that Fission will call to execute the logic, the actual file name will be named related to the main logic. build.sh is the script to automatically install the dependencies declared in requirement.txt during the function packaging phase in Fission. requirements.txt contain the dependencies in the main function that need to install in advance in fission package.

We use Fission CLI to create YAML file for the package, function, trigger and other resources locally and upload to the cluster. This way is easy to manage, and batch operate, also can be directly integrated into the CI/CD pipeline in GitLab to automate the build, test and go alive which suitable for team collaboration and rapid iteration.

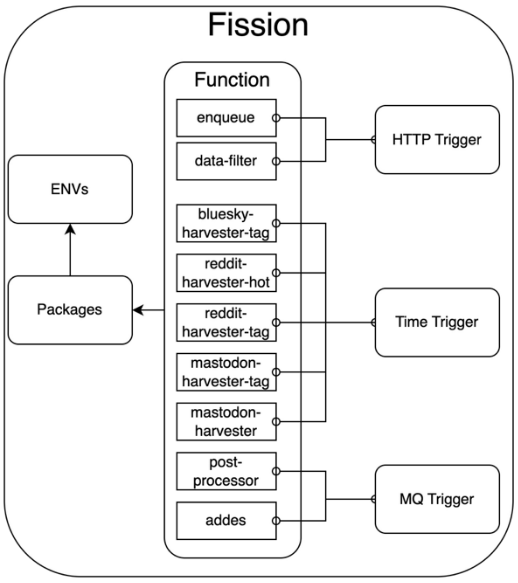


Figure 3.3: functions deployed in Fission in cluster

The figure above shows the overall structure of the Fission function deployed in the cluster. Different functions bind different types of triggers depending on their purpose:

- Time Trigger: harvester relevant functions are mainly run periodically via Timer Trigger, and they are mainly to harvester posts on the corresponding social platform.
- MQ Trigger: The *post-processor* and *addes* are bound to the MQ Trigger.
    - post-processor: the function which is bound to the list of raw posts in Redis, such as mastodon, bluesky and reddit, and its main purpose is to compute the sentiment value of each raw post and extract the top 5 keyword from the content of the raw post.

    - addes: in the Redis, there is a list called elastic, this function is bound to this list, the main purpose is to check the final data, make sure that each data is complete and finally put into the Elasticsearch.
- HTTP Trigger: the functions, enqueue and data-filter, expose the REST interface to the public via the HTTP Trigger
    - enqueue: this function acts as unified data entry and message distribution component. All the data by the harvester functions is sent in JSON format via HTTP request to the */enqueue/<topic>* interface of the Fission Router, and the enqueue function will send the data to the corresponding Redis list.
    - data-filter: this function opens the interface to front-end application via HTTP trigger to support filtering data from Elasticsearch according to the query condition set by the user.

Our system consists of number of function modules with clear responsibilities and deployed in the Fission. The functions work together through different triggering methods to form a loosely coupled and easily scalable event-driven processing architecture.

3.5. Elasticsearch

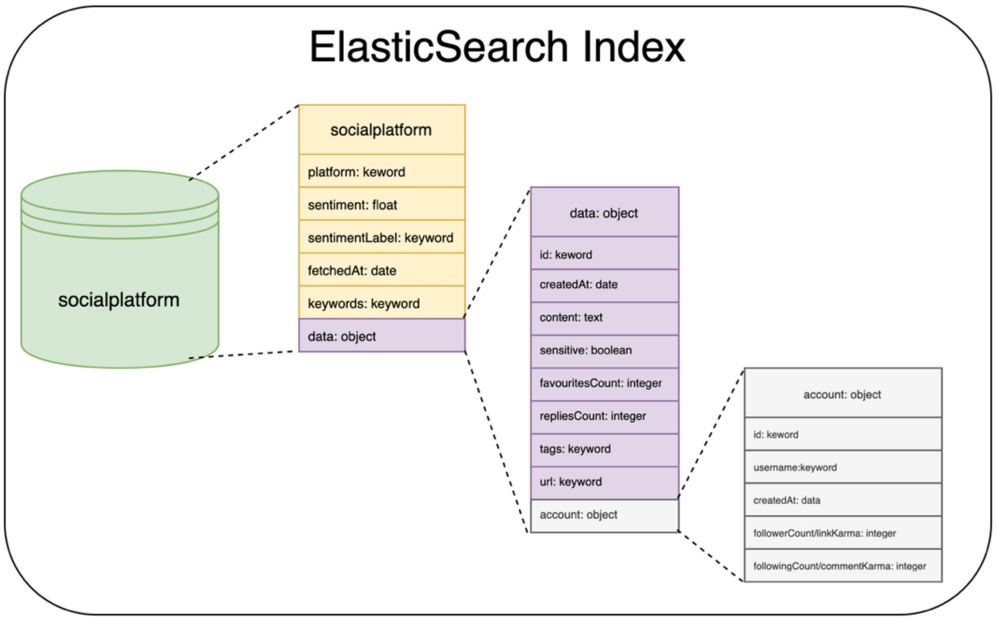


Figure 3.5: Database Structure and data mapping

In order to achieve efficient and structured social platform storage and retrieval, we use Elasticsearch as database. Elasticsearch is a distributed document database based on Apache Lucene, which designed for full-text search, aggregation analysis and is widely used in information retrieval and visualisation platforms [7]. Using Elasticsearch bring many advantages to our project:

- Supporting JSON Document Structure: the harvester in our Fission captures data in JSON format, while Elasticsearch supports JSON document structure natively and can store it directly without structure conversion.
- Powerful Search Capabilities: Elasticsearch supports query DSL such as full text search, exact match and time range filtering, which is ideal for supporting for our front-end visualisation and interactive analysis. In the early stages, we used Kibana to explore the visualisation for our data from Elasticsearch.
- Scalability and Clustering Support: Elasticsearch supports multiple shards and replicas to scale horizontally in Kubernetes to cope with large scale data writes and highly concurrent queries.
- Integration: Elasticsearch is natively supported by Kubernetes and can be quickly deployed via Helm. And Kibana integrates seamlessly for visual querying, debugging and data exploration.

We installed Elasticsearch and Kibana using Helm and ran it in a separate elastic namespace in Kubernetes. After team discussion, we decided to store data from different social platforms in a shared index instead of creating separate indexes for each platform. We unified the data structure to facilitate horizontal analysis and comparison and reduce management complexity. We created an index named *socialplatform* to unify the storage of social data collected and processed from each platform. The index is configured with 3 primary shards and 1 replica to support concurrent queries and disaster recovery (Appendix 8.4 shows the complete content of index). Figure 3.5 demonstrates that the field structure of the Index is designed with multiple levels of nesting, including the data platform, sentiment, keywords and related original data and account data.

We deployed a function called *addes* in Fission, which is the last step of the data processing chain. The function receives the processed data that has been processed by sentiment and keyword extraction and performs the final data validation to ensure data integrity and index consistency. We specify a custom ID for each data in the format "{platform}_{post_id}" which ensures that the same social platform post will only be indexed once, and Elasticsearch will reject duplicate insertions to avoiding data redundancy.

We also deployed a function called *data-filter* in Fission, which main purpose is to expose an interface to the frontend via HTTP Trigger. The frontend user can customise the filtering of social platform data based on keywords, tags and time, the front-end simply send a payload containing the filtering criteria to the interface to return structured search results, the filter conditions include:

- content: indicated to data.content, support fuzzy search and full-text search with OR spliced keywords
- tags: indicated to data.tags, exact match

Copyright © Feng. J, Yi. F, Pan. S, Gu. S, Chen. W, 2025

- keywords: indicated to keywords, exact match
- data_range: indicated to data.createdAt

The data-filter function support scroll paging feature and big data query. In order to support the paging of large number of results and limit the amount of reading, we used Elasticsearch Scroll API to realise data batch pulling. User can control the number of single batch and total query through the *size* and *max_docs* field in the payload to avoid overloading the system resources.

3.6. Error Handling

In this project, our data collection work is across many social media platforms, for example, Mastodon, Reddit, and Bluesky. These platforms have very different data structures, also their API rules are not same. So, an effective error handling mechanism is very important for ensuring data collection's continuity, and data conversion's accuracy, also the final dataset quality. This chapter will give detail explanation for main error types encountered during data get and integration, also the limitation factors, and the according handling strategies.

3.6.1. Challenges and Treatments in Heterogeneous Data Unification

One of the core challenges of multi-platform data integration is its inherent heterogeneity. Data returned from each platform is obvious different in field names, in level structures, and data formats. For to ensure the subsequence analysis's uniformity and high efficiency, all collected original data must be converted and mapped into a pre-defined, standardized JSON data structure (the specific structure can see in this report's Mastodon data example and the defined unified JSON format).

In this conversion process, main error handlings are focused on these aspects:

- **Field Mapping and Missing Value Dealing:** For source data fields that not completely correspond to target unified format, we built clear mapping rules. For non-key fields defined in target format, if they are missing in source data, we adopt measures like leaving blank or give pre-defined null value (for example, sentiment analysis fields sentiment and sentimentLabel are set to null during initial fetch). This is to guarantee complete import of data record. If key fields (like post ID, content, creation time, etc.) are missing, this record will be marked in conversion stage, and relevant information recorded, to avoid incomplete data flow into downstream.

- **Data Type Check:** When converting data, conduct basic check for field's data type, to ensure it meets the unified format's data type rules (for example, favouritesCount should be integer, sensitive must be boolean). If type mismatch happens, system will try safe conversion. If conversion fail, errors will be recorded.

- **Format Standardization:** Specially for fields like date-time that need specific format (like WorkspaceedAt, createdAt need to be UTC standard time format), conduct strict format check and conversion, to ensure time information's consistency and comparability.

Through above treatments, our aim is to maximumly keep valid information, and at same time ensure the data that entering subsequence processing and final storage is consistent in structure and standard.

3.6.2. API Limitations of Specific Platform and Data Get Strategies

Each social platform API has its specific use limits and data get characteristics. This requires us to take targeted error responses and strategy adjustments in data collection.

Reddit Platform

- **API Access Limit:** Reddit API has official set access frequency limit (for example, each account about 3000 requests per day). For to strictly obey this limit, and avoid service interruption caused by exceeding quota, our data collection strategy (see 3.1.2 Reddit part for detail) focuses on targeted topic and community for data fetching, not doing large-scale, indiscriminate full crawling. This method helps to maximize data collection's relevance and efficiency within API permission scope.
- **Data Supplement Strategy (Comment Data):** Considering that only post content may not fully capture discussion of some topics, this project also includes Reddit post's comments into collection scope. When processing comment data, we need to ensure comments are correctly associated with their original main post. Error handling focuses on dealing with temporary failures that may happen when getting comments, or some comment data is incomplete, to ensure these problems do not interrupt normal collection of main post and other comments.

Bluesky Platform

- **Data Fetching Limitation:** Bluesky platform's API has specific constraints on data acquisition. For instance, when fetching data in a designated time window, the upper limit for posts obtained in single request is about 1000. Furthermore, data get scope at current stage is mainly focused on content published by followed users. This makes it more difficult to conduct wide data searching across whole platform.
- **Coping Strategy:** Because of above limitations, we adopted strategy of targeted data collection after research topics are clear. This means, when specific analysis targets (like particular events or topics) are decided, collector will try to

focus on users or content streams related to that target. On error handling level, when API returns response indicating limit reached or no more data available, collector will record such situation, and make according to action based on preset logic (like wait for next collection window or end current target's collection), to avoid invalid requests.

3.6.3. Management of Large-Scale External Data Import (Reddit Archive Data)

This project was once considered to integrate a large-scale external Reddit historical data archive (original mentioned total size about 2.98TB). Although final processed and included data for analysis is about two million plus entries, much smaller than this archive's total, when processing any large external data import, potential system resource pressure (like CPU, disk I/O) and impact on existing cluster stability are things must be considered.

For this kind of large data import, main error management and risk avoidance strategy is to use phased, selective import method. Specifically:

- Break down data import process into many smaller, manageable data batches.
- Execute import operations during periods of lower system load, to reduce performance impact on other running components.
- This small-batch processing way also makes it easier if meet problems like data corruption, format incompatibility during import, to locate problem batch and isolate or re-process it, without causing whole import task failure or polluting global dataset.

3.6.4. Conclusion

Error handling measures described in this section, from heterogeneous data's unification mapping, to obeying different platform API's use limits, and also managing potential risks of large data import, together they form important part of this project's data assurance system. Implementation of these strategies aims to improve stability of data collection and accuracy of data, to lay solid foundation for subsequent data analysis work. Error handling is a continuous focus and optimization process. As our understanding of platform characteristics deepens and new challenges appear, related strategies will also be constantly adjusted and improved.

3.7. Unit test

To verify the robustness and fault tolerance of the fetch_post_data() function, a series of unit tests were conducted using both fixed and random mutations on live posts retrieved from the Mastodon API.

1. A valid live post was retrieved using the Mastodon API.
2. The post was mutated to simulate various forms of malformed or invalid inputs:
    - Missing fields (e.g., account, username)
    - Incorrect data types (e.g., tags set to a string instead of a list)
    - Invalid formats (e.g., created_at set to "not-a-date")
3. Each mutated post was passed into the fetch_post_data(post) function.
4. The system's response was observed:
    - Whether exceptions were raised
    - Whether errors were logged or caught
    - Whether ingestion was properly rejected or processed

Description	Mutation Type	Target Path	Excepted Outcome
Remove "account"	Field removal	["account"]	Raise error and skip post
Remove "account.username"	Field removal	["account", "username"]	Raise error and skip post
Set "created_at" to None	Type mutation	["created_at"]	Raise error and skip post
Set "followers_count" to string	Type mutation	["account", "followers_count"]	Raise error and skip post
Set "tags" to string	Type mutation	["tags"]	Raise error and skip post
Set "account.created_at" invalid	Format mutation	["account", "created_at"]	Raise error and skip post

The implementation of mutation-based unit testing for the `fetch_post_data()` function played a crucial role in improving the overall robustness and stability of the system. By simulating incorrect or unexpected input data, these tests effectively reproduced real-world scenarios where social media APIs may return inconsistent or corrupted data. During testing, several issues were identified in the initial version of the ingestion function, including:

- Missing fields such as `account` or `username` leading to unhandled exceptions
- Type mismatches (e.g., `followers_count` provided as a string) causing downstream errors
- Improperly formatted timestamps (e.g., `created_at` set to a non-ISO string) resulting in parsing failures

These failures revealed critical vulnerabilities where invalid posts could crash the entire pipeline or interrupt the data flow.
In response to these findings, the `fetch_post_data()` function was revised to include:

- Field validation and default fallback to gracefully handle missing or null values
- Type checking and data sanitization to ensure that only well-structured posts are passed to downstream modules

As a result, the system can now process live data from Mastodon without crashing, even when encountering corrupt or incomplete posts.

These improvements significantly enhanced the resilience of the ingestion subsystem, ensuring that:

- Unexpected API changes or data inconsistencies do not compromise system integrity
- Downstream analytics modules, such as sentiment analysis and keyword extraction, operate on clean, structured data
- The system can scale reliably in real-world conditions, where noisy data is common

Importantly, the testing framework developed for Mastodon ingestion was also extended to other platform ingestion functions, such as those handling data from Bluesky and Reddit. By applying mutation-based validation during early development stages, format inconsistencies were detected and resolved proactively. This approach has substantially reduced the likelihood of runtime failures due to structural anomalies, contributing to a more stable and unified ingestion pipeline across all supported platforms.

3.8 Redis

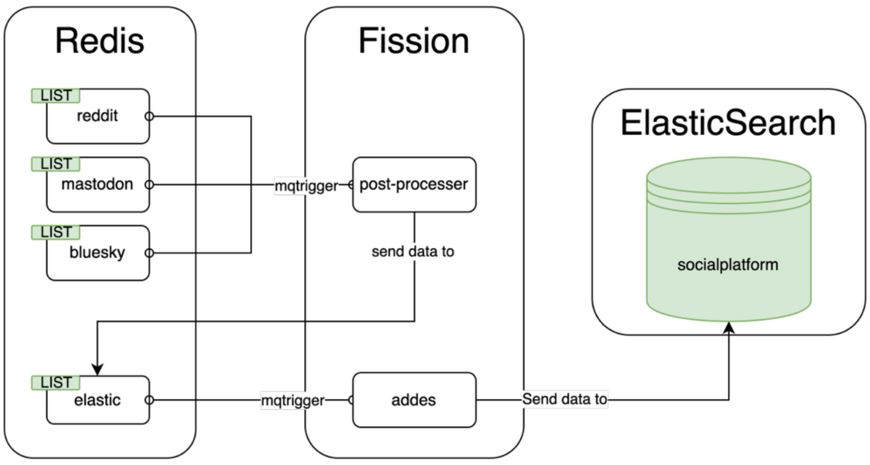


Figure 3.8: Event Driven Processing Flowchart

We have implemented an event-driven processing pipeline using Redis as a data communication and triggering intermediary between functions. In our system, there are many independent harvester functions, if they directly send data into Elasticsearch after fetch data, it may encounter potential problems such as storing duplicates data and some data may have missing content. As shown in the figure above, we have designed the data processing flow as event driven functions, and it can be divided into three parts:

- upstream function: harvesters function, responsible for fetch raw data from different platforms and send into corresponding list such as reddit and mastodon, via *enqueue* function
- midstream function: post-processer function, consumes data from Redis via MQTrigger, complete sentiment analysis and keyword extraction and then writes it to another transit list called elastic
- downstream function: addes function, structural validation and ensures that data store to Elasticsearch conforms to uniform standards.

Redis also act as the role of task scheduler and crawler control in our system. Some of harvester functions do not directly harvest random data, but dynamically construct harvesting tasks based on a list of tags stored in Redis. Using Redis significantly improves modularity, scalability and controllability of data flow of the system and is a key component in building a stable and efficient processing.

3.9 GitLab

We use GitLab as our code developing and version controlling platform, and we followed standardised Git branching strategy to keep the development process disciplined and manageable. Our project link shows on the appendix 8.2.

We set up a protection rule on the main branch that prevents anyone pushing code directly to the main branch. All code changes must go through branch merge process to ensure that main remains deployable and stable. We also set up a develop branch as integration test branch. Each developer will create a feature branch and develops the function code. Once completed, we will create pull request and other team member will doing the code review and testing to verify the code. After testing, the stable version of the develop branch will merge into the main branch. This whole process ensures code quality assurance, teamwork and parallel development and clarity of version control.

We have considered automating the deployment process with CI/CD and we have not done this part, but we have realised the value of CI/CD in improving deployment efficiencies and lowering the cost of collaboration during the development process. We have considered that implement an automatic generation and verification of function YAML files, when the developer uploads the relevant Fission function, the script can be automatically executed to generate standardised function and verify the integrity of the fields and the consistency of the dependencies to avoid manual errors. With the GitLab Runner, the CI process automatically preforms Fission spec apply after code merge, the function can update to the Cluster and closes the loop from code commit to development.

4. Data Analysis and Result

4.1. Which AFL teams are most supported?

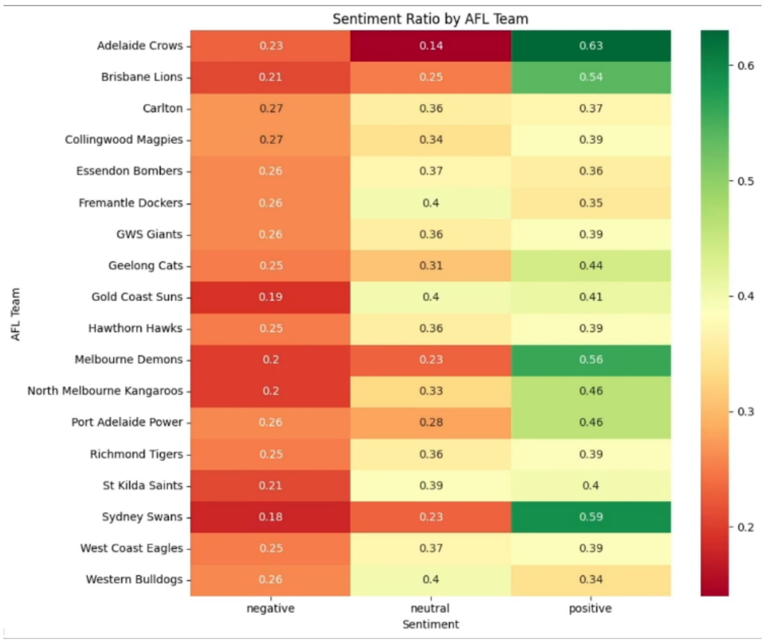


Figure 4.1: Sentiment Ratio by AFL Team

The Australian Football League (AFL) is one of the most iconic and influential professional sports leagues in Australia, with its related topics widely discussed across social media platforms and generating a high level of public interest and enthusiasm. In this analysis, a total of 365,645 social media posts were sampled from an Elasticsearch database, with content collected from platforms such as Mastodon, Reddit, and Bluesky. Each post contained keywords, tags, sentiment labels, sentiment scores, and createdAt.

To identify AFL-related content, a team alias dictionary was constructed to associate all 18 AFL teams with their commonly used nicknames and abbreviations. Based on this dictionary, each post was tagged with one or more teams by matching values in the keywords and tags fields.

Two interactive visualizations were developed in Jupyter Notebook to examine public sentiment toward different teams. he first program generated a pie chart, allowing users to select any AFL team and view the distribution of sentiment labels—positive, neutral, and negative—across related posts. The results indicated that the Adelaide Crows had the highest proportion of positive sentiment, while Carlton and Fremantle showed relatively higher proportions of negative sentiment. In other words, when people mention the Adelaide Crows, they are more likely to use positive language, such as praising their performance or effort. Conversely, a higher proportion of negative sentiment suggests that discussions about teams like Carlton and Fremantle are more frequently accompanied by less favorable expressions. The second visualization was a heatmap that summarized the sentiment composition for each team across all posts. A color gradient ranging from red (indicating negative sentiment) to green (positive sentiment) was applied to highlight differences between teams.

This analysis demonstrates the practicality of combining structured entity recognition with interactive visualization tools to extract meaningful insights from large-scale social media data. The findings offer a clear perspective on how different AFL teams are perceived in online discourse. Notably, the Adelaide Crows emerged as the most positively perceived team in the dataset, suggesting a strong level of public favorability. Future work may incorporate event timelines, match results, or temporal modeling to expand the analysis toward more dynamic, context-aware sentiment tracking.

4.2. How do Australians feel about the rising cost of living and inflation between 2023 and 2025?

This scenario presents the data analysis and key findings related to cost-of-living discussions across Australian cities over the past three years. Our aim is to understand how Australian's sentiment around this topic varies geographically and over time, and what expressions indicate growing public stress.

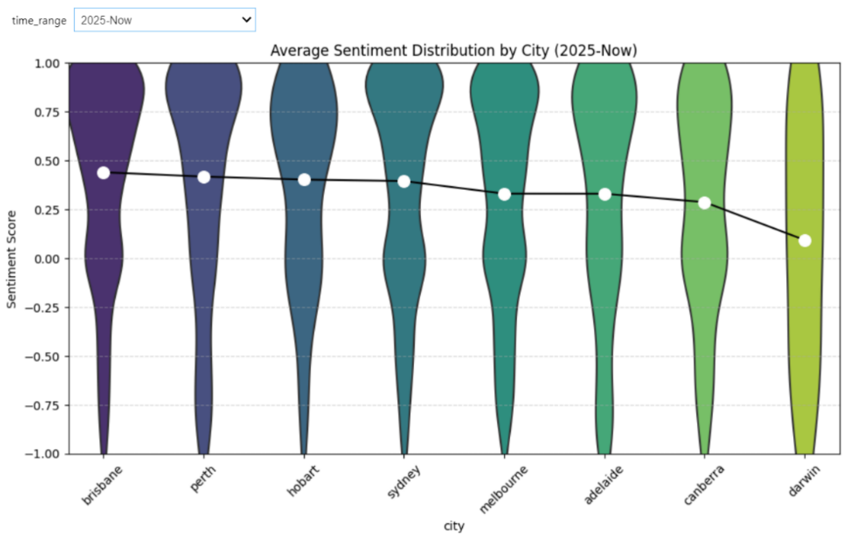


Figure 4.2.1: The distribution of Average Sentiment by City (2025~now)

First, we analysed sentiment distributions by city. From Figure 4.2.1, we found that cities like Melbourne and Sydney consistently show lower average sentiment scores compared to cities such as Brisbane or Perth. This pattern is especially clear in the violin plot, which visualizes the full spread of sentiment across locations. When filtered by year, the sentiment in Melbourne and Sydney not only remains lower but drops significantly in ranking from 2023 to 2025.

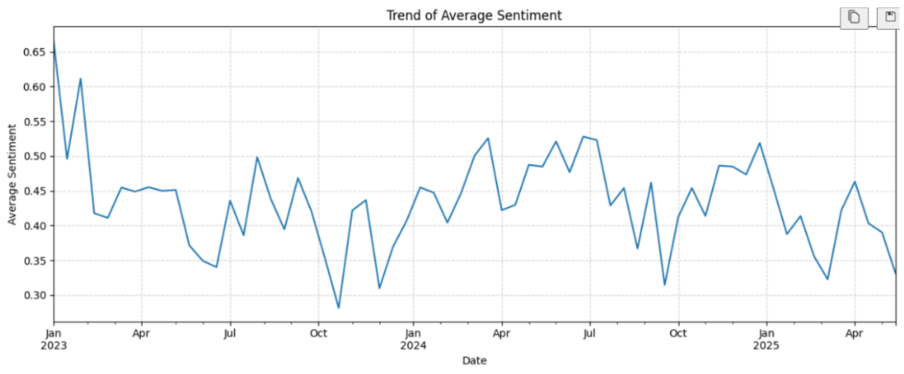


Figure 4.2.2: The overall Trend of the Average Sentiment since 2023

From Figure 4.2.2, the time series line chart of the total average sentiment also confirmed this downward trend. The sentiment score has declined gradually, with particularly sharp drops in late 2023 and in early 2025. These periods probably correspond to national economic or political issues, such as rising rents or cost of living pressures reported in the news.

Although other cities' ranks in the violin plot changes as well, Melbourne and Sydney are clearly leading the negative sentiment trend. This could be attributed to their high population density, housing stress, and large proportion of posts in our dataset (that might bias the result). In support of this, keyword analysis showed that stress-related terms in highly negative posts, such as "police", "crime", "murder", and "rent", which frequently appear in the lowest sentiment (around -1) group. Meanwhile, posts with more positive sentiments are likely to use words such as "love", "enjoy", and "friends". This contrast reflects a split in public sentiment around affordability and safety.

In terms of platforms, Reddit is by far the most dominant source of data, especially in large cities. Therefore, it covers the widest range of sentiments, from highly positive to extremely negative. This makes Reddit an important channel for tracking changes in public opinion. In contrast, other platforms like Mastodon and Bluesky contributed fewer posts and narrower sentiment ranges.

Overall, the data clearly suggests a growing dissatisfaction with the cost of living in Australia, particularly in major cities like Melbourne and Sydney. The current situations of declining sentiment and strong use of negatively charged keywords is a reminder of growing concerns about cost of living in Australia.

4.3. Who is likely to win the upcoming Australian election based on public sentiment?

Although the result of the Australian federal election is already known, we wanted to look back and study how people talked about the election on social media. This helps us understand what people were feeling, which parties they supported, and how this changed in different places. We collected more than 2 million posts from three platforms: Reddit, Mastodon, and Bluesky. The post contained two regions would be counted as two posts when drawing the spatial maps. To find posts about the election, we used a keyword list with names of political parties, leaders, and election topics like "vote" and "campaign."

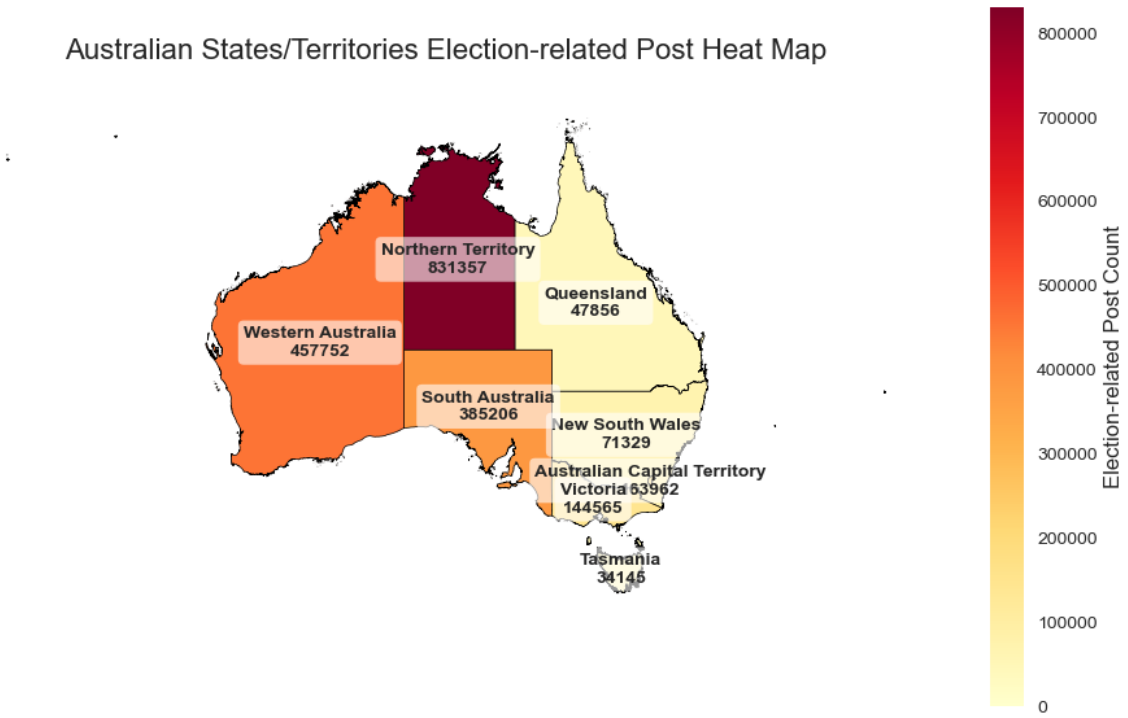


Figure 4.3.1 Australian States Election-related Post Heat Map

Figure 4.3.1 is a heat map of post counts by state. It shows where most of the election-related posts came from. Northern Territory (NT) had the highest number of posts, followed by Western Australia (WA) and South Australia (SA). These places had a lot of online discussion. Tasmania (TAS) and Queensland (QLD) had fewer posts. This may mean people in NT and WA were more active online about politics.

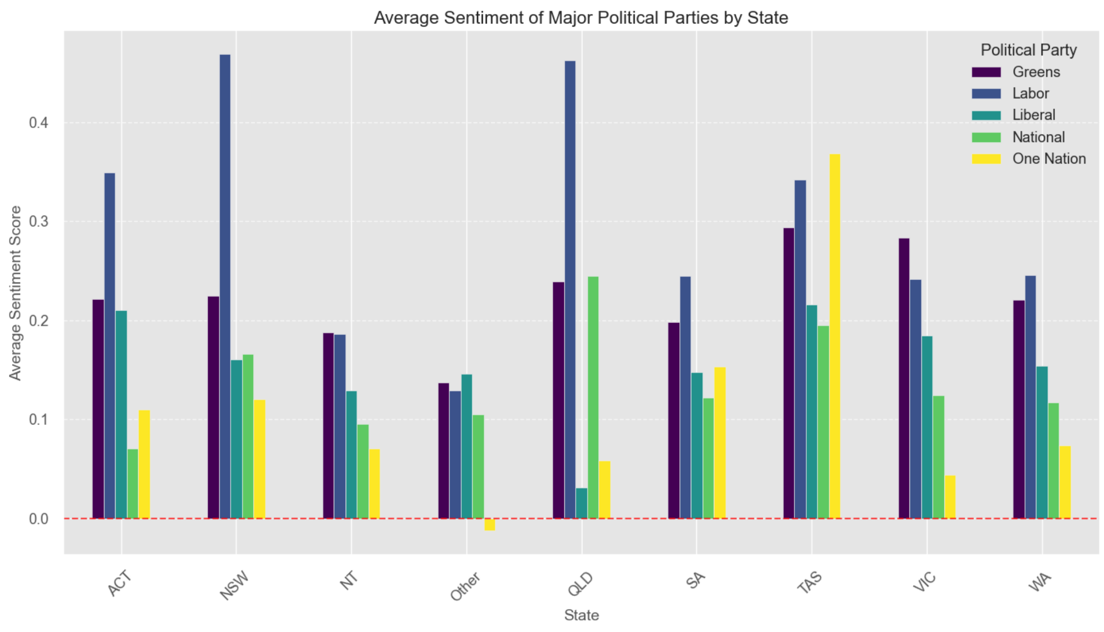


Figure 4.3.2: Average Sentiment of Major Political Parties by State

Figure 4.3.2 is a bar chart of average sentiment for each party in each state. It shows how people felt about the main political parties: Greens, Labor, Liberal, National, and One Nation. In most states, the Labor Party received the most positive sentiment. In TAS, One Nation had the highest average sentiment. This is very different from its scores in other states, where it was much less supported.

These results help us understand how people in different parts of Australia talked about the election. The Labor Party had the highest average sentiment in most states. Even in states where it was not number one, it was still in the top three out of five main parties. This shows that many people spoke about Labor in a positive way. This matches the real result of the 2025 federal election, where Labor won again and Anthony Albanese stayed as Prime Minister.

5. Improvement

Although our current analysis provides valuable insights into public sentiment around the cost of living in Australia, there are several limitations that could be addressed to enhance the accuracy of the analysis.

First, improvements can be made in the data collection process. The current pipeline relies on APIs that are slow and limited (such as allowing only 1,000 posts per topic), particularly on Reddit. As a result, we were partially forced to rely on existing external datasets, which often lack posts with relatively recent publishing times. To resolve this, future improvements could consider adopting a distributed crawling mechanism, which would represent a fundamentally different method of data acquisition.

Second, the sentiment analysis method used in this project is somewhat too lightweight [8]. The results generated by this library may not be rigorous enough. The current sentiment scoring model lacks the nuanced understanding required to interpret informal language, sarcasm, or complex opinions. Future versions of the system could incorporate large language models (LLMs) such as BERT-based classifiers. These models offer context-aware sentiment prediction based on their pre-trained knowledge, which can more accurately capture the tone and emotional content of human language.

Another area that requires improvement is data diversity and platform coverage. Although Reddit provided a rich and diverse source of content, it dominated the dataset due to its open API access. Platforms such as X, Instagram, and Facebook offer vast amounts of relevant user content, but their data access is very expensive. While Mastodon and Bluesky serve more localized user bases and better reflect Australian voices, their post volumes remain low (especially in small cities), making it difficult to ensure balanced regional representation. In the future, exploring alternative sources such as public forums, news comment sections, or even mobile app reviews could help expand both the scope and depth of regional insights.

In summary, enhancing the quality of sentiment classification, improving data acquisition efficiency, and diversifying data sources are key directions that could significantly strengthen the system. These potential improvements would allow for a more comprehensive and reliable understanding of public opinion in Australia.

6. Teamwork and Contribution

Name	Contributions
Josh Feng	Report: 3.1.1 Mastodon, 3.1.4 Post data processing and send to Elasticsearch, 3.2 MRC and NeCTAR research cloud, 3.3 Kubernetes, 3.4 Fission, 3.5 Elasticsearch, 3.8 Redis, 3.9 GitLab, 7 Conclusion. Coding: Mastodon harvesters, post-process, addes and data-filter fission function, deploy function into Fission, design elasticsearch index and redis method.
Fan Yi	Report:1.1 introduction, 1.2 research, 3.7-unit test, 4.4 which team are most support Coding: unit Test, afl scenario analysis
Siyan Pan	Report: 3.1.2 Reddit, 3.6 error handling Coding: exploring Reddit harvesters, exploring frontend scenarios analysis

Shijia Gu	Report: Section 2.1 Scenario, Section 4.2 Cost of Living, Section 5 Improvement Coding: Reddit harvesters, comment scraping, Cost-of-Living Scenario analysis (frontend).
Weiqi Chen	Report: Section 2.2 Functionality, Section 2.3 Frontend, Section 2.4 Backend, Section 3.1.3 Bluesky, Section 4.3 Election Coding: Basic structure of Bluesky Scraping. Election Scenario Analysis.

7. Conclusion

In conclusion, we build and demonstrate a complete social media data collection and analysis system which make the real-time sensing and visualisation of topics connect to Australia. Through integrating multiple platform data sources and building an event-driven architecture based on Kubernetes, we build up automatic data fetching, processing and storage.

The backend system adopts the multi-layer collaboration of Fission, Redis and Elasticsearch which effectively make data decoupling and improve the overall maintainability and scalability. Also in the frontend, user can flexibly explore data under different topics that connect to Australia to support multi-scenarios analysis and fetch the real-time data including AFL, cost of living and election. Of course, our system is not always perfect and there are still some issues that deserve to be optimised, and we have come up with ideas for the future improvements.

We also provided a demonstration video, which demonstrates the architecture of the system as well as the front-end for different scenarios, the link is in appendix 8.1. Also, the source code of the project also provided the link is in appendix 8.2.

8. Appendix

8.1. Video Link

https://www.youtube.com/watch?v=4iBKayKnqRA

8.2. GitLab Link

https://gitlab.unimelb.edu.au/chenhaof/comp90024_team_55

8.3. Kubernetes Cluster Provisioning

```
openstack coe cluster create\
  --cluster-template "kubernetes-v1.31.1-melbourne-qh2-uom-v4" \
  --node-count 3 \
  --master-count 1 \
  --master-flavor "uom.mse.2c9g"\
  --flavor "uom.mse.2c9g"\
  comp90024
```

8.4. Elasticsearch Index: socialplatfrom.yaml

```
settings:
  index:
    number_of_shards: 3
    number_of_replicas: 1

mappings:
  properties:
    platform:
      type: keyword
    version:
      type: float
    fetchedAt:
      type: date
    sentiment:
      type: float
    sentimentLabel:
      type: keyword
    keywords:
      type: keyword
    data:
      properties:
        id:
          type: keyword
        createdAt:
          type: date
        content:
          type: text
        sensitive:
          type: boolean
        favouritesCount:
          type: integer
        repliesCount:
          type: integer
        tags:
          type: keyword
        url:
          type: keyword
        account:
          properties:
            id:
              type: keyword
            username:
              type: keyword
            createdAt:
              type: date
            followersCount/linkKarma:
              type: integer
            followingCount/commentKarma:
              type: integer
```

9. References

[1] Mastodon Australia. (2025). Mastodon Hosted on Mastodon.au. https://mastodon.au/explore

[2] Apis - Mastodon documentation. (2025). Joinmastodon.org. https://docs.joinmastodon.org/api/

[3] Melbourne Research Cloud Documentation. (2019). Unimelb.edu.au. https://docs.cloud.unimelb.edu.au/

[4] ARDC Nectar Research Cloud | ARDC. (n.d.). Https://Ardc.edu.au/. https://ardc.edu.au/services/ardc-nectar-research-cloud/

[5] openstack. (2019). Build the future of Open Infrastructure. OpenStack. https://www.openstack.org/

[6] Fission. (2024, September 3). Fission. Fission. https://fission.io/docs/

[7] Elastic. (n.d.). Elasticsearch: The Official Distributed Search & Analytics Engine. Elastic. https://www.elastic.co/elasticsearch

[8] vaderSentiment. (n.d.). Python Sentiment Analysis Tool Based on VADER. GitHub. https://github.com/cjhutto/vaderSentiment

[9] Bluesky Social. (n.d.). Bsky.app. https://bsky.app/

[10] Reddit. (2005). Reddit. Reddit. https://www.reddit.com/

[11] Kubernetes. (2019). Production-Grade Container Orchestration. Kubernetes.io. https://kubernetes.io/